SpiNNaker 7 Manchester Documentation

Release

Nov 16, 2017

Contents

1 SpiNNUtils **3**
1.1 SpiNNUtils . . . 3
1.1.1 Contents . . . 3
1.1.2 Indices and tables . . . 14

2 SpiNNMachine **15**
2.1 SpiNNMachine . . . 15
2.1.1 Functional Requirements . . . 15
2.1.2 Use Cases . . . 15
2.1.3 Contents . . . 16
2.1.4 Indices and tables . . . 56

3 SpiNNStorageHandlers **59**
3.1 SpiNNStorageHandlers . . . 59
3.1.1 Contents . . . 59
3.1.2 Indices and tables . . . 75

4 PACMAN **77**
4.1 PACMAN . . . 77
4.1.1 Functional Requirements . . . 77
4.1.2 Contents . . . 80
4.1.3 Indices and tables . . . 216

5 SpiNNMan **217**
5.1 SpiNNMan . . . 217
5.1.1 Functional Requirements . . . 217
5.1.2 Use Cases . . . 218
5.1.3 Contents . . . 219
5.1.4 Indices and tables . . . 412

6 DataSpecification **413**
6.1 DataSpecification . . . 413
6.1.1 Functional Requirements . . . 413
6.1.2 Use Cases . . . 414
6.1.3 Contents . . . 414
6.1.4 Indices and tables . . . 489

7 spalloc **491**

8 SpiNNFrontEndCommon **493**
8.1 SpiNNFrontEndCommon 493
8.1.1 Contents 493
8.1.2 Indices and tables 614

9 SpiNNakerGraphFrontEnd **615**

10 sPyNNaker **617**
10.1 sPyNNaker 618
10.1.1 Contents 618
10.1.2 Indices and tables 838

11 sPyNNaker7 **839**

12 spalloc_server **841**

13 Indices and tables **843**

Python Module Index **845**

These pages document the main Manchester python code for the older PyNN 0.7 branch of the SpiNNaker Project which can be found on github

Alternative ways to run SpiNNaker are using the SpiNNakerGraphFrontEnd, or running it with more recent PyNN .

CHAPTER 1

SpiNNUtils

This provides basic utility functions and classes to other parts of SpiNNaker's tooling. Nothing in here knows anything about SpiNNaker functionality.

1.1 SpiNNUtils

1.1.1 Contents

spinn_utilities

spinn_utilities package

Subpackages

spinn_utilities.configs package

Submodules

spinn_utilities.configs.camel_case_config_parser module

class `spinn_utilities.configs.camel_case_config_parser.`**`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

optionxform (*optionstr*)

read (*filenames*)

read_files

spinn_utilities.configs.case_sensitive_parser module

class spinn_utilities.configs.case_sensitive_parser.**CaseSensitiveParser** (*defaults=None*, *dict_type=<class 'collections.OrderedDict'>*, *allow_no_value=False*)

Bases: ConfigParser.RawConfigParser

optionxform (*optionstr*)

spinn_utilities.configs.no_config_found_exception module

exception `spinn_utilities.configs.no_config_found_exception.`**`NoConfigFoundException`**

Bases: `exceptions.Exception`

Throws when an existing Section has an extra config value

spinn_utilities.configs.unexpected_config_exception module

exception `spinn_utilities.configs.unexpected_config_exception.`**`UnexpectedConfigException`**

Bases: `exceptions.Exception`

Throws when an existing Section has an extra config value

Module contents

spinn_utilities.testing package

Submodules

spinn_utilities.testing.log_checker module

`spinn_utilities.testing.log_checker.`**`assert_logs_contains`**(*level_name*, *log_records*, *sub_message*)

`spinn_utilities.testing.log_checker.`**`assert_logs_contains_once`**(*level_name*, *log_records*, *message*)

`spinn_utilities.testing.log_checker.`**`assert_logs_error_contains`**(*log_records*, *sub_message*)

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 bu testfixtures.LogCapture
- **`sub_message`** – String which should be part of an ERROR log

Rasies AssertionError

`spinn_utilities.testing.log_checker.`**`assert_logs_error_not_contains`**(*log_records*, *sub_message*)

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 bu testfixtures.LogCapture
- **`sub_message`** – String which should be part of an ERROR log

Rasies AssertionError

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 bu testfixtures.LogCapture
- **sub_message** – String which should be part of an INFO log

Rasies AssertionError

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

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 bu testfixtures.LogCapture
- **sub_message** – String which should be part of an INFO log

Rasies AssertionError

spinn_utilities.testing.log_checker.**assert_logs_not_contains**(*level_name*, *log_records*, *sub_message*)

Module contents

Submodules

spinn_utilities.abstract_base module

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, ...): ...

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):

...

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)

spinn_utilities.conf_loader module

`spinn_utilities.conf_loader.`**`check_config`**(*config*, *cfg_file*, *validation_config=None*, *default_config=None*)

Checks the config read up to this point to see if it is outdated

Once one difference is found a full reports is generated and an error

raised

Any section specific list as Dead will cause a error

Any section in the defaults should not have extra values. It will never have less as the defaults are in the config

Errors on any values listed as PreviousValues. These are specific values in specific options no longer supported For example old algorithm names

Parameters

- **`config`** – Config as read in up to this point
- **`cfg_file`** – Path of last file read in
- **`validation_config`** – Path containing the validation rules
- **`default_configs`** – List of Paths to defaults

`spinn_utilities.conf_loader.`**`install_cfg_and_IOError`**(*filename*, *defaults*, *config_locations*)

Installs a local config based on the tamplates and thorws an Error

This method is called when no user config is found.

It will create a file in the users home directory based on the defaults.

Then it prints a helpful messages and thros and error with the same message

Parameters

- **`filename`** (*`str`*) – Name under which to save the new config file

- **defaults** (*List[str]*) – List of full paths to the default config files. Each of which MUST have an associated template file with exactly the same path plus .template
- **config_locations** – List of paths the user configs where looked for, Onlty used for the message

raise NoConfigFoundException: Always raised

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

Load the configuration

Parameters **config_parsers** (*list of (str, ConfigParser)*) – The parsers to parse the config with, as a list of (section name, parser); config will only be parsed if the section_name is found in the configuration files already loaded

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

Create the root logger with the given level.

Create filters based on logging levels

spinn_utilities.conf_loader.**outdated_config**(*cfg_file*, *validation_config*, *default_configs*)

Prints why a config file is outdated and raises an exception

Reads a config file by itself (Without others)

Reports errors in this config based on the validation_config and the defaults_configs

Reports any values listed as PreviousValues. These are specific values in specific options no longer supported For example old algorithm names

Checks all sections not defined as UserSections (Default Machine) IE ones the user is expected to change

Any section specific list as Dead will be reported

Any section in the default config is compared reporting any unexpected values reporting the smaller of values non default or values same as default

Any other section is ignored as assumed being used by an extenstion

Parameters

- **cfg_file** – Path to be checked
- **validation_config** – Path containing the validation rules
- **default_configs** – List of Paths to defaults

Returns

spinn_utilities.conf_loader.**read_a_config**(*config*, *cfg_file*, *validation_config=None*, *default_config=None*)

Reads in a config file and then directly its machine_spec_file

Parameters

- **config** – config to do the reading
- **cfg_file** – path to file which should be read in

Returns list of files read including and machine_spec_files

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 of 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

Returns Nothing is returned

Return type None

`binary_paths`

`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, or None if no executable was found in the set of folders

Return type str

spinn_utilities.helpful_functions module

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

Get possible components

Parameters

- **`module`** –
- **`terminator`** –

Return type dict

`spinn_utilities.helpful_functions.`**`set_up_output_application_data_specifics`**(*where_to_write_applic*
max_application_binar
app_id,
n_calls_to_run,
this_run_time_string)

Parameters

- **`where_to_write_application_data_files`** – the location where all app data is by default written to
- **`max_application_binaries_kept`** – The max number of report folders to keep active at any one time
- **`app_id`** – the id used for identifying the simulation on the SpiNNaker machine
- **`n_calls_to_run`** – the counter of how many times run has been called.

- **this_run_time_string** – the time stamp string for this run

Returns the run folder for this simulation to hold app data

`spinn_utilities.helpful_functions.`**`set_up_report_specifics`**(*default_report_file_path*, *max_reports_kept*, *app_id*, *n_calls_to_run*, *this_run_time_string=None*)

Parameters

- **default_report_file_path** – The location where all reports reside
- **max_reports_kept** – The max number of report folders to keep active at any one time
- **app_id** – the id used for identifying the simulation on the SpiNNaker machine
- **n_calls_to_run** – the counter of how many times run has been called.
- **this_run_time_string** – holder for the timestamp for future runs

Returns The folder for this run, the time_stamp

`spinn_utilities.helpful_functions.`**`write_finished_file`**(*app_data_runtime_folder*, *report_default_directory*)

spinn_utilities.log module

class `spinn_utilities.log.`**`ConfiguredFilter`**(*conf*)

Bases: `object`

filter(*record*)

Get the level for the deepest parent, and filter appropriately.

class `spinn_utilities.log.`**`ConfiguredFormatter`**(*conf*)

Bases: `logging.Formatter`

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.

spinn_utilities.logger_utils module

`spinn_utilities.logger_utils.`**`reset`**()

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

spinn_utilities.ordered_set module

class `spinn_utilities.ordered_set.`**`OrderedSet`**(*iterable=None*)

Bases: `_abcoll.MutableSet`

add(*key*)

discard(*key*)

pop(*last=True*)

update(*iterable*)

spinn_utilities.overrides module

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

Bases: object

A decorator for indicating that a method overrides another method in a super class. 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** – 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** – Additional arguments taken by the subclass method over the superclass method e.g. that are to be injected

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

Returns set of python package names

spinn_utilities.package_loader.**load_module**(*name*, *remove_pyc_files=False*, *exclusions=[]*, *gather_errors=True*)

Loads this modules and all its children

Parameters

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

spinn_utilities.package_loader.**load_modules**(*directory*, *prefix*, *remove_pyc_files=False*, *exclusions=[]*, *gather_errors=True*)

Loads 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** – path to check for python files

- **prefix** – package prefix top add to the file name
- **remove_pyc_files** – True if .pyc files should be deleted
- **exclusions** – a list of modules to exclude
- **gather_errors** – True if errors should be gathered, False to report on first error

spinn_utilities.progress_bar module

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

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** – 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

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):
  ...
NameError: name 'evil_func' is not defined
```

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

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

spinn_utilities.timer module

class spinn_utilities.timer.**Timer**

Bases: object

A timer used for performance measurements

measured_interval

start_timing()

take_sample()

Module contents

1.1.2 Indices and tables

- genindex
- modindex
- search

SpiNNUtils_github

SpiNNUtils_individual_docs

CHAPTER 2

SpiNNMachine

This package is used to provide a Python representation of a SpiNNaker machine

2.1 SpiNNMachine

A python abstraction of a SpiNNaker Machine. The main functionality is provided by `spinn_machine.Machine`.

2.1.1 Functional Requirements

- Create a machine which represents the current state of a machine, in terms of the available chips, cores on the chips, SDRAM available, routable links between chips and available routing entries.
- Create a machine which represents an abstract ideal machine.
  - There can only be one chip in the machine with given x, y coordinates
  - There can only be one processor in each chip with a given processor id
  - There can only be one link in the router of each chip with a given id
- Add a chip to a given machine to represent an external device.
  - A chip with the same x, y coordinates must not already exist in the machine
- Add a link to a router of a given chip to represent a connection to an external device.
  - A link with the given id must not already exist in the chip
- Create a representation of a multicast routing entry to be shared between modules that deal with routing entries.

2.1.2 Use Cases

- `Machine` is returned as a representation of the current state of a machine.

- `Machine` is used as an outline of a machine on which a simulation will be run e.g. for placement of executables and/or finding routes between placed executables.
- `Machine` is extended to add a virtual `Chip` on the machine representing an external peripheral connected to the machine directly via a link from a chip, so that routes can be directed to and from the external peripheral
- `MulticastRoutingEntry` is returned in a list of entries, which indicate the current set of routing entries within a multicast routing table on a chip on the machine.
- `MulticastRoutingEntry` is sent in a list of routing entries to set up routing on a chip on the machine.

2.1.3 Contents

spinn_machine

spinn_machine package

Subpackages

spinn_machine.link_data_objects package

Submodules

spinn_machine.link_data_objects.abstract_link_data module

class `spinn_machine.link_data_objects.abstract_link_data.`**`AbstractLinkData`**(*connected_chip_x*, *connected_chip_y*, *connected_link*, *board_address*)

Bases: `object`

Data object for spinnaker links

`board_address`
property method for board address

`connected_chip_x`
property method for connected chip x

`connected_chip_y`
property method for connected chip y

`connected_link`
property for connected link

spinn_machine.link_data_objects.fpga_link_data module

class spinn_machine.link_data_objects.fpga_link_data.**FPGALinkData**(*fpga_link_id*, *fpga_id*, *connected_chip_x*, *connected_chip_y*, *connected_link*, *board_address*)

Bases: *spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData*

Data object for FPGA links

fpga_id

fpga_link_id

spinn_machine.link_data_objects.spinnaker_link_data module

class spinn_machine.link_data_objects.spinnaker_link_data.**SpinnakerLinkData**(*spinnaker_link_id*, *connected_chip_x*, *connected_chip_y*, *connected_link*, *board_address*)

Bases: *spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData*

Data object for spinnaker links

spinnaker_link_id

Get the id of the spinnaker link

Module contents

class spinn_machine.link_data_objects.**AbstractLinkData**(*connected_chip_x*, *connected_chip_y*, *connected_link*, *board_address*)

Bases: object

Data object for spinnaker links

board_address

property method for board address

connected_chip_x

property method for connected chip x

connected_chip_y

property method for connected chip y

connected_link

property for connected link

class spinn_machine.link_data_objects.**FPGALinkData**(*fpga_link_id*, *fpga_id*, *connected_chip_x*, *connected_chip_y*, *connected_link*, *board_address*)

Bases: *spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData*

Data object for FPGA links

fpga_id

fpga_link_id

class spinn_machine.link_data_objects.**SpinnakerLinkData**(*spinnaker_link_id*, *connected_chip_x*, *connected_chip_y*, *connected_link*, *board_address*)

Bases: *spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData*

Data object for spinnaker links

spinnaker_link_id

Get the id of the spinnaker link

spinn_machine.tags package

Submodules

spinn_machine.tags.abstract_tag module

class spinn_machine.tags.abstract_tag.**AbstractTag**(*board_address*, *tag*, *port*)

Bases: object

board_address

The board address of the tag

port

The port of the tag

tag

The tag id of the tag

spinn_machine.tags.iptag module

class spinn_machine.tags.iptag.**IPTag**(*board_address*, *destination_x*, *destination_y*, *tag*, *ip_address*, *port=None*, *strip_sdp=False*, *traffic_identifier='DEFAULT'*)

Bases: *spinn_machine.tags.abstract_tag.AbstractTag*

Used to hold data that is contained within an IPTag

Parameters

- **board_address** (*str or None*) – The ip address of the board on which the tag is allocated
- **destination_x** (*int*) – The x-coordinate where users of this tag should send packets to
- **destination_y** (*int*) – The y-coordinate where users of this tag should send packets to

- **tag** (*int*) – The tag of the SDP packet
- **ip_address** (*str*) – The IP address to which SDP packets with the tag will be sent
- **port** (*int or None if not yet assigned*) – The port to which the SDP packets with the tag will be sent
- **strip_sdp** (*bool*) – Indicates whether the SDP header should be removed
- **traffic_identifier** (*str*) – the identifier for traffic transmitted using this tag

Raises **None** – No known exceptions are raised

destination_x
The x-coordinate where users of this tag should send packets to

destination_y
The y-coordinate where users of this tag should send packets to

ip_address
Return the IP address of the tag

strip_sdp
Return if the SDP header is to be stripped

traffic_identifier
The identifier of traffic using this tag

spinn_machine.tags.reverse_iptag module

class spinn_machine.tags.reverse_iptag.**ReverseIPTag**(*board_address*, *tag*, *port*, *destination_x*, *destination_y*, *destination_p*, *sdp_port=1*)

Bases: *spinn_machine.tags.abstract_tag.AbstractTag*

Used to hold data that is contained within an IPTag

Parameters

- **board_address** (*str or None*) – The ip address of the board on which the tag is allocated
- **tag** (*int*) – The tag of the SDP packet
- **port** (*int*) – The UDP port on which SpiNNaker will listen for packets
- **destination_x** (*int*) – The x-coordinate of the chip to send packets to
- **destination_y** (*int*) – The y-coordinate of the chip to send packets to
- **destination_p** (*int*) – The id of the processor to send packets to
- **sdp_port** (*int*) – The optional port number to use for SDP packets that are formed on the machine (default is 1)

Raises **None** – No known exceptions are raised

destination_p
The destination p for a reverse ip tag

destination_x
The destination x for a reverse ip tag

destination_y
The destination y for a reverse ip tag

sdp_port
The SDP port of the tag

Module contents

class spinn_machine.tags.**AbstractTag**(*board_address*, *tag*, *port*)
Bases: object

board_address
The board address of the tag

port
The port of the tag

tag
The tag id of the tag

class spinn_machine.tags.**IPTag**(*board_address*, *destination_x*, *destination_y*, *tag*, *ip_address*, *port=None*, *strip_sdp=False*, *traffic_identifier='DEFAULT'*)
Bases: *spinn_machine.tags.abstract_tag.AbstractTag*

Used to hold data that is contained within an IPTag

Parameters

- **board_address** (*str or None*) – The ip address of the board on which the tag is allocated
- **destination_x** (*int*) – The x-coordinate where users of this tag should send packets to
- **destination_y** (*int*) – The y-coordinate where users of this tag should send packets to
- **tag** (*int*) – The tag of the SDP packet
- **ip_address** (*str*) – The IP address to which SDP packets with the tag will be sent
- **port** (*int or None if not yet assigned*) – The port to which the SDP packets with the tag will be sent
- **strip_sdp** (*bool*) – Indicates whether the SDP header should be removed
- **traffic_identifier** (*str*) – the identifier for traffic transmitted using this tag

Raises **None** – No known exceptions are raised

destination_x
The x-coordinate where users of this tag should send packets to

destination_y
The y-coordinate where users of this tag should send packets to

ip_address
Return the IP address of the tag

strip_sdp
Return if the SDP header is to be stripped

traffic_identifier
The identifier of traffic using this tag

class spinn_machine.tags.**ReverseIPTag**(*board_address*, *tag*, *port*, *destination_x*, *destination_y*, *destination_p*, *sdp_port=1*)

Bases: *spinn_machine.tags.abstract_tag.AbstractTag*

Used to hold data that is contained within an IPTag

Parameters

- **board_address** (*str or None*) – The ip address of the board on which the tag is allocated
- **tag** (*int*) – The tag of the SDP packet
- **port** (*int*) – The UDP port on which SpiNNaker will listen for packets
- **destination_x** (*int*) – The x-coordinate of the chip to send packets to
- **destination_y** (*int*) – The y-coordinate of the chip to send packets to
- **destination_p** (*int*) – The id of the processor to send packets to
- **sdp_port** (*int*) – The optional port number to use for SDP packets that are formed on the machine (default is 1)

Raises **None** – No known exceptions are raised

destination_p
The destination p for a reverse ip tag

destination_x
The destination x for a reverse ip tag

destination_y
The destination y for a reverse ip tag

sdp_port
The SDP port of the tag

spinn_machine.utilities package

Submodules

spinn_machine.utilities.ordered_set module

class spinn_machine.utilities.ordered_set.**OrderedSet**(*iterable=None*)

Bases: _abcoll.MutableSet

add(*key*)

discard(*key*)

pop(*last=True*)

update(*iterable*)

spinn_machine.utilities.progress_bar module

class `spinn_machine.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

`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`** – 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

Module contents

Submodules

spinn_machine.chip module

class `spinn_machine.chip.`**`Chip`**(*x*, *y*, *processors*, *router*, *sdram*, *nearest_ethernet_x*, *nearest_ethernet_y*, *ip_address=None*, *virtual=False*, *tag_ids=OrderedSet([1, 2, 3, 4, 5, 6, 7])*)

Bases: `object`

Represents a chip with a number of cores, an amount of SDRAM shared between the cores, and a router. The chip is iterable over the processors providing (processor_id, processor) where:

- processor_id is the id of a processor
- processor is the processor with processor_id

Parameters

- **`x`** (*`int`*) – the x-coordinate of the chip's position in the two-dimensional grid of chips
- **`y`** (*`int`*) – the y-coordinate of the chip's position in the two-dimensional grid of chips

- **processors** (iterable of *spinn_machine.Processor*) – an iterable of processor objects
- **router** (*spinn_machine.Router*) – a router for the chip
- **sdram** (*spinn_machine.SDRAM*) – an SDRAM for the chip
- **ip_address** (*str*) – the IP address of the chip or None if no Ethernet attached
- **virtual** (*bool*) – boolean which defines if this chip is a virtual one
- **tag_ids** (*iterable of int*) – Id to identify the chip for SDP
- **nearest_ethernet_x** (*int or None*) – the nearest Ethernet x coord
- **nearest_ethernet_y** (*int or None*) – the nearest Ethernet y coord

Raises *spinn_machine.exceptions.SpinnMachineAlreadyExistsException* – If processors contains any two processors with the same processor_id

IPTAG_IDS = OrderedSet([1, 2, 3, 4, 5, 6, 7])

get_first_none_monitor_processor()

Get the first processor in the list which is not a monitor core

Returns a processor

get_processor_with_id(*processor_id*)

Return the processor with the specified id or None if the processor does not exist. Also implemented as __contains__(processor_id)

Parameters **processor_id** (*int*) – the id of the processor to return

Returns the processor with the specified id or None if no such processor

Return type *spinn_machine.Processor*

Raises **None** – does not raise any known exceptions

ip_address

The IP address of the chip

Returns IP address of the chip, or None if there is no Ethernet connected to the chip

Return type str

Raises **None** – does not raise any known exceptions

is_processor_with_id(*processor_id*)

Determines if a processor with the given id exists in the chip. Also implemented as __getitem__(processor_id)

Parameters **processor_id** (*int*) – the processor id to check for

Returns True or False based on the existence of the processor

Return type bool

Raises **None** – does not raise any known exceptions

n_processors

The total number of processors

n_user_processors

The total number of processors that are not monitors

nearest_ethernet_x

the x coord of the nearest Ethernet chip

Returns the x coord of the nearest Ethernet chip

Return type int

Raises **None** – does not raise any known exceptions

nearest_ethernet_y

the y coord of the nearest Ethernet chip

Returns the y coord of the nearest Ethernet chip

Return type int

Raises **None** – does not raise any known exceptions

processors

An iterable of available processors

Returns iterable of processors

Return type iterable of :py:class:spinn_machine.Processor‘

Raises **None** – does not raise any known exceptions

reserve_a_system_processor()

This method should ONLY be called via Machine.reserve_system_processors

Sets one of the none monitor processors as a system processor

Updates n_user_processors

Returns The id of the processor reserved, or None if no processor could be found

Return type int or None

router

The router object associated with the chip

Returns router associated with the chip

Return type *spinn_machine.Router*

Raises **None** – does not raise any known exceptions

sdram

The sdram associated with the chip

Returns sdram associated with the chip

Return type *spinn_machine.SDRAM*

Raises **None** – does not raise any known exceptions

tag_ids

The tag ids supported by this chip

Returns the set of ids.

Raises **None** – this method does not raise any exception

virtual

boolean which defines if the chip is virtual or not

Returns if the chip is virtual

Return type boolean

Raises **None** – this method does not raise any known exceptions

x

The x-coordinate of the chip in the two-dimensional grid of chips

Returns the x-coordinate of the chip

Return type int

Raises **None** – does not raise any known exceptions

y

The y-coordinate of the chip in the two-dimensional grid of chips

Returns the y-coordinate of the chip

Return type int

Raises **None** – does not raise any known exceptions

spinn_machine.core_subset module

class `spinn_machine.core_subset.`**`CoreSubset`**(*x*, *y*, *processor_ids*)

Bases: `object`

Represents a subset of the cores on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip
- **processor_ids** (*iterable of int*) – An iterable of processor ids on the chip

add_processor(*processor_id*)

Adds a processor id to this subset

Parameters **processor_id** (*int*) – A processor id

Returns Nothing is returned

Return type None

processor_ids

The subset of processor ids on the chip

Returns An iterable of processor ids

Return type iterable of int

x

The x-coordinate of the chip

Returns The x-coordinate

Return type int

y

The y-coordinate of the chip

Returns The y-coordinate

Return type int

spinn_machine.core_subsets module

class `spinn_machine.core_subsets.`**`CoreSubsets`**(*core_subsets=None*)

Bases: `object`

Represents a group of CoreSubsets, with a maximum of one per chip

Parameters **`core_subsets`** (iterable of *`spinn_machine.CoreSubset`*) – An iterable of cores for each desired chip

`add_core_subset`(*core_subset*)

Add a core subset to the set

Parameters **`core_subset`** (*`spinn_machine.CoreSubset`*) – The core subset to add

Returns Nothing is returned

Return type None

`add_processor`(*x*, *y*, *processor_id*)

Add a processor on a given chip to the set

Parameters

- **`x`** (*`int`*) – The x-coordinate of the chip
- **`y`** (*`int`*) – The y-coordinate of the chip
- **`processor_id`** (*`int`*) – A processor id

Returns Nothing is returned

Return type None

`core_subsets`

The one-per-chip subsets

Returns Iterable of core subsets

Return type iterable of *`spinn_machine.CoreSubset`*

`get_core_subset_for_chip`(*x*, *y*)

Get the core subset for a chip

Parameters

- **`x`** (*`int`*) – The x-coordinate of a chip
- **`y`** (*`int`*) – The y-coordinate of a chip

Returns The core subset of a chip, which will be empty if not added

Return type *`spinn_machine.CoreSubset`*

`is_chip`(*x*, *y*)

Determine if the chip with coordinates (x, y) is in the subset

Parameters

- **`x`** (*`int`*) – The x-coordinate of a chip
- **`y`** (*`int`*) – The y-coordinate of a chip

Returns True if the chip with coordinates (x, y) is in the subset

Return type bool

is_core (*x*, *y*, *processor_id*)

Determine if there is a chip with coordinates (x, y) in the subset, which has a core with the given id in the subset

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip
- **processor_id** (*int*) – The id of a core

Returns True if there is a chip with coordinates (x, y) in the subset, which has a core with the given id in the subset

spinn_machine.exceptions module

exception spinn_machine.exceptions.**SpinnMachineAlreadyExistsException** (*item*, *value*)

Bases: *spinn_machine.exceptions.SpinnMachineException*

Indicates that something already exists of which there can only be one

Parameters

- **item** (*str*) – The item of which there is already one of
- **value** (*str*) – The value of the item

item

The item of which there is already one

value

The value of the item

exception spinn_machine.exceptions.**SpinnMachineException**

Bases: exceptions.Exception

A generic exception which all other exceptions extend

exception spinn_machine.exceptions.**SpinnMachineInvalidParameterException** (*parameter*, *value*, *problem*)

Bases: *spinn_machine.exceptions.SpinnMachineException*

Indicates that there is a problem with a parameter value

Parameters

- **parameter** (*str*) – The name of the parameter that has an invalid value
- **value** (*str*) – The value of the parameter that is invalid
- **problem** (*str*) – The reason for the exception

parameter

The name of the parameter

problem

The problem with the setting of the parameter

value

The value of the parameter

spinn_machine.fixed_route_entry module

class `spinn_machine.fixed_route_entry.`**`FixedRouteEntry`**(*processor_ids*, *link_ids*)

Bases: `object`

`link_ids`

The destination link ids

Returns An iterable of link ids

Return type iterable of int

`processor_ids`

The destination processor ids

Returns An iterable of processor ids

Return type iterable of int

spinn_machine.link module

a Link in a spinnaker machine

class `spinn_machine.link.`**`Link`**(*source_x*, *source_y*, *source_link_id*, *destination_x*, *destination_y*, *multicast_default_from*, *multicast_default_to*)

Bases: `object`

Represents a directional link between chips in the machine

Parameters

- **`source_x`** (*int*) – The x-coordinate of the source chip of the link
- **`source_y`** (*int*) – The y-coordinate of the source chip of the link
- **`source_link_id`** (*int*) – The id of the link in the source chip
- **`destination_x`** (*int*) – The x-coordinate of the destination chip of the link
- **`destination_y`** (*int*) – The y-coordinate of the destination chip of the link
- **`multicast_default_from`** (*int*) – Traffic received on the link identified by multicast_default_from will be sent to the link herein defined if no entry is present in the multicast routing table. On SpiNNaker chips, multicast_default_from is usually the same as multicast_default_to. None if no such default exists, or the link does not exist.
- **`multicast_default_to`** (*int*) – Traffic received on the link herein defined will be sent to the link identified by multicast_default_from if no entry is present in the multicast routing table. On SpiNNaker chips, multicast_default_to is usually the same as multicast_default_from. None if no such link exists, or the link does not exist.

Raises **`None`** – No known exceptions are raised

`destination_x`

The x-coordinate of the destination chip of this link

Returns The x-coordinate

Return type int

`destination_y`

The y-coordinate of the destination chip of this link

Returns The y-coordinate

Return type int

multicast_default_from
The id of the link for which this link is the default

Returns The id of a link, or None if no such link

Return type int

multicast_default_to
The id of the link to which to send default routed multicast

Returns The id of a link, or None if no such link

Return type int

source_link_id
The id of the link on the source chip

Returns The link id

Return type int

source_x
The x-coordinate of the source chip of this link

Returns The x-coordinate

Return type int

source_y
The y-coordinate of the source chip of this link

Returns The y-coordinate

Return type int

spinn_machine.machine module

class spinn_machine.machine.**Machine**(*chips*, *boot_x*, *boot_y*)
Bases: object

A Representation of a Machine with a number of Chips. Machine is also iterable, providing ((x, y), chip) where:

- x is the x-coordinate of a chip
- y is the y-coordinate of a chip
- chip is the chip with the given x, y coordinates

Parameters

- **chips** (iterable of *spinn_machine.Chip*) – An iterable of chips in the machine
- **boot_x** (*int*) – The x-coordinate of the chip used to boot the machine
- **boot_y** (*int*) – The y-coordinate of the chip used to boot the machine

Raises *spinn_machine.exceptions.SpinnMachineAlreadyExistsException* – If any two chips have the same x and y coordinates

BOARD_48_CHIP_GAPS = set([(2, 7), (2, 6), (7, 0), (7, 1), (6, 1), (0, 4), (6, 0), (0, 7), (0, 6), (1, 5), (1, 6), (5, 0), (1, 7), (3, 7), (7

LINK_ADD_TABLE = [(1, 0), (1, 1), (0, 1), (-1, 0), (-1, -1), (0, -1)]

MAX_BANDWIDTH_PER_ETHERNET_CONNECTED_CHIP = 2560

add_chip(*chip*)

Add a chip to the machine

Parameters **chip** (*spinn_machine.Chip*) – The chip to add to the machine

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If a chip with the same x and y coordinates already exists

add_chips(*chips*)

Add some chips to the machine

Parameters **chips** (iterable of *spinn_machine.Chip*) – an iterable of chips

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If a chip with the same x and y coordinates as one being added already exists

add_fpga_links(*version_no*)

Add FPGA links that are on a given machine depending on the version of the board.

Parameters **version_no** – which version of board to use

add_spinnaker_links(*version_no*)

Add SpiNNaker links that are on a given machine depending on the version of the board.

Parameters **version_no** – which version of board to use

boot_chip

The chip used to boot the machine

Return type *py:class:spinn_machine.Chip*

boot_x

The x-coordinate of the chip used to boot the machine

Return type int

boot_y

The y-coordinate of the chip used to boot the machine

Return type int

chip_coordinates

An iterable of chip coordinates in the machine

Returns An iterable of chip coordinates

Return type iterable of (int, int)

chips

An iterable of chips in the machine

Returns An iterable of chips

Return type iterable of *spinn_machine.Chip*

Raises **None** – does not raise any known exceptions

cores_and_link_output_string()
Get a string detailing the number of cores and links

Return type str

ethernet_connected_chips
The chips in the machine that have an Ethernet connection

Returns An iterable of chips

Return type iterable of *spinn_machine.Chip*

get_chip_at(*x*, *y*)
Get the chip at a specific (x, y) location. Also implemented as __getitem__((x, y))

Parameters

- **x** (*int*) – the x-coordinate of the requested chip
- **y** (*int*) – the y-coordinate of the requested chip

Returns the chip at the specified location, or None if no such chip

Return type *spinn_machine.Chip*

Raises **None** – does not raise any known exceptions

static **get_chip_over_link**(*x*, *y*, *link*, *width*, *height*)
Get the x and y coordinates of the chip over the given link

Parameters

- **x** – The x coordinate of the chip to start from
- **y** – The y coordinate of the chip to start from
- **link** – The id of the link to traverse, between 0 and 5
- **width** – The width of the machine being considered
- **height** – The height of the machine being considered

get_chips_on_board(*chip*)
Get the chips that are on the same board as the given chip

Parameters **chip** – The chip to find other chips on the same board as

Returns An iterable of (x, y) coordinates of chips on the same board

get_cores_and_link_count()
Get the number of cores and links from the machine

Returns tuple of (n_cores, n_links)

get_fpga_link_with_id(*fpga_id*, *fpga_link_id*, *board_address=None*)
Get an FPGA link data item that corresponds to the FPGA and FPGA link for a given board address.

Parameters

- **fpga_id** (*int*) – the id of the FPGA that the data is going through. Refer to technical document located here for more detail: https://drive.google.com/file/d/0B9312BuJXntlVWowQlJ3RE8wWVE
- **fpga_link_id** (*int*) – the link id of the FPGA. Refer to technical document located here for more detail: https://drive.google.com/file/d/0B9312BuJXntlVWowQlJ3RE8wWVE
- **board_address** (*str*) – the board address that this spinnaker link is associated with

Return type *spinn_machine.link_data_objects.FPGALinkData*

Returns the given FPGA link object or None if no such link

get_spinnaker_link_with_id(*spinnaker_link_id*, *board_address=None*)

Get a spinnaker link with a given id

Parameters

- **spinnaker_link_id** (*int*) – The id of the link
- **board_address** (*str or None*) – the board address that this spinnaker link is associated with

Returns The spinnaker link data or None if no link

Return type *spinn_machine.link_data_objects.SpinnakerLinkData*

has_wrap_arounds

If the machine has wrap around links

Returns True if wrap around links exist, false otherwise

Return type bool

is_chip_at(*x*, *y*)

Determine if a chip exists at the given coordinates. Also implemented as __contains__((x, y))

Parameters

- **x** (*int*) – x location of the chip to test for existence
- **y** (*int*) – y location of the chip to test for existence

Returns True if the chip exists, False otherwise

Return type bool

Raises **None** – does not raise any known exceptions

is_link_at(*x*, *y*, *link*)

Determine if a link exists at the given coordinates

Parameters

- **x** (*int*) – The x location of the chip to test the link of
- **y** (*int*) – The y location of the chip to test the link of
- **link** (*int*) – The link to test the existence of

max_chip_x

The maximum x-coordinate of any chip in the board

Returns The maximum x-coordinate

Return type int

max_chip_y

The maximum y-coordinate of any chip in the board

Returns The maximum y-coordinate

Return type int

maximum_user_cores_on_chip

The maximum number of user cores on any chip

n_chips

reserve_system_processors()

Sets one of the none monitor system processors as a system processor on every Chip

Updates maximum_user_cores_on_chip

Returns A CoreSubsets of reserved cores, and a list of (x, y) of chips where a non-system core was not available

Return type (*spinn_machine.CoreSubsets*, list of (int, int))

spinnaker_links

The set of spinnaker links in the machine

Returns An iterable of spinnaker links

Return type iterable of *spinn_machine.link_data_objects.SpinnakerLinkData*

total_available_user_cores

provides total number of cores on the machine which are not monitor cores

Returns total

Return type int

total_cores

provides total number of cores on the machine, includes monitors

Returns total

Return type int

spinn_machine.multicast_routing_entry module

class spinn_machine.multicast_routing_entry.**MulticastRoutingEntry**(*routing_entry_key*, *mask*, *processor_ids*, *link_ids*, *defaultable*)

Bases: object

Represents an entry in a multicast routing table

Parameters

- **routing_entry_key** (*int*) – The routing key_combo
- **mask** (*int*) – The route key_combo mask
- **processor_ids** (*iterable of int*) – The destination processor ids
- **link_ids** (*iterable of int*) – The destination link ids
- **defaultable** (*bool*) – if this entry is defaultable (it receives packets from its directly opposite route position)

Raises *spinn_machine.exceptions.SpinnMachineAlreadyExistsException* –

- If processor_ids contains the same id more than once
- If link_ids contains the same id more than once

defaultable

if this entry is a defaultable entry

Returns the bool that represents if a entry is defaultable or not

Return type bool

link_ids
The destination link ids

Returns An iterable of link ids

Return type iterable of int

mask
The routing mask

Returns The routing mask

Return type int

merge (*other_entry*)
Merges together two multicast routing entries. The entry to merge must have the same key and mask. The merge will join the processor ids and link ids from both the entries. This could be used to add a new destination to an existing route in a routing table. It is also possible to use the add (+) operator or the or (|) operator with the same effect.

Parameters **other_entry** (*MulticastRoutingEntry*) – The multicast entry to merge with this entry

Returns A new multicast routing entry with merged destinations

Return type *MulticastRoutingEntry*

Raises ***spinn_machine.exceptions.SpinnMachineInvalidParameterException*** – If the key and mask of the other entry do not match

processor_ids
The destination processor ids

Returns An iterable of processor ids

Return type iterable of int

routing_entry_key
The routing key

Returns The routing key

Return type int

spinn_machine.processor module

class spinn_machine.processor.**Processor** (*processor_id*, *clock_speed=200000000*, *is_monitor=False*, *dtcm_available=65536*)

Bases: object

A processor object included in a chip

Parameters

- **processor_id** (*int*) – id of the processor in the chip
- **clock_speed** (*int*) – The number of cpu cycles per second of the processor
- **is_monitor** (*bool*) – Determines if the processor is considered the monitor processor, and so should not be otherwise allocated

- **dtcm_available** (*int*) – Data Tightly Coupled Memory available

Raises *spinn_machine.exceptions.SpinnMachineInvalidParameterException* – If the clock speed is negative

CLOCK_SPEED = 200000000

DTCM_AVAILABLE = 65536

clock_speed

The clock speed of the processor in cycles per second

Returns The clock speed in cycles per second

Return type int

clone_as_system_processor()

Creates a clone of this processor but changing it to a system processor.

The current implementation does not distinguish between monitor processors and reinjector ones but could do so at a later stage.

Returns A new Processor with the same properties INCLUDING id except now set as a System processor

Return type *spinn_machine.Processor*

cpu_cycles_available

The number of cpu cycles available from this processor per ms

Returns the number of cpu cycles available on this processor

Return type int

dtcm_available

The amount of DTCM available on this processor

Returns the amount of DTCM available on this processor

Return type int

is_monitor

Determines if the processor is the monitor, and therefore not to be allocated

WARNING: Currently rejection processeors are also marked as monitors

Returns True if the processor is the monitor, False otherwise

Return type bool

processor_id

The id of the processor

Returns id of the processor

Return type int

spinn_machine.router module

class spinn_machine.router.**Router**(*links*, *emergency_routing_enabled=False*, *clock_speed=157286400*, *n_available_multicast_entries=1024*)

Bases: object

Represents a router of a chip, with a set of available links. The router is iterable over the links, providing (source_link_id, link) where:

- source_link_id is the id of a link
- link is the link with id source_link_id

Parameters

- **links** (iterable of *spinn_machine.Link*) – iterable of links
- **emergency_routing_enabled** (*bool*) – Determines if the router emergency routing is operating
- **clock_speed** (*int*) – The router clock speed in cycles per second
- **n_available_multicast_entries** (*int*) – The number of entries available in the routing table

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If any two links have the same source_link_id

ROUTER_DEFAULT_AVAILABLE_ENTRIES = 1024

ROUTER_DEFAULT_CLOCK_SPEED = 157286400

add_link (*link*)

Add a link to the router of the chip

Parameters **link** (*spinn_machine.Link*) – The link to be added

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If another link already exists with the same source_link_id

clock_speed

The clock speed of the router in cycles per second

Returns The clock speed in cycles per second

Return type int

Raises **None** – does not raise any known exceptions

static **convert_routing_table_entry_to_spinnaker_route** (*routing_table_entry*)

Convert a routing table entry represented in software to a binary routing table entry usable on the machine

Parameters **routing_table_entry** (*spinn_machine.MulticastRoutingEntry*) – The entry to convert

Return type int

emergency_routing_enabled

Indicator of whether emergency routing is enabled

Returns True if emergency routing is enabled, False otherwise

Return type bool

Raises **None** – does not raise any known exceptions

get_link (*source_link_id*)

Get the link with the given id, or None if no such link. Also implemented as __getitem__(source_link_id)

Parameters **source_link_id** (*int*) – The id of the link to find

Returns The link, or None if no such link

Return type *spinn_machine.Link*

Raises **None** – No known exceptions are raised

get_neighbouring_chips_coords()

Utility method to convert links into x and y coordinates

Returns iterable list of destination coordinates in x and y dict

Return type iterable of dict

is_link(*source_link_id*)

Determine if there is a link with id source_link_id. Also implemented as __contains__(source_link_id)

Parameters **source_link_id** (*int*) – The id of the link to find

Returns True if there is a link with the given id, False otherwise

Return type bool

Raises **None** – No known exceptions are raised

links

The available links of this router

Returns an iterable of available links

Return type iterable of *spinn_machine.Link*

Raises **None** – does not raise any known exceptions

n_available_multicast_entries

The number of available multicast entries in the routing tables

Returns The number of available entries

Return type int

Raises **None** – does not raise any known exceptions

spinn_machine.sdram module

class spinn_machine.sdram.**SDRAM**(*size=122683392*)

Bases: object

Represents the properties of the SDRAM of a chip in the machine

Parameters **size** (*int*) – the space available in SDRAM

DEFAULT_SDRAM_BYTES = 122683392

size

The SDRAM available for user applications

Returns The space available in bytes

Return type int

spinn_machine.spinnaker_triad_geometry module

class `spinn_machine.spinnaker_triad_geometry.`**`SpiNNakerTriadGeometry`**(*triad_width*, *triad_height*, *roots*, *centre*)

Bases: `object`

Geometry of a "triad" of SpiNNaker boards

The geometry is defined by the arguments to the constructor; the standard arrangement can be obtained from get_spinn5_geometry

Note that the geometry defines what a Triad is in terms of the dimensions of a triad and where the Ethernet chips occur in the triad

Parameters

- **`triad_width`** (*`int`*) – triad_width of a triad in chips
- **`triad_height`** (*`int`*) – triad_height of a triad in chips
- **`roots`** (*`list of (int, int)`*) – locations of the Ethernet connected chips
- **`centre`** (*`(float, float)`*) – the distance from each Ethernet chip to the centre of the hexagon

`get_ethernet_chip_coordinates`(*x*, *y*, *width*, *height*, *root_x=0*, *root_y=0*)

Get the coordinates of a chip's local Ethernet connected chip according to this triad geometry object

> **Warning:** `local_eth_coord()` will always produce the coordinates of the Ethernet-connected SpiNNaker chip on the same SpiNN-5 board as the supplied chip. This chip may not actually be working.

Parameters

- **`x`** (*`int`*) – x-coordinate of the chip to find the nearest Ethernet of
- **`y`** (*`int`*) – y-coordinate of the chip to find the nearest Ethernet of
- **`width`** (*`int`*) – width of the spinnaker machine (must be a multiple of the triad width of this geometry)
- **`height`** (*`int`*) – height of the spinnaker machine (must be a multiple of the triad height of this geometry)
- **`root_x`** (*`int`*) – x-coordinate of the boot chip (default 0, 0)
- **`root_y`** (*`int`*) – y-coordinate of the boot chip (default 0, 0)

Returns The coordinates of the closest Ethernet chip

Return type (int, int)

`get_local_chip_coordinate`(*x*, *y*, *root_x=0*, *root_y=0*)

Get the coordinates of a chip on its board of a multi-board system relative to the Ethernet chip of the board.

> **Note:** This function assumes the system is constructed from SpiNN-5 boards

Parameters

- **x** (*int*) – The x-coordinate of the chip to find the location of
- **y** (*int*) – The y-coordinate of the chip to find the location of
- **root_x** (*int*) – The x-coordinate of the boot chip (default 0, 0)
- **root_y** (*int*) – The y-coordinate of the boot chip (default 0, 0)

Returns the coordinates of the chip relative to its board

Return type (int, int)

static **get_spinn5_geometry**()

Get the geometry object for a SpiNN-5 arrangement of boards

Returns a SpiNNakerTriadGeometry object.

spinn5_triad_geometry = None

spinn_machine.virtual_machine module

class spinn_machine.virtual_machine.**VirtualMachine**(*width=None, height=None, with_wrap_arounds=False, version=None, n_cpus_per_chip=18, with_monitors=True, sdram_per_chip=None, down_chips=None, down_cores=None, down_links=None*)

Bases: *spinn_machine.machine.Machine*

A Virtual SpiNNaker machine

Parameters

- **width** (*int*) – the width of the virtual machine in chips
- **height** (*int*) – the height of the virtual machine in chips
- **with_wrap_arounds** (*bool*) – bool defining if wrap around links exist
- **version** (*int*) – the version id of a board; if None, a machine is created with the correct dimensions, otherwise the machine will be a single board of the given version
- **n_cpus_per_chip** (*int*) – The number of CPUs to put on each chip
- **with_monitors** (*bool*) – True if CPU 0 should be marked as a monitor
- **sdram_per_chip** (*int or None*) – The amount of SDRAM to give to each chip

add_chip(*chip*)

Add a chip to the machine

Parameters **chip** (*spinn_machine.Chip*) – The chip to add to the machine

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If a chip with the same x and y coordinates already exists

chip_coordinates

chips

get_chip_at (*x*, *y*)

get_cores_and_link_count ()

is_chip_at (*x*, *y*)

is_link_at (*x*, *y*, *link*)

maximum_user_cores_on_chip

n_chips

reserve_system_processors ()

Sets one of the none monitor system processors as a system processor on every Chip

Updates maximum_user_cores_on_chip

:rtype None

Module contents

A python abstraction of a SpiNNaker Machine. The main functionality is provided by `spinn_machine.Machine`.

Functional Requirements

- Create a machine which represents the current state of a machine, in terms of the available chips, cores on the chips, SDRAM available, routable links between chips and available routing entries.
- Create a machine which represents an abstract ideal machine.
  - There can only be one chip in the machine with given x, y coordinates
  - There can only be one processor in each chip with a given processor id
  - There can only be one link in the router of each chip with a given id
- Add a chip to a given machine to represent an external device.
  - A chip with the same x, y coordinates must not already exist in the machine
- Add a link to a router of a given chip to represent a connection to an external device.
  - A link with the given id must not already exist in the chip
- Create a representation of a multicast routing entry to be shared between modules that deal with routing entries.

Use Cases

- `Machine` is returned as a representation of the current state of a machine.
- `Machine` is used as an outline of a machine on which a simulation will be run e.g. for placement of executables and/or finding routes between placed executables.
- `Machine` is extended to add a virtual `Chip` on the machine representing an external peripheral connected to the machine directly via a link from a chip, so that routes can be directed to and from the external peripheral
- `MulticastRoutingEntry` is returned in a list of entries, which indicate the current set of routing entries within a multicast routing table on a chip on the machine.
- `MulticastRoutingEntry` is sent in a list of routing entries to set up routing on a chip on the machine.

class `spinn_machine.Chip`(*x*, *y*, *processors*, *router*, *sdram*, *nearest_ethernet_x*, *nearest_ethernet_y*, *ip_address=None*, *virtual=False*, *tag_ids=OrderedSet([1, 2, 3, 4, 5, 6, 7])*)

Bases: `object`

Represents a chip with a number of cores, an amount of SDRAM shared between the cores, and a router. The chip is iterable over the processors providing (processor_id, processor) where:

- processor_id is the id of a processor
- processor is the processor with processor_id

Parameters

- **x** (*int*) – the x-coordinate of the chip's position in the two-dimensional grid of chips
- **y** (*int*) – the y-coordinate of the chip's position in the two-dimensional grid of chips
- **processors** (iterable of *spinn_machine.Processor*) – an iterable of processor objects
- **router** (*spinn_machine.Router*) – a router for the chip
- **sdram** (*spinn_machine.SDRAM*) – an SDRAM for the chip
- **ip_address** (*str*) – the IP address of the chip or None if no Ethernet attached
- **virtual** (*bool*) – boolean which defines if this chip is a virtual one
- **tag_ids** (*iterable of int*) – Id to identify the chip for SDP
- **nearest_ethernet_x** (*int or None*) – the nearest Ethernet x coord
- **nearest_ethernet_y** (*int or None*) – the nearest Ethernet y coord

Raises *spinn_machine.exceptions.SpinnMachineAlreadyExistsException* – If processors contains any two processors with the same processor_id

IPTAG_IDS = OrderedSet([1, 2, 3, 4, 5, 6, 7])

get_first_none_monitor_processor()

Get the first processor in the list which is not a monitor core

Returns a processor

get_processor_with_id(*processor_id*)

Return the processor with the specified id or None if the processor does not exist. Also implemented as __contains__(processor_id)

Parameters **processor_id** (*int*) – the id of the processor to return

Returns the processor with the specified id or None if no such processor

Return type *spinn_machine.Processor*

Raises **None** – does not raise any known exceptions

ip_address

The IP address of the chip

Returns IP address of the chip, or None if there is no Ethernet connected to the chip

Return type str

Raises **None** – does not raise any known exceptions

is_processor_with_id(*processor_id*)

Determines if a processor with the given id exists in the chip. Also implemented as __getitem__(processor_id)

Parameters **processor_id** (*int*) – the processor id to check for

Returns True or False based on the existence of the processor

Return type bool

Raises **None** – does not raise any known exceptions

n_processors

The total number of processors

n_user_processors

The total number of processors that are not monitors

nearest_ethernet_x

the x coord of the nearest Ethernet chip

Returns the x coord of the nearest Ethernet chip

Return type int

Raises **None** – does not raise any known exceptions

nearest_ethernet_y

the y coord of the nearest Ethernet chip

Returns the y coord of the nearest Ethernet chip

Return type int

Raises **None** – does not raise any known exceptions

processors

An iterable of available processors

Returns iterable of processors

Return type iterable of :py:class:spinn_machine.Processor‘

Raises **None** – does not raise any known exceptions

reserve_a_system_processor()

This method should ONLY be called via Machine.reserve_system_processors

Sets one of the none monitor processors as a system processor

Updates n_user_processors

Returns The id of the processor reserved, or None if no processor could be found

Return type int or None

router

The router object associated with the chip

Returns router associated with the chip

Return type *spinn_machine.Router*

Raises **None** – does not raise any known exceptions

sdram

The sdram associated with the chip

Returns sdram associated with the chip

Return type *spinn_machine.SDRAM*

Raises **None** – does not raise any known exceptions

tag_ids

The tag ids supported by this chip

Returns the set of ids.

Raises **None** – this method does not raise any exception

virtual

boolean which defines if the chip is virtual or not

Returns if the chip is virtual

Return type boolean

Raises **None** – this method does not raise any known exceptions

x

The x-coordinate of the chip in the two-dimensional grid of chips

Returns the x-coordinate of the chip

Return type int

Raises **None** – does not raise any known exceptions

y

The y-coordinate of the chip in the two-dimensional grid of chips

Returns the y-coordinate of the chip

Return type int

Raises **None** – does not raise any known exceptions

class spinn_machine.**CoreSubset**(*x*, *y*, *processor_ids*)

Bases: object

Represents a subset of the cores on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip
- **processor_ids** (*iterable of int*) – An iterable of processor ids on the chip

add_processor(*processor_id*)

Adds a processor id to this subset

Parameters **processor_id** (*int*) – A processor id

Returns Nothing is returned

Return type None

processor_ids

The subset of processor ids on the chip

Returns An iterable of processor ids

Return type iterable of int

x

The x-coordinate of the chip

Returns The x-coordinate

Return type int

y

The y-coordinate of the chip

Returns The y-coordinate

Return type int

class spinn_machine.**CoreSubsets**(*core_subsets=None*)

Bases: object

Represents a group of CoreSubsets, with a maximum of one per chip

Parameters **core_subsets** (iterable of *spinn_machine.CoreSubset*) – An iterable of cores for each desired chip

add_core_subset(*core_subset*)

Add a core subset to the set

Parameters **core_subset** (*spinn_machine.CoreSubset*) – The core subset to add

Returns Nothing is returned

Return type None

add_processor(*x*, *y*, *processor_id*)

Add a processor on a given chip to the set

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip
- **processor_id** (*int*) – A processor id

Returns Nothing is returned

Return type None

core_subsets

The one-per-chip subsets

Returns Iterable of core subsets

Return type iterable of *spinn_machine.CoreSubset*

get_core_subset_for_chip(*x*, *y*)

Get the core subset for a chip

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip

Returns The core subset of a chip, which will be empty if not added

Return type *spinn_machine.CoreSubset*

is_chip(*x*, *y*)

Determine if the chip with coordinates (x, y) is in the subset

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip

Returns True if the chip with coordinates (x, y) is in the subset

Return type bool

is_core (*x*, *y*, *processor_id*)

Determine if there is a chip with coordinates (x, y) in the subset, which has a core with the given id in the subset

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip
- **processor_id** (*int*) – The id of a core

Returns True if there is a chip with coordinates (x, y) in the subset, which has a core with the given id in the subset

class spinn_machine.**FixedRouteEntry** (*processor_ids*, *link_ids*)

Bases: object

link_ids

The destination link ids

Returns An iterable of link ids

Return type iterable of int

processor_ids

The destination processor ids

Returns An iterable of processor ids

Return type iterable of int

class spinn_machine.**Link** (*source_x*, *source_y*, *source_link_id*, *destination_x*, *destination_y*, *multicast_default_from*, *multicast_default_to*)

Bases: object

Represents a directional link between chips in the machine

Parameters

- **source_x** (*int*) – The x-coordinate of the source chip of the link
- **source_y** (*int*) – The y-coordinate of the source chip of the link
- **source_link_id** (*int*) – The id of the link in the source chip
- **destination_x** (*int*) – The x-coordinate of the destination chip of the link
- **destination_y** (*int*) – The y-coordinate of the destination chip of the link
- **multicast_default_from** (*int*) – Traffic received on the link identified by multicast_default_from will be sent to the link herein defined if no entry is present in the multicast routing table. On SpiNNaker chips, multicast_default_from is usually the same as multicast_default_to. None if no such default exists, or the link does not exist.

- **multicast_default_to** (*int*) – Traffic received on the link herein defined will be sent to the link identified by multicast_default_from if no entry is present in the multicast routing table. On SpiNNaker chips, multicast_default_to is usually the same as multicast_default_from. None if no such link exists, or the link does not exist.

Raises **None** – No known exceptions are raised

destination_x

The x-coordinate of the destination chip of this link

Returns The x-coordinate

Return type int

destination_y

The y-coordinate of the destination chip of this link

Returns The y-coordinate

Return type int

multicast_default_from

The id of the link for which this link is the default

Returns The id of a link, or None if no such link

Return type int

multicast_default_to

The id of the link to which to send default routed multicast

Returns The id of a link, or None if no such link

Return type int

source_link_id

The id of the link on the source chip

Returns The link id

Return type int

source_x

The x-coordinate of the source chip of this link

Returns The x-coordinate

Return type int

source_y

The y-coordinate of the source chip of this link

Returns The y-coordinate

Return type int

class spinn_machine.**Machine**(*chips*, *boot_x*, *boot_y*)

Bases: object

A Representation of a Machine with a number of Chips. Machine is also iterable, providing ((x, y), chip) where:

- x is the x-coordinate of a chip
- y is the y-coordinate of a chip
- chip is the chip with the given x, y coordinates

Parameters

- **chips** (iterable of *spinn_machine.Chip*) – An iterable of chips in the machine
- **boot_x** (*int*) – The x-coordinate of the chip used to boot the machine
- **boot_y** (*int*) – The y-coordinate of the chip used to boot the machine

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If any two chips have the same x and y coordinates

BOARD_48_CHIP_GAPS = set([(2, 7), (2, 6), (7, 0), (7, 1), (6, 1), (0, 4), (6, 0), (0, 7), (0, 6), (1, 5), (1, 6), (5, 0), (1, 7), (3, 7), (7

LINK_ADD_TABLE = [(1, 0), (1, 1), (0, 1), (-1, 0), (-1, -1), (0, -1)]

MAX_BANDWIDTH_PER_ETHERNET_CONNECTED_CHIP = 2560

add_chip(*chip*)

Add a chip to the machine

Parameters **chip** (*spinn_machine.Chip*) – The chip to add to the machine

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If a chip with the same x and y coordinates already exists

add_chips(*chips*)

Add some chips to the machine

Parameters **chips** (iterable of *spinn_machine.Chip*) – an iterable of chips

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If a chip with the same x and y coordinates as one being added already exists

add_fpga_links(*version_no*)

Add FPGA links that are on a given machine depending on the version of the board.

Parameters **version_no** – which version of board to use

add_spinnaker_links(*version_no*)

Add SpiNNaker links that are on a given machine depending on the version of the board.

Parameters **version_no** – which version of board to use

boot_chip

The chip used to boot the machine

Return type *py:class:spinn_machine.Chip*

boot_x

The x-coordinate of the chip used to boot the machine

Return type int

boot_y

The y-coordinate of the chip used to boot the machine

Return type int

chip_coordinates
An iterable of chip coordinates in the machine

Returns An iterable of chip coordinates

Return type iterable of (int, int)

chips
An iterable of chips in the machine

Returns An iterable of chips

Return type iterable of *spinn_machine.Chip*

Raises **None** – does not raise any known exceptions

cores_and_link_output_string()
Get a string detailing the number of cores and links

Return type str

ethernet_connected_chips
The chips in the machine that have an Ethernet connection

Returns An iterable of chips

Return type iterable of *spinn_machine.Chip*

get_chip_at(*x*, *y*)
Get the chip at a specific (x, y) location. Also implemented as __getitem__((x, y))

Parameters

- **x** (*int*) – the x-coordinate of the requested chip
- **y** (*int*) – the y-coordinate of the requested chip

Returns the chip at the specified location, or None if no such chip

Return type *spinn_machine.Chip*

Raises **None** – does not raise any known exceptions

static **get_chip_over_link**(*x*, *y*, *link*, *width*, *height*)
Get the x and y coordinates of the chip over the given link

Parameters

- **x** – The x coordinate of the chip to start from
- **y** – The y coordinate of the chip to start from
- **link** – The id of the link to traverse, between 0 and 5
- **width** – The width of the machine being considered
- **height** – The height of the machine being considered

get_chips_on_board(*chip*)
Get the chips that are on the same board as the given chip

Parameters **chip** – The chip to find other chips on the same board as

Returns An iterable of (x, y) coordinates of chips on the same board

get_cores_and_link_count()
Get the number of cores and links from the machine

Returns tuple of (n_cores, n_links)

get_fpga_link_with_id(*fpga_id*, *fpga_link_id*, *board_address=None*)

Get an FPGA link data item that corresponds to the FPGA and FPGA link for a given board address.

Parameters

- **fpga_id** (*int*) – the id of the FPGA that the data is going through. Refer to technical document located here for more detail: https://drive.google.com/file/d/0B9312BuJXntlVWowQlJ3RE8wWVE
- **fpga_link_id** (*int*) – the link id of the FPGA. Refer to technical document located here for more detail: https://drive.google.com/file/d/0B9312BuJXntlVWowQlJ3RE8wWVE
- **board_address** (*str*) – the board address that this spinnaker link is associated with

Return type *spinn_machine.link_data_objects.FPGALinkData*

Returns the given FPGA link object or None if no such link

get_spinnaker_link_with_id(*spinnaker_link_id*, *board_address=None*)

Get a spinnaker link with a given id

Parameters

- **spinnaker_link_id** (*int*) – The id of the link
- **board_address** (*str or None*) – the board address that this spinnaker link is associated with

Returns The spinnaker link data or None if no link

Return type *spinn_machine.link_data_objects.SpinnakerLinkData*

has_wrap_arounds

If the machine has wrap around links

Returns True if wrap around links exist, false otherwise

Return type bool

is_chip_at(*x*, *y*)

Determine if a chip exists at the given coordinates. Also implemented as __contains__((x, y))

Parameters

- **x** (*int*) – x location of the chip to test for existence
- **y** (*int*) – y location of the chip to test for existence

Returns True if the chip exists, False otherwise

Return type bool

Raises **None** – does not raise any known exceptions

is_link_at(*x*, *y*, *link*)

Determine if a link exists at the given coordinates

Parameters

- **x** (*int*) – The x location of the chip to test the link of
- **y** (*int*) – The y location of the chip to test the link of
- **link** (*int*) – The link to test the existence of

max_chip_x

The maximum x-coordinate of any chip in the board

Returns The maximum x-coordinate

Return type int

max_chip_y

The maximum y-coordinate of any chip in the board

Returns The maximum y-coordinate

Return type int

maximum_user_cores_on_chip

The maximum number of user cores on any chip

n_chips

reserve_system_processors()

Sets one of the none monitor system processors as a system processor on every Chip

Updates maximum_user_cores_on_chip

Returns A CoreSubsets of reserved cores, and a list of (x, y) of chips where a non-system core was not available

Return type (*spinn_machine.CoreSubsets*, list of (int, int))

spinnaker_links

The set of spinnaker links in the machine

Returns An iterable of spinnaker links

Return type iterable of *spinn_machine.link_data_objects.SpinnakerLinkData*

total_available_user_cores

provides total number of cores on the machine which are not monitor cores

Returns total

Return type int

total_cores

provides total number of cores on the machine, includes monitors

Returns total

Return type int

class spinn_machine.**MulticastRoutingEntry**(*routing_entry_key*, *mask*, *processor_ids*, *link_ids*, *defaultable*)

Bases: object

Represents an entry in a multicast routing table

Parameters

- **routing_entry_key** (*int*) – The routing key_combo
- **mask** (*int*) – The route key_combo mask
- **processor_ids** (*iterable of int*) – The destination processor ids
- **link_ids** (*iterable of int*) – The destination link ids
- **defaultable** (*bool*) – if this entry is defaultable (it receives packets from its directly opposite route position)

Raises *spinn_machine.exceptions.SpinnMachineAlreadyExistsException* –

- If processor_ids contains the same id more than once
- If link_ids contains the same id more than once

defaultable
if this entry is a defaultable entry

Returns the bool that represents if a entry is defaultable or not

Return type bool

link_ids
The destination link ids

Returns An iterable of link ids

Return type iterable of int

mask
The routing mask

Returns The routing mask

Return type int

merge(*other_entry*)
Merges together two multicast routing entries. The entry to merge must have the same key and mask. The merge will join the processor ids and link ids from both the entries. This could be used to add a new destination to an existing route in a routing table. It is also possible to use the add (+) operator or the or (|) operator with the same effect.

Parameters **other_entry** (*MulticastRoutingEntry*) – The multicast entry to merge with this entry

Returns A new multicast routing entry with merged destinations

Return type *MulticastRoutingEntry*

Raises ***spinn_machine.exceptions.SpinnMachineInvalidParameterException*** – If the key and mask of the other entry do not match

processor_ids
The destination processor ids

Returns An iterable of processor ids

Return type iterable of int

routing_entry_key
The routing key

Returns The routing key

Return type int

class spinn_machine.**Processor**(*processor_id*, *clock_speed=200000000*, *is_monitor=False*, *dtcm_available=65536*)

Bases: object

A processor object included in a chip

Parameters

- **processor_id** (*int*) – id of the processor in the chip
- **clock_speed** (*int*) – The number of cpu cycles per second of the processor

- **is_monitor** (*bool*) – Determines if the processor is considered the monitor processor, and so should not be otherwise allocated
- **dtcm_available** (*int*) – Data Tightly Coupled Memory available

Raises *spinn_machine.exceptions.SpinnMachineInvalidParameterException* – If the clock speed is negative

CLOCK_SPEED = 200000000

DTCM_AVAILABLE = 65536

clock_speed

The clock speed of the processor in cycles per second

Returns The clock speed in cycles per second

Return type int

clone_as_system_processor()

Creates a clone of this processor but changing it to a system processor.

The current implementation does not distinguish between monitor processors and reinjector ones but could do so at a later stage.

Returns A new Processor with the same properties INCLUDING id except now set as a System processor

Return type *spinn_machine.Processor*

cpu_cycles_available

The number of cpu cycles available from this processor per ms

Returns the number of cpu cycles available on this processor

Return type int

dtcm_available

The amount of DTCM available on this processor

Returns the amount of DTCM available on this processor

Return type int

is_monitor

Determines if the processor is the monitor, and therefore not to be allocated

WARNING: Currently rejection processeors are also marked as monitors

Returns True if the processor is the monitor, False otherwise

Return type bool

processor_id

The id of the processor

Returns id of the processor

Return type int

class spinn_machine.**Router**(*links*, *emergency_routing_enabled=False*, *clock_speed=157286400*, *n_available_multicast_entries=1024*)

Bases: object

Represents a router of a chip, with a set of available links. The router is iterable over the links, providing (source_link_id, link) where:

- source_link_id is the id of a link

- link is the link with id source_link_id

Parameters

- **links** (iterable of *spinn_machine.Link*) – iterable of links
- **emergency_routing_enabled** (*bool*) – Determines if the router emergency routing is operating
- **clock_speed** (*int*) – The router clock speed in cycles per second
- **n_available_multicast_entries** (*int*) – The number of entries available in the routing table

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If any two links have the same source_link_id

ROUTER_DEFAULT_AVAILABLE_ENTRIES = 1024

ROUTER_DEFAULT_CLOCK_SPEED = 157286400

add_link (*link*)

Add a link to the router of the chip

Parameters **link** (*spinn_machine.Link*) – The link to be added

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If another link already exists with the same source_link_id

clock_speed

The clock speed of the router in cycles per second

Returns The clock speed in cycles per second

Return type int

Raises **None** – does not raise any known exceptions

static **convert_routing_table_entry_to_spinnaker_route** (*routing_table_entry*)

Convert a routing table entry represented in software to a binary routing table entry usable on the machine

Parameters **routing_table_entry** (*spinn_machine.MulticastRoutingEntry*) – The entry to convert

Return type int

emergency_routing_enabled

Indicator of whether emergency routing is enabled

Returns True if emergency routing is enabled, False otherwise

Return type bool

Raises **None** – does not raise any known exceptions

get_link (*source_link_id*)

Get the link with the given id, or None if no such link. Also implemented as __getitem__(source_link_id)

Parameters **source_link_id** (*int*) – The id of the link to find

Returns The link, or None if no such link

Return type *spinn_machine.Link*

Raises **None** – No known exceptions are raised

get_neighbouring_chips_coords()

Utility method to convert links into x and y coordinates

Returns iterable list of destination coordinates in x and y dict

Return type iterable of dict

is_link(*source_link_id*)

Determine if there is a link with id source_link_id. Also implemented as __contains__(source_link_id)

Parameters **source_link_id** (*int*) – The id of the link to find

Returns True if there is a link with the given id, False otherwise

Return type bool

Raises **None** – No known exceptions are raised

links

The available links of this router

Returns an iterable of available links

Return type iterable of *spinn_machine.Link*

Raises **None** – does not raise any known exceptions

n_available_multicast_entries

The number of available multicast entries in the routing tables

Returns The number of available entries

Return type int

Raises **None** – does not raise any known exceptions

class spinn_machine.**SDRAM**(*size=122683392*)

Bases: object

Represents the properties of the SDRAM of a chip in the machine

Parameters **size** (*int*) – the space available in SDRAM

DEFAULT_SDRAM_BYTES = 122683392

size

The SDRAM available for user applications

Returns The space available in bytes

Return type int

class spinn_machine.**SpiNNakerTriadGeometry**(*triad_width*, *triad_height*, *roots*, *centre*)

Bases: object

Geometry of a "triad" of SpiNNaker boards

The geometry is defined by the arguments to the constructor; the standard arrangement can be obtained from get_spinn5_geometry

Note that the geometry defines what a Triad is in terms of the dimensions of a triad and where the Ethernet chips occur in the triad

Parameters

- **triad_width** (*int*) – triad_width of a triad in chips

- **triad_height** (*int*) – triad_height of a triad in chips
- **roots** (*list of (int, int)*) – locations of the Ethernet connected chips
- **centre** (*(float, float)*) – the distance from each Ethernet chip to the centre of the hexagon

get_ethernet_chip_coordinates(*x*, *y*, *width*, *height*, *root_x=0*, *root_y=0*)

Get the coordinates of a chip's local Ethernet connected chip according to this triad geometry object

> **Warning:** `local_eth_coord()` will always produce the coordinates of the Ethernet-connected SpiNNaker chip on the same SpiNN-5 board as the supplied chip. This chip may not actually be working.

Parameters

- **x** (*int*) – x-coordinate of the chip to find the nearest Ethernet of
- **y** (*int*) – y-coordinate of the chip to find the nearest Ethernet of
- **width** (*int*) – width of the spinnaker machine (must be a multiple of the triad width of this geometry)
- **height** (*int*) – height of the spinnaker machine (must be a multiple of the triad height of this geometry)
- **root_x** (*int*) – x-coordinate of the boot chip (default 0, 0)
- **root_y** (*int*) – y-coordinate of the boot chip (default 0, 0)

Returns The coordinates of the closest Ethernet chip

Return type (int, int)

get_local_chip_coordinate(*x*, *y*, *root_x=0*, *root_y=0*)

Get the coordinates of a chip on its board of a multi-board system relative to the Ethernet chip of the board.

> **Note:** This function assumes the system is constructed from SpiNN-5 boards

Parameters

- **x** (*int*) – The x-coordinate of the chip to find the location of
- **y** (*int*) – The y-coordinate of the chip to find the location of
- **root_x** (*int*) – The x-coordinate of the boot chip (default 0, 0)
- **root_y** (*int*) – The y-coordinate of the boot chip (default 0, 0)

Returns the coordinates of the chip relative to its board

Return type (int, int)

static **get_spinn5_geometry**()

Get the geometry object for a SpiNN-5 arrangement of boards

Returns a SpiNNakerTriadGeometry object.

spinn5_triad_geometry = None

class spinn_machine.**VirtualMachine**(*width=None*, *height=None*, *with_wrap_arounds=False*, *version=None*, *n_cpus_per_chip=18*, *with_monitors=True*, *sdram_per_chip=None*, *down_chips=None*, *down_cores=None*, *down_links=None*)

Bases: *spinn_machine.machine.Machine*

A Virtual SpiNNaker machine

Parameters

- **width** (*int*) – the width of the virtual machine in chips
- **height** (*int*) – the height of the virtual machine in chips
- **with_wrap_arounds** (*bool*) – bool defining if wrap around links exist
- **version** (*int*) – the version id of a board; if None, a machine is created with the correct dimensions, otherwise the machine will be a single board of the given version
- **n_cpus_per_chip** (*int*) – The number of CPUs to put on each chip
- **with_monitors** (*bool*) – True if CPU 0 should be marked as a monitor
- **sdram_per_chip** (*int or None*) – The amount of SDRAM to give to each chip

add_chip(*chip*)

Add a chip to the machine

Parameters **chip** (*spinn_machine.Chip*) – The chip to add to the machine

Returns Nothing is returned

Return type None

Raises ***spinn_machine.exceptions.SpinnMachineAlreadyExistsException*** – If a chip with the same x and y coordinates already exists

chip_coordinates

chips

get_chip_at(*x*, *y*)

get_cores_and_link_count()

is_chip_at(*x*, *y*)

is_link_at(*x*, *y*, *link*)

maximum_user_cores_on_chip

n_chips

reserve_system_processors()

Sets one of the none monitor system processors as a system processor on every Chip

Updates maximum_user_cores_on_chip

:rtype None

2.1.4 Indices and tables

- genindex
- modindex
- search

SpiNNMachine_github

SpiNNMachine_individual_docs

CHAPTER 3

SpiNNStorageHandlers

This package provides classes to handle data storage, both in memory (through a bytearray buffer) and in a file. The file may be permanent or temporary.

3.1 SpiNNStorageHandlers

3.1.1 Contents

spinn_storage_handlers

spinn_storage_handlers package

Subpackages

spinn_storage_handlers.abstract_classes package

Submodules

spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage module

class `spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.`**`AbstractBuffered`**

Bases: `object`

An object that can store and read back buffered data

`close``()`

Closes the data storage

Return type None

Raises *spinn_storage_handlers.exceptions.DataReadException* – If the data storage cannot be closed

eof()

Check if the read pointer is a the end of the data storage

Returns True if the read pointer is at the end of the data storage, False otherwise

Return type bool

read(*data_size*)

Read data from the data storage from the position indicated by the read pointer index

Parameters **data_size** (*int*) – number of bytes to be read

Returns a bytearray containing the data read

Return type bytearray

read_all()

Read all the data contained in the data storage starting from position 0 to the end

Returns a bytearray containing the data read

Return type bytearray

readinto(*data*)

Read some bytes of data from the underlying storage into a pre-defined array. Will block until some bytes are available, but may not fill the array completely.

Parameters **data** (*bytearray*) – The place where the data is to be stored

Returns The number of bytes stored in data

Return type int

Raises **IOError** – If an error occurs reading from the underlying storage

seek_read(*offset*, *whence=0*)

Set the data storage's current read position to the offset

Parameters

- **offset** (*int*) – Position of the read pointer within the buffer
- **whence** – One of: * os.SEEK_SET which means absolute buffer positioning (default) * os.SEEK_CUR which means seek relative to the current read position * os.SEEK_END which means seek relative to the buffer's end

Return type None

seek_write(*offset*, *whence=0*)

Set the data storage's current write position to the offset

Parameters

- **offset** (*int*) – Position of the write pointer within the buffer
- **whence** – One of: * os.SEEK_SET which means absolute buffer positioning (default) * os.SEEK_CUR which means seek relative to the current write position * os.SEEK_END which means seek relative to the buffer's end

Return type None

tell_read()

The current position of the read pointer

Returns The current position of the read pointer

Return type int

tell_write()
The current position of the write pointer

Returns The current position of the write pointer

Return type int

write(*data*)
Store data in the data storage in the position indicated by the write pointer index

Parameters **data** (*bytearray*) – the data to be stored

Return type None

spinn_storage_handlers.abstract_classes.abstract_byte_reader module

class spinn_storage_handlers.abstract_classes.abstract_byte_reader.**AbstractByteReader**
Bases: object

An abstract reader of bytes. Note that due to endianness concerns, the methods of this reader should be used directly for the appropriate data type being read; e.g. an int should be written using read_int rather than calling read_byte 4 times unless a specific endianness is being achieved.

is_at_end()
returns true if the reader is currently at the end of the byte reader

Returns returns true if the reader is currently at the end of the byte reader false otherwise

Return type bool

read_byte()
Reads the next byte

Returns A byte

Return type int

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are no more bytes to read

read_bytes(*size=None*)
Reads an array of bytes

Parameters **size** (*int*) – The number of bytes to read, or None to read all of the remaining bytes

Returns An array of bytes

Return type bytearray

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read the requested size. Note that if there are no more bytes and size is None, an empty array will be returned

read_int()
Read the next four bytes as in int value

Returns An int

Return type int

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read an int

read_long()
Reads the next eight bytes as a long value

Returns A long

Return type long

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read a long

read_short()
Reads the next two bytes as a short value

Returns A short

Return type int

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read a short

spinn_storage_handlers.abstract_classes.abstract_byte_writer module

class spinn_storage_handlers.abstract_classes.abstract_byte_writer.**AbstractByteWriter**
Bases: object

An abstract writer of bytes. Note that due to endianness concerns, the methods of this writer should be used directly for the appropriate data type being written; e.g. an int should be written using write_int rather than calling write_byte 4 times with the parts of the int unless a specific endianness is being achieved.

get_n_bytes_written()
Determines how many bytes have been written in total

Returns The number of bytes written

Return type int

Raises **None** – No known exception is raised

write_byte(*byte_value*)
Writes the lowest order byte of the given value

Parameters **byte_value** (*int*) – The byte to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_bytes(*byte_iterable*)
Writes a set of bytes

Parameters **byte_iterable** (*iterable of bytes*) – The bytes to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_int(*int_value*)
Writes a four byte value

Parameters **int_value** (*int*) – The integer to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_long(*long_value*)
Writes an eight byte value

Parameters **long_value** (*long*) – The long to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_short(*short_value*)
Writes the two lowest order bytes of the given value

Parameters **short_value** (*int*) – The short to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

spinn_storage_handlers.abstract_classes.abstract_context_manager module

class spinn_storage_handlers.abstract_classes.abstract_context_manager.**AbstractContextManager**
Bases: object

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

close()

spinn_storage_handlers.abstract_classes.abstract_data_reader module

class spinn_storage_handlers.abstract_classes.abstract_data_reader.**AbstractDataReader**
Bases: object

Abstract reader used to read data from somewhere

read(*n_bytes*)
Read some bytes of data from the underlying storage. Will block until some bytes are available, but might not return the full n_bytes. The size of the returned array indicates how many bytes were read.

Parameters **n_bytes** (*int*) – The number of bytes to read

Returns An array of bytes

Return type bytearray

Raises **IOError** – If an error occurs reading from the underlying storage

readall()
Read the rest of the bytes from the underlying stream

Returns The bytes read

Return type bytearray

Raises **IOError** – If there is an error obtaining the bytes

readinto(*data*)
Read some bytes of data from the underlying storage into a pre-defined array. Will block until some bytes are available, but may not fill the array completely.

Parameters **data** (*bytearray*) – The place where the data is to be stored

Returns The number of bytes stored in data

Return type int

Raises **IOError** – If an error occurs reading from the underlying storage

tell()
Returns the position of the file cursor

Returns Position of the file cursor

Return type int

spinn_storage_handlers.abstract_classes.abstract_data_writer module

class spinn_storage_handlers.abstract_classes.abstract_data_writer.**AbstractDataWriter**
Bases: object

Abstract writer used to write data somewhere

tell()
Returns the position of the file cursor

Returns Position of the file cursor

Return type int

write(*data*)
Write some bytes of data to the underlying storage. Does not return until all the bytes have been written.

Parameters **data** (*bytearray*) – The data to write

Returns Nothing is returned

Return type None

Raises **IOError** – If an error occurs writing to the underlying storage

Module contents

class `spinn_storage_handlers.abstract_classes.`**`AbstractBufferedDataStorage`**

Bases: `object`

An object that can store and read back buffered data

`close`()

Closes the data storage

Return type None

Raises *`spinn_storage_handlers.exceptions.DataReadException`* – If the data storage cannot be closed

`eof`()

Check if the read pointer is a the end of the data storage

Returns True if the read pointer is at the end of the data storage, False otherwise

Return type bool

`read`(*data_size*)

Read data from the data storage from the position indicated by the read pointer index

Parameters **`data_size`** (*`int`*) – number of bytes to be read

Returns a bytearray containing the data read

Return type bytearray

`read_all`()

Read all the data contained in the data storage starting from position 0 to the end

Returns a bytearray containing the data read

Return type bytearray

`readinto`(*data*)

Read some bytes of data from the underlying storage into a pre-defined array. Will block until some bytes are available, but may not fill the array completely.

Parameters **`data`** (*`bytearray`*) – The place where the data is to be stored

Returns The number of bytes stored in data

Return type int

Raises **`IOError`** – If an error occurs reading from the underlying storage

`seek_read`(*offset*, *whence=0*)

Set the data storage's current read position to the offset

Parameters

- **`offset`** (*`int`*) – Position of the read pointer within the buffer
- **`whence`** – One of: * os.SEEK_SET which means absolute buffer positioning (default) * os.SEEK_CUR which means seek relative to the current read position * os.SEEK_END which means seek relative to the buffer's end

Return type None

`seek_write`(*offset*, *whence=0*)

Set the data storage's current write position to the offset

Parameters

- **offset** (*int*) – Position of the write pointer within the buffer
- **whence** – One of: * os.SEEK_SET which means absolute buffer positioning (default) * os.SEEK_CUR which means seek relative to the current write position * os.SEEK_END which means seek relative to the buffer's end

Return type None

tell_read()

The current position of the read pointer

Returns The current position of the read pointer

Return type int

tell_write()

The current position of the write pointer

Returns The current position of the write pointer

Return type int

write(*data*)

Store data in the data storage in the position indicated by the write pointer index

Parameters **data** (*bytearray*) – the data to be stored

Return type None

class spinn_storage_handlers.abstract_classes.**AbstractByteReader**

Bases: object

An abstract reader of bytes. Note that due to endianness concerns, the methods of this reader should be used directly for the appropriate data type being read; e.g. an int should be written using read_int rather than calling read_byte 4 times unless a specific endianness is being achieved.

is_at_end()

returns true if the reader is currently at the end of the byte reader

Returns returns true if the reader is currently at the end of the byte reader false otherwise

Return type bool

read_byte()

Reads the next byte

Returns A byte

Return type int

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are no more bytes to read

read_bytes(*size=None*)

Reads an array of bytes

Parameters **size** (*int*) – The number of bytes to read, or None to read all of the remaining bytes

Returns An array of bytes

Return type bytearray

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read the requested size. Note that if there are no more bytes and size is None, an empty array will be returned

read_int()

Read the next four bytes as in int value

Returns An int

Return type int

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read an int

read_long()

Reads the next eight bytes as a long value

Returns A long

Return type long

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read a long

read_short()

Reads the next two bytes as a short value

Returns A short

Return type int

Raises

- **IOError** – If there is an error reading from the stream
- **EOFError** – If there are too few bytes to read a short

class spinn_storage_handlers.abstract_classes.**AbstractByteWriter**

Bases: object

An abstract writer of bytes. Note that due to endianness concerns, the methods of this writer should be used directly for the appropriate data type being written; e.g. an int should be written using write_int rather than calling write_byte 4 times with the parts of the int unless a specific endianness is being achieved.

get_n_bytes_written()

Determines how many bytes have been written in total

Returns The number of bytes written

Return type int

Raises **None** – No known exception is raised

write_byte(*byte_value*)

Writes the lowest order byte of the given value

Parameters **byte_value** (*int*) – The byte to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_bytes(*byte_iterable*)
Writes a set of bytes

Parameters **byte_iterable** (*iterable of bytes*) – The bytes to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_int(*int_value*)
Writes a four byte value

Parameters **int_value** (*int*) – The integer to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_long(*long_value*)
Writes an eight byte value

Parameters **long_value** (*long*) – The long to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

write_short(*short_value*)
Writes the two lowest order bytes of the given value

Parameters **short_value** (*int*) – The short to write

Returns Nothing is returned

Return type None

Raises **IOError** – If there is an error writing to the stream

class spinn_storage_handlers.abstract_classes.**AbstractContextManager**
Bases: object

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

close()

class spinn_storage_handlers.abstract_classes.**AbstractDataReader**
Bases: object

Abstract reader used to read data from somewhere

read(*n_bytes*)
Read some bytes of data from the underlying storage. Will block until some bytes are available, but might not return the full n_bytes. The size of the returned array indicates how many bytes were read.

Parameters **n_bytes** (*int*) – The number of bytes to read

Returns An array of bytes

Return type bytearray

Raises **IOError** – If an error occurs reading from the underlying storage

readall()
Read the rest of the bytes from the underlying stream

Returns The bytes read

Return type bytearray

Raises **IOError** – If there is an error obtaining the bytes

readinto(*data*)
Read some bytes of data from the underlying storage into a pre-defined array. Will block until some bytes are available, but may not fill the array completely.

Parameters **data** (*bytearray*) – The place where the data is to be stored

Returns The number of bytes stored in data

Return type int

Raises **IOError** – If an error occurs reading from the underlying storage

tell()
Returns the position of the file cursor

Returns Position of the file cursor

Return type int

class spinn_storage_handlers.abstract_classes.**AbstractDataWriter**
Bases: object

Abstract writer used to write data somewhere

tell()
Returns the position of the file cursor

Returns Position of the file cursor

Return type int

write(*data*)
Write some bytes of data to the underlying storage. Does not return until all the bytes have been written.

Parameters **data** (*bytearray*) – The data to write

Returns Nothing is returned

Return type None

Raises **IOError** – If an error occurs writing to the underlying storage

Submodules

spinn_storage_handlers.buffered_bytearray_data_storage module

class spinn_storage_handlers.buffered_bytearray_data_storage.**BufferedBytearrayDataStorage**
Bases: *spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Data storage based on a bytearray buffer with two pointers, one for reading and one for writing.

close()

eof()

read(*data_size*)

read_all()

readinto(*data*)

seek_read(*offset*, *whence=0*)

seek_write(*offset*, *whence=0*)

tell_read()

tell_write()

write(*data*)

spinn_storage_handlers.buffered_file_data_storage module

class spinn_storage_handlers.buffered_file_data_storage.**BufferedFileDataStorage**(*filename*, *mode*)

Bases: *spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Data storage based on a temporary file with two pointers, one for reading and one for writing.

close()

eof()

filename

property method

read(*data_size*)

read_all()

readinto(*data*)

See *spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage.readinto()*

seek_read(*offset*, *whence=0*)

seek_write(*offset*, *whence=0*)

tell_read()

tell_write()

write(*data*)

spinn_storage_handlers.buffered_tempfile_data_storage module

class spinn_storage_handlers.buffered_tempfile_data_storage.**BufferedTempfileDataStorage**

Bases: *spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Data storage based on a temporary file with two pointers, one for reading and one for writing.

close()

eof()

classmethod initialise(*lru_max*)

read(*data_size*)

read_all()

readinto(*data*)

seek_read(*offset*, *whence=0*)

seek_write(*offset*, *whence=0*)

tell_read()

tell_write()

write(*data*)

spinn_storage_handlers.exceptions module

exception spinn_storage_handlers.exceptions.**BufferedBytearrayOperationNotImplemented**(*message*)
Bases: exceptions.Exception

An exception that denotes that the operation required is unavailable for a byteArray buffer

Parameters **message** (*str*) – A message to indicate what when wrong

exception spinn_storage_handlers.exceptions.**DataReadException**(*message*)
Bases: exceptions.Exception

An exception that indicates that there was an error reading from the underlying medium

Parameters **message** (*str*) – A message to indicate what when wrong

exception spinn_storage_handlers.exceptions.**DataWriteException**(*message*)
Bases: exceptions.Exception

An exception that indicates that there was an error writing to the underlying medium

Parameters **message** (*str*) – A message to indicate what when wrong

spinn_storage_handlers.file_data_reader module

class spinn_storage_handlers.file_data_reader.**FileDataReader**(*filename*)
Bases: *spinn_storage_handlers.abstract_classes.abstract_data_reader.AbstractDataReader*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

A reader that can read data from a file

Parameters **filename** (*str*) – The file to read

Raises ***spinn_storage_handlers.exceptions.DataReadException*** – If the file cannot found or opened for reading

close()
Closes the file

Return type None

Raises ***spinn_storage_handlers.exceptions.DataReadException*** – If the file cannot be closed

read(*n_bytes*)
See `data_specification.abstract_data_reader.AbstractDataReader.read()`

readall()
See `data_specification.abstract_data_reader.AbstractDataReader.readall()`

readinto(*data*)
See `data_specification.abstract_data_reader.AbstractDataReader.readinto()`

tell()
Returns the position of the file cursor

Returns Position of the file cursor

Return type int

spinn_storage_handlers.file_data_writer module

class spinn_storage_handlers.file_data_writer.**FileDataWriter**(*filename*)
Bases: *spinn_storage_handlers.abstract_classes.abstract_data_writer.AbstractDataWriter*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Parameters **filename** (*str*) – The file to write to

Raises ***spinn_storage_handlers.exceptions.DataWriteException*** – If the file cannot found or opened for writing

close()
Closes the file

Return type None

Raises ***spinn_storage_handlers.exceptions.DataWriteException*** – If the file cannot be closed

filename
property method

tell()
Returns the position of the file cursor

Returns Position of the file cursor

Return type int

write(*data*)
See `data_specification.abstract_data_writer.AbstractDataWriter.write()`

Module contents

class spinn_storage_handlers.**BufferedBytearrayDataStorage**
Bases: *spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Data storage based on a bytearray buffer with two pointers, one for reading and one for writing.

close()

eof()

read(*data_size*)

read_all()

readinto(*data*)

seek_read(*offset*, *whence=0*)

seek_write(*offset*, *whence=0*)

tell_read()

tell_write()

write(*data*)

class spinn_storage_handlers.**BufferedFileDataStorage**(*filename*, *mode*)

Bases: *spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Data storage based on a temporary file with two pointers, one for reading and one for writing.

close()

eof()

filename

property method

read(*data_size*)

read_all()

readinto(*data*)

See *spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage.readinto()*

seek_read(*offset*, *whence=0*)

seek_write(*offset*, *whence=0*)

tell_read()

tell_write()

write(*data*)

class spinn_storage_handlers.**BufferedTempfileDataStorage**

Bases: *spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Data storage based on a temporary file with two pointers, one for reading and one for writing.

close()

eof()

classmethod initialise(*lru_max*)

read(*data_size*)

read_all()

readinto(*data*)

seek_read(*offset*, *whence=0*)

seek_write(*offset*, *whence=0*)

tell_read()

tell_write()

write(*data*)

class spinn_storage_handlers.**FileDataReader**(*filename*)

Bases: *spinn_storage_handlers.abstract_classes.abstract_data_reader.AbstractDataReader*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

A reader that can read data from a file

Parameters **filename** (*str*) – The file to read

Raises ***spinn_storage_handlers.exceptions.DataReadException*** – If the file cannot found or opened for reading

close()

Closes the file

Return type None

Raises ***spinn_storage_handlers.exceptions.DataReadException*** – If the file cannot be closed

read(*n_bytes*)

See data_specification.abstract_data_reader.AbstractDataReader.read()

readall()

See data_specification.abstract_data_reader.AbstractDataReader.readall()

readinto(*data*)

See data_specification.abstract_data_reader.AbstractDataReader.readinto()

tell()

Returns the position of the file cursor

Returns Position of the file cursor

Return type int

class spinn_storage_handlers.**FileDataWriter**(*filename*)

Bases: *spinn_storage_handlers.abstract_classes.abstract_data_writer.AbstractDataWriter*, *spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager*

Parameters **filename** (*str*) – The file to write to

Raises ***spinn_storage_handlers.exceptions.DataWriteException*** – If the file cannot found or opened for writing

close()

Closes the file

Return type None

Raises *spinn_storage_handlers.exceptions.DataWriteException* – If the file cannot be closed

filename
property method

tell()
Returns the position of the file cursor

Returns Position of the file cursor

Return type int

write(*data*)
See `data_specification.abstract_data_writer.AbstractDataWriter.write()`

3.1.2 Indices and tables

- genindex
- modindex
- search

SpiNNStorageHandlers_github

SpiNNStorageHandlers_individual_docs

CHAPTER 4

PACMAN

This package provides utilities for partitioning, placing a routing on a SpiNNaker machine

4.1 PACMAN

Provides various functions which together can be used to take a graph and split it into pieces that can be loaded on to a machine, along with routes between the pieces.

4.1.1 Functional Requirements

- Creation of an Application Graph of Vertices indicating points of computation within the graph and Edges between the vertices indicating a directional communication between the vertices; and a similar Machine Graph.
  - Vertices in the Application Graph will have a number of *atoms* - an atom cannot be broken down in to anything smaller.
  - Vertices in the Application Graph must be able to indicate what machine resources are required by any given subset of the atoms.
  - Vertices in the Machine Graph must be able to fit on a single chip of the machine in terms of resource usage.
  - A Vertex can have a number of constraints which must be respected by any algorithm which uses the graph. Algorithms must check that they can support the given constraints and must fail if they cannot. Provided constraints include support for:
    * The maximum number of atoms which any Machine Graph Vertex can contain for a given Application Graph vertex
    * The chip and/or processor on to which a Machine Graph Vertex should be placed.
    * A set of Application Graph Vertices whose corresponding Machine Graph vertices should contain the same number of atoms.

    * A set of Application Graph Vertices whose corresponding Machine Graph vertices should be placed on the same chip if they contain the same atom.
  - It should be possible to create new constraints as the need arises.
  - Multiple edges can exist between the same two vertices.
  - It must be possible to build the Machine Graph directly without requiring that it is created by one of the other modules.
  - It is *not* required that there is a Machine Graph Edge between every pair of Machine Graph Vertex from the same Application Graph Vertex.
  - Where a Machine Graph is created from an Application Graph, it should be possible to find the corresponding Vertices and Edges from one graph to the other.
- Creation of multicast routing info consisting of key/mask combinations assigned to Edges of the Machine Graph.
  - It must be possible to build this information directly without requiring that it is created by one of the other modules.
  - There should be exactly one key/mask combination for each Edge in the Machine Graph, which will represent all the keys which will be sent in all packets from the Vertex at the start of the Edge down that Edge.
  - It is possible for a Vertex to send several different keys down several different Edges, but only one per Edge (but note that it is acceptable for different keys to be assigned to different Edges between the same two Vertices).
  - There should be no overlap between the key/mask combinations of Edges which come from different Vertices i.e. no two Edges which start at different Vertices should have the same key/mask combination.
- Partitioning of an Application graph with respect to a machine, such that the resources consumed by each Vertex does not exceed those provided by each chip on the machine.
  - It should be possible to select from a range of partitioning algorithms or provide one, although a default should be provided in the absence of such a choice .
  - Any partitioning constraints should be met; if there are any that cannot, or that are not understood by the algorithm in use an exception should be thrown. Non-partitioning constraints can be ignored, although these can be used if it makes sense for the given algorithm.
  - It must be possible to create at least one grouping of the generated Vertices so that each group fits within the resources provided by a single chip on the machine.
  - It should not be assumed that a given grouping of Vertices will be the final grouping on the machine, although it is acceptable to make hints through additional constraints about what is likely to work.
  - The machine itself must not be altered by the partitioning, so that it can be used in further processing.
  - The graph itself must not be altered by the partitioning, so that it can be used in further processing.
  - No two Machine Graph Vertices created from a single Application Graph Vertex can contain the same atom.
  - Any Edges in the Application Graph must be split with the Vertices to create a number of Machine Graph edges, such that where there was a vertex v connected to a vertex w by a single edge in the Application Graph, there should be an Edge in the Machine Graph between every Vertex of Application Graph Vertex v and every Vertex of Application Graph Vertex w; for example, if there are 2 Machine Graph Vertices for each of v and w, and one Edge between them in the Application Graph, then there will be 4 new Edges in the Machine Graph for this Edge.
- Placement of a Machine Graph on a given machine, such that the resources required by any combination of Vertices placed on any chip in the machine does not exceed the resources provided by that chip.

  - It should be possible to choose from a range of placement algorithms or provide one, although a default should be provided in the absence of such a choice.
  - Any placement constraints should be met; if there are any that cannot, or that are not understood by placement algorithm, an exception should be thrown. Non-placement constraints can be ignored, although these can be used if it makes sense for the given algorithm.
  - The machine itself should not be altered by placement so that it can be used in further processing.
  - The graph itself should not be altered by placement so that it can be used in further processing.
  - The returned placements should only contain a single placement for each vertex.
  - The placements should be such that the vertices with edges between them must be able to communicate with each other.
- Allocation of multicast routing keys and masks to a Machine Graph such that each vertex sends out packets with a different key/mask combination.
  - This can use the placement information if required. If an algorithm requires placement information but none is provided an exception is thrown.
- Routing of edges between vertices with a given allocation of routing keys and masks with respect to a given machine.
  - It should be possible to choose from a range of routing algorithms, or provide one, although a default should be provided in the absence of such a choice
  - For any vertex, following the routes from the placement of the vertex should result exactly in the set of placements of the destination vertices described by all the edges which start at that vertex. No additional destination should be reached, and no fewer than this set of destinations should be reached.
- It should be possible to call each of the modules independently. There should be no assumption that one of the other modules has produced the data input for any other module.
- There should be no assumption about how the inputs and outputs are stored.
- Any utility functions that provide access to internal structures within a data structure should operate in approximately O(1) time; for example, where an object of type *obj* holds a number of objects of type *subobj* with property *prop*, requesting a list of *subobj* objects contained within *obj* with property value *prop* = *value* should not iterate through a list of such objects, but should instead maintain a mapping that allows access to such objects in O(1) time. If this is not possible, *obj* should only provide access to a list of *subobj* objects, allowing the caller to filter these themselves. This will ensure that no misunderstanding can be made about the speed of operation of a function.

4.1.2 Contents

pacman

pacman package

Subpackages

pacman.executor package

Subpackages

pacman.executor.algorithm_classes package

Submodules

pacman.executor.algorithm_classes.abstract_algorithm module

class `pacman.executor.algorithm_classes.abstract_algorithm.`**`AbstractAlgorithm`**(*algorithm_id, required_inputs, optional_inputs, outputs*)

Bases: `object`

Represents the metadata for an algorithm

Parameters

- **`algorithm_id`** (*`str`*) – The unique id of the algorithm
- **`required_inputs`** (*`list of AbstractInput`*) – The inputs required by the algorithm
- **`optional_inputs`** (*`list of AbstractInput`*) – The optional inputs for the algorithm, which will be provided when available
- **`outputs`** (*`list of Output`*) – The output types of the algorithm

`algorithm_id`
The id for this algorithm

`call`(*inputs*)
Call the algorithm with the given inputs and return the outputs

Parameters **`inputs`** – A dict of input type -> value

Returns A dict of output type -> value

`optional_inputs`
The optional inputs of the algorithm

`outputs`
The outputs of the algorithm

required_inputs
The required inputs of the algorithm

write_provenance_header(*provenance_file*)
Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

pacman.executor.algorithm_classes.abstract_python_algorithm module

class pacman.executor.algorithm_classes.abstract_python_algorithm.**AbstractPythonAlgorithm**(*algo re- quir op- tion out- puts pyth*

Bases: *pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgorithm*

An algorithm written in Python

Parameters

- **algorithm_id** (*str*) – The unique id of the algorithm
- **required_inputs** (*list of AbstractInput*) – The inputs required by the algorithm
- **optional_inputs** (*list of AbstractInput*) – The optional inputs for the algorithm, which will be provided when available
- **outputs** (*list of Output*) – The output types of the algorithm
- **python_module** – The module containing the python code to execute

call(*inputs*)
Call the algorithm with the given inputs and return the outputs

Parameters **inputs** – A dict of input type -> value

Returns A dict of output type -> value

call_python(*inputs*)
Call the algorithm

Parameters **inputs** – A dict of parameter name -> value

Returns The result of calling the python algorithm

pacman.executor.algorithm_classes.external_algorithm module

class pacman.executor.algorithm_classes.external_algorithm.**ExternalAlgorithm**(*algorithm_id, required_inputs, optional_inputs, outputs, command_line_arguments*)

Bases: *pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgorithm*

the container for a algorithm which is external to the SpiNNaker software

call(*inputs*)

Call the algorithm with the given inputs and return the outputs

Parameters **inputs** – A dict of input type -> value

Returns A dict of output type -> value

write_provenance_header(*provenance_file*)

Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

pacman.executor.algorithm_classes.python_class_algorithm module

class pacman.executor.algorithm_classes.python_class_algorithm.**PythonClassAlgorithm**(*algorithm_id, required_input optional_inputs outputs, python_modu python_class python_meth*

Bases: *pacman.executor.algorithm_classes.abstract_python_algorithm.AbstractPythonAlgorithm*

An algorithm that is a class

Parameters

- **algorithm_id** (*str*) – The unique id of the algorithm
- **required_inputs** (*list of AbstractInput*) – The inputs required by the algorithm
- **optional_inputs** (*list of AbstractInput*) – The optional inputs for the algorithm, which will be provided when available
- **outputs** (*list of Output*) – The output types of the algorithm

- **python_module** – The module containing the python code to execute
- **python_class** – The class of the algorithm
- **python_method** – The method of the algorithm, or None if the class is callable

call_python(*inputs*)
Call the algorithm

Parameters **inputs** – A dict of parameter name -> value

Returns The result of calling the python algorithm

write_provenance_header(*provenance_file*)
Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

pacman.executor.algorithm_classes.python_function_algorithm module

class pacman.executor.algorithm_classes.python_function_algorithm.**PythonFunctionAlgorithm**(*algo re- quir op- tion out- puts pyth pyth*

Bases: *pacman.executor.algorithm_classes.abstract_python_algorithm.AbstractPythonAlgorithm*

An algorithm that is a function

Parameters

- **algorithm_id** (*str*) – The unique id of the algorithm
- **required_inputs** (*list of AbstractInput*) – The inputs required by the algorithm
- **optional_inputs** (*list of AbstractInput*) – The optional inputs for the algorithm, which will be provided when available
- **outputs** (*list of Output*) – The output types of the algorithm
- **python_module** – The module containing the python code to execute

Python_function The name of the function to call

call_python(*inputs*)
Call the algorithm

Parameters **inputs** – A dict of parameter name -> value

Returns The result of calling the python algorithm

write_provenance_header(*provenance_file*)
Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

Module contents

class pacman.executor.algorithm_classes.**AbstractAlgorithm**(*algorithm_id*, *required_inputs*, *optional_inputs*, *outputs*)

Bases: object

Represents the metadata for an algorithm

Parameters

- **algorithm_id** (*str*) – The unique id of the algorithm
- **required_inputs** (*list of AbstractInput*) – The inputs required by the algorithm
- **optional_inputs** (*list of AbstractInput*) – The optional inputs for the algorithm, which will be provided when available
- **outputs** (*list of Output*) – The output types of the algorithm

algorithm_id
The id for this algorithm

call(*inputs*)
Call the algorithm with the given inputs and return the outputs

Parameters **inputs** – A dict of input type -> value

Returns A dict of output type -> value

optional_inputs
The optional inputs of the algorithm

outputs
The outputs of the algorithm

required_inputs
The required inputs of the algorithm

write_provenance_header(*provenance_file*)
Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

class pacman.executor.algorithm_classes.**AbstractPythonAlgorithm**(*algorithm_id*, *required_inputs*, *optional_inputs*, *outputs*, *python_module*)

Bases: pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgorithm

An algorithm written in Python

Parameters

- **algorithm_id** (*str*) – The unique id of the algorithm

- **required_inputs** (*list of AbstractInput*) – The inputs required by the algorithm
- **optional_inputs** (*list of AbstractInput*) – The optional inputs for the algorithm, which will be provided when available
- **outputs** (*list of Output*) – The output types of the algorithm
- **python_module** – The module containing the python code to execute

call(*inputs*)

Call the algorithm with the given inputs and return the outputs

Parameters **inputs** – A dict of input type -> value

Returns A dict of output type -> value

call_python(*inputs*)

Call the algorithm

Parameters **inputs** – A dict of parameter name -> value

Returns The result of calling the python algorithm

class pacman.executor.algorithm_classes.**ExternalAlgorithm**(*algorithm_id*, *required_inputs*, *optional_inputs*, *outputs*, *command_line_arguments*)

Bases: *pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgorithm*

the container for a algorithm which is external to the SpiNNaker software

call(*inputs*)

Call the algorithm with the given inputs and return the outputs

Parameters **inputs** – A dict of input type -> value

Returns A dict of output type -> value

write_provenance_header(*provenance_file*)

Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

class pacman.executor.algorithm_classes.**PythonClassAlgorithm**(*algorithm_id*, *required_inputs*, *optional_inputs*, *outputs*, *python_module*, *python_class*, *python_method=None*)

Bases: *pacman.executor.algorithm_classes.abstract_python_algorithm.AbstractPythonAlgorithm*

An algorithm that is a class

Parameters

- **algorithm_id** (*str*) – The unique id of the algorithm
- **required_inputs** (*list of AbstractInput*) – The inputs required by the algorithm

- **optional_inputs** (*list of AbstractInput*) – The optional inputs for the algorithm, which will be provided when available
- **outputs** (*list of Output*) – The output types of the algorithm
- **python_module** – The module containing the python code to execute
- **python_class** – The class of the algorithm
- **python_method** – The method of the algorithm, or None if the class is callable

call_python(*inputs*)

Call the algorithm

Parameters **inputs** – A dict of parameter name -> value

Returns The result of calling the python algorithm

write_provenance_header(*provenance_file*)

Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

class pacman.executor.algorithm_classes.**PythonFunctionAlgorithm**(*algorithm_id*, *required_inputs*, *optional_inputs*, *outputs*, *python_module*, *python_function*)

Bases: *pacman.executor.algorithm_classes.abstract_python_algorithm.AbstractPythonAlgorithm*

An algorithm that is a function

Parameters

- **algorithm_id** (*str*) – The unique id of the algorithm
- **required_inputs** (*list of AbstractInput*) – The inputs required by the algorithm
- **optional_inputs** (*list of AbstractInput*) – The optional inputs for the algorithm, which will be provided when available
- **outputs** (*list of Output*) – The output types of the algorithm
- **python_module** – The module containing the python code to execute

Python_function The name of the function to call

call_python(*inputs*)

Call the algorithm

Parameters **inputs** – A dict of parameter name -> value

Returns The result of calling the python algorithm

write_provenance_header(*provenance_file*)

Writes the header info for this algorithm So things like name, module, class, function and command_line_arguments

But not anything about input and outputs as this is done elsewhere :param provenance_file: File to write to :type provenance_file: file

pacman.executor.algorithm_decorators package

Submodules

pacman.executor.algorithm_decorators.abstract_input module

class pacman.executor.algorithm_decorators.abstract_input.**AbstractInput**

Bases: object

An abstract input to an algorithm

get_inputs_by_name(*inputs*)

Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)

Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name

The name of the input

param_types

The types of the input

pacman.executor.algorithm_decorators.algorithm_decorator module

class pacman.executor.algorithm_decorators.algorithm_decorator.**AllOf**(**items*)

Bases: object

Indicates that all of the items specified are required

Parameters **items** (*str*, pacman.executor.algorithm_decorators.AllOf, pacman.executor.algorithm_decorators.OneOf) – The items required

items

The items specified

real_class

The AbstractInput class to use for this input

class pacman.executor.algorithm_decorators.algorithm_decorator.**OneOf**(**items*)

Bases: object

Indicates that one of the items specified is required

Parameters **items** (*str*, pacman.executor.algorithm_decorators.AllOf, pacman.executor.algorithm_decorators.OneOf) – The items required

items

The items specified

real_class
The AbstractInput class to use for this input

pacman.executor.algorithm_decorators.algorithm_decorator.**algorithm**(*input_definitions*, *outputs*, *algorithm_id=None*, *required_inputs=None*, *optional_inputs=None*, *method=None*)

Define an object to be a PACMAN algorithm that can be executed by the PacmanAlgorithmExecutor.

Can be used to decorate either a class or a function (not a method). If this decorates a class, the class must be callable (i.e. have a __call__ method), or else a method must be specified to call to run the algorithm.

The inputs and outputs referenced below refer to the parameters of the method or function.

Parameters

- **input_definitions** (*dict of str -> (str or list of str)*) – dict of algorithm parameter name to list of types, one for each required algorithm parameter, and one for each optional parameter that is used in this algorithm call
- **outputs** (*list of str*) – A list of types output from the algorithm that must match the order in which they are returned.
- **algorithm_id** (*str*) – Optional unique id of the algorithm; if not specified, the name of the class or function is used.
- **required_inputs** (*list of (str or pacman.executor.algorithm_decorators.OneOf or pacman.executor.algorithm_decorators.AllOf)*) – Optional list of required algorithm parameter names; if not specified those parameters which have no default values are used.
- **optional_inputs** (*list of (str or pacman.executor.algorithm_decorators.OneOf or pacman.executor.algorithm_decorators.AllOf)*) – Optional list of optional algorithm parameter names; if not specified those parameters which have default values are used.
- **method** – The optional name of the method to call if decorating a class; if not specified, __call__ is used (i.e. it is assumed to be callable). Must not be used if decorating a function

pacman.executor.algorithm_decorators.algorithm_decorator.**algorithms**(*algorithms*)

Specify multiple algorithms for a single class or function

Parameters **algorithms** – A list of algorithm definitions

pacman.executor.algorithm_decorators.algorithm_decorator.**get_algorithms**()

Get the dict of known algorithm id -> algorithm data

pacman.executor.algorithm_decorators.algorithm_decorator.**reset_algorithms**()

Reset the known algorithms

pacman.executor.algorithm_decorators.algorithm_decorator.**scan_packages**(*packages*, *recursive=True*)

Scan packages for algorithms

Parameters

- **packages** – The names of the packages to scan (using dotted notation), or the actual package modules
- **recursive** – True if sub-packages should be examined

Returns A dict of algorithm name -> algorithm data

pacman.executor.algorithm_decorators.all_of_input module

class pacman.executor.algorithm_decorators.all_of_input.**AllOfInput**(*inputs*)

Bases: *pacman.executor.algorithm_decorators.abstract_input.AbstractInput*

A composite input for which all input parameters must be matched

Parameters **inputs** – The inputs that make up this input

get_inputs_by_name(*inputs*)

Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)

Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name

The name of the input

param_types

The types of the input

pacman.executor.algorithm_decorators.one_of_input module

class pacman.executor.algorithm_decorators.one_of_input.**OneOfInput**(*inputs*)

Bases: *pacman.executor.algorithm_decorators.abstract_input.AbstractInput*

An input for which one of the input parameters must be matched

Parameters **inputs** – The inputs that make up this input

get_inputs_by_name(*inputs*)

Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)

Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name
The name of the input

param_types
The types of the input

pacman.executor.algorithm_decorators.output module

class pacman.executor.algorithm_decorators.output.**Output**(*output_type*, *file_name_type=None*)

Bases: object

Represents an output from an algorithm

Parameters

- **output_type** – The type of the output
- **file_name_type** – If the output is file based, the type of the input holding the file name

file_name_type

output_type

pacman.executor.algorithm_decorators.single_input module

class pacman.executor.algorithm_decorators.single_input.**SingleInput**(*name*, *param_types*)

Bases: *pacman.executor.algorithm_decorators.abstract_input.AbstractInput*

An input that is just one item

Parameters

- **name** (*str*) – The name of the input parameter
- **param_types** (*list of str*) – The ordered possible types of the input parameter

get_inputs_by_name(*inputs*)
Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)
Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name
The name of the input

param_types
The types of the input

Module contents

class pacman.executor.algorithm_decorators.**AbstractInput**

Bases: object

An abstract input to an algorithm

get_inputs_by_name(*inputs*)

Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)

Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name

The name of the input

param_types

The types of the input

class pacman.executor.algorithm_decorators.**AllOfInput**(*inputs*)

Bases: *pacman.executor.algorithm_decorators.abstract_input.AbstractInput*

A composite input for which all input parameters must be matched

Parameters **inputs** – The inputs that make up this input

get_inputs_by_name(*inputs*)

Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)

Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name

The name of the input

param_types

The types of the input

class pacman.executor.algorithm_decorators.**OneOfInput**(*inputs*)

Bases: *pacman.executor.algorithm_decorators.abstract_input.AbstractInput*

An input for which one of the input parameters must be matched

Parameters **inputs** – The inputs that make up this input

get_inputs_by_name(*inputs*)

Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)
Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name
The name of the input

param_types
The types of the input

class pacman.executor.algorithm_decorators.**Output**(*output_type*, *file_name_type=None*)
Bases: object

Represents an output from an algorithm

Parameters

- **output_type** – The type of the output
- **file_name_type** – If the output is file based, the type of the input holding the file name

file_name_type

output_type

class pacman.executor.algorithm_decorators.**SingleInput**(*name*, *param_types*)
Bases: *pacman.executor.algorithm_decorators.abstract_input.AbstractInput*

An input that is just one item

Parameters

- **name** (*str*) – The name of the input parameter
- **param_types** (*list of str*) – The ordered possible types of the input parameter

get_inputs_by_name(*inputs*)
Get the inputs that match this input by parameter name

Parameters **inputs** – A dict of type to value

Returns A dict of parameter name to value

Return type dict

input_matches(*inputs*)
Determine if this input is in the set of inputs

Parameters **inputs** – A set of input types

Returns True if this input type is in the list

name
The name of the input

param_types
The types of the input

class pacman.executor.algorithm_decorators.**AllOf**(**items*)

Bases: object

Indicates that all of the items specified are required

Parameters **items** (*str*, pacman.executor.algorithm_decorators.AllOf, pacman.executor.algorithm_decorators.OneOf) – The items required

items

The items specified

real_class

The AbstractInput class to use for this input

class pacman.executor.algorithm_decorators.**OneOf**(**items*)

Bases: object

Indicates that one of the items specified is required

Parameters **items** (*str*, pacman.executor.algorithm_decorators.AllOf, pacman.executor.algorithm_decorators.OneOf) – The items required

items

The items specified

real_class

The AbstractInput class to use for this input

pacman.executor.algorithm_decorators.**algorithm**(*input_definitions*, *outputs*, *algorithm_id=None*, *required_inputs=None*, *optional_inputs=None*, *method=None*)

Define an object to be a PACMAN algorithm that can be executed by the PacmanAlgorithmExecutor.

Can be used to decorate either a class or a function (not a method). If this decorates a class, the class must be callable (i.e. have a __call__ method), or else a method must be specified to call to run the algorithm.

The inputs and outputs referenced below refer to the parameters of the method or function.

Parameters

- **input_definitions** (*dict of str -> (str or list of str)*) – dict of algorithm parameter name to list of types, one for each required algorithm parameter, and one for each optional parameter that is used in this algorithm call
- **outputs** (*list of str*) – A list of types output from the algorithm that must match the order in which they are returned.
- **algorithm_id** (*str*) – Optional unique id of the algorithm; if not specified, the name of the class or function is used.
- **required_inputs** (*list of (str or* pacman.executor.algorithm_decorators.OneOf *or* pacman.executor.algorithm_decorators.AllOf*)*) – Optional list of required algorithm parameter names; if not specified those parameters which have no default values are used.
- **optional_inputs** (*list of (str or* pacman.executor.algorithm_decorators.OneOf *or* pacman.executor.algorithm_decorators.AllOf*)*) – Optional list of optional algorithm parameter names; if not specified those parameters which have default values are used.
- **method** – The optional name of the method to call if decorating a class; if not specified, __call__ is used (i.e. it is assumed to be callable). Must not be used if decorating a function

pacman.executor.algorithm_decorators.**algorithms**(*algorithms*)
Specify multiple algorithms for a single class or function

Parameters **algorithms** – A list of algorithm definitions

pacman.executor.algorithm_decorators.**get_algorithms**()
Get the dict of known algorithm id -> algorithm data

pacman.executor.algorithm_decorators.**reset_algorithms**()
Reset the known algorithms

pacman.executor.algorithm_decorators.**scan_packages**(*packages*, *recursive=True*)
Scan packages for algorithms

Parameters

- **packages** – The names of the packages to scan (using dotted notation), or the actual package modules
- **recursive** – True if sub-packages should be examined

Returns A dict of algorithm name -> algorithm data

Submodules

pacman.executor.algorithm_metadata_xml_reader module

class pacman.executor.algorithm_metadata_xml_reader.**AlgorithmMetadataXmlReader**(*xml_paths*)
Bases: object

Converts an XML file into algorithm data

Parameters **xml_paths** – paths to extra metadata files

Return type None

decode_algorithm_data_objects()

Returns the algorithm data objects which represent all the algorithm's inputs and outputs

pacman.executor.injection_decorator module

exception pacman.executor.injection_decorator.**InjectionException**
Bases: exceptions.Exception

Raised when there is an error with injection

pacman.executor.injection_decorator.**clear_injectables**()
Clear the current set of injectables

pacman.executor.injection_decorator.**do_injection**(*objects_to_inject*, *objects_to_inject_into=None*)
Perform the actual injection of objects

Parameters

- **objects_to_inject** (*dict(str)->object*) – The objects to be injected as a dict of type name -> object of type
- **objects_to_inject_into** (*list*) – The objects whose classes support_injection, or None to use all instances that have been created

pacman.executor.injection_decorator.**inject**(*type_to_inject*)

Marks a method as something to be called to inject an object of the given type. The type is just a name for the type, and should match up at some point with some generated data

Parameters **type_to_inject** – The type to be injected using this method

pacman.executor.injection_decorator.**inject_items**(*types*)

Indicates values that need to be injected into the method

Parameters **types** – A dict of method argument name to type name to be injected

class pacman.executor.injection_decorator.**injection_context**(*injection_dictionary*)

Bases: object

Provides a context for injection to use with "with"

Parameters **injection_dictionary** – The dictionary of items to inject whilst in the context

pacman.executor.injection_decorator.**provide_injectables**(*injectables*)

Set the objects from which values should be injected into methods

Parameters **injectables** – A dict of type to value

pacman.executor.injection_decorator.**requires_injection**(*types_required*)

Indicates that injection of the given types is required before this method is called; an Exception is raised if the types have not been injected

Parameters **types_required** (*list of str*) – A list of types that must have been injected

pacman.executor.injection_decorator.**supports_injection**(*injectable_class*)

Indicate that the class has methods on which objects can be injected

pacman.executor.pacman_algorithm_executor module

class pacman.executor.pacman_algorithm_executor.**PACMANAlgorithmExecutor**(*algorithms*, *optional_algorithms*, *inputs*, *required_outputs*, *xml_paths=None*, *packages=None*, *do_timings=True*, *print_timings=False*, *do_immediate_injection=True*, *do_post_run_injection=False*, *inject_inputs=True*, *do_direct_injection=True*, *use_unscanned_annotated_alg*, *provenance_path=None*, *provenance_name=None*)

Bases: object

An executor of PACMAN algorithms where the order is deduced from the input and outputs of the algorithm using an XML description of the algorithm

Parameters

- **algorithms** – A list of algorithms that must all be run
- **optional_algorithms** – A list of algorithms that must be run if their inputs are available
- **inputs** – A dict of input type to value
- **required_outputs** – A list of output types that must be generated
- **xml_paths** – An optional list of paths to XML files containing algorithm descriptions; if not specified, only detected algorithms will be used (or else those found in packages)
- **packages** – An optional list of packages to scan for decorated algorithms; if not specified, only detected algorithms will be used (or else those specified in packages
- **do_timings** – True if timing information should be printed after each algorithm, False otherwise
- **do_immediate_injection** – Perform injection with objects as they are created; can result in multiple calls to the same inject-annotated methods
- **do_post_run_injection** – Perform injection at the end of the run. This will only set the last object of any type created.
- **inject_inputs** – True if inputs should be injected; only active if one of do_immediate_injection or do_post_run_injection is True. These variables define when the injection of inputs is done; if immediate injection is True, injection of inputs is done at the start of the run, otherwise it is done at the end.
- **do_direct_injection** – True if direct injection into methods should be supported. This will allow any of the inputs or generated outputs to be injected into a method
- **use_unscanned_annotated_algorithms** – True if algorithms that have been detected outside of the packages argument specified above should be used
- **provenance_path** – Path to file to append full provenance data to If None no provenance is written

algorithm_timings

execute_mapping()

Executes the algorithms

Return type None

get_item(*item_type*)

Get an item from the outputs of the execution

Parameters **item_type** – the item from the internal type mapping to be returned

Returns the returned item

get_items()

Get all the outputs from a execution

Returns dictionary of types as keys and values.

Module contents

class pacman.executor.**AlgorithmMetadataXmlReader**(*xml_paths*)

Bases: object

Converts an XML file into algorithm data

Parameters **xml_paths** – paths to extra metadata files

Return type None

decode_algorithm_data_objects()

Returns the algorithm data objects which represent all the algorithm's inputs and outputs

class pacman.executor.**PACMANAlgorithmExecutor**(*algorithms*, *optional_algorithms*, *inputs*, *required_outputs*, *xml_paths=None*, *packages=None*, *do_timings=True*, *print_timings=False*, *do_immediate_injection=True*, *do_post_run_injection=False*, *inject_inputs=True*, *do_direct_injection=True*, *use_unscanned_annotated_algorithms=True*, *provenance_path=None*, *provenance_name=None*)

Bases: object

An executor of PACMAN algorithms where the order is deduced from the input and outputs of the algorithm using an XML description of the algorithm

Parameters

- **algorithms** – A list of algorithms that must all be run
- **optional_algorithms** – A list of algorithms that must be run if their inputs are available
- **inputs** – A dict of input type to value
- **required_outputs** – A list of output types that must be generated
- **xml_paths** – An optional list of paths to XML files containing algorithm descriptions; if not specified, only detected algorithms will be used (or else those found in packages)
- **packages** – An optional list of packages to scan for decorated algorithms; if not specified, only detected algorithms will be used (or else those specified in packages
- **do_timings** – True if timing information should be printed after each algorithm, False otherwise
- **do_immediate_injection** – Perform injection with objects as they are created; can result in multiple calls to the same inject-annotated methods
- **do_post_run_injection** – Perform injection at the end of the run. This will only set the last object of any type created.
- **inject_inputs** – True if inputs should be injected; only active if one of do_immediate_injection or do_post_run_injection is True. These variables define when the injection of inputs is done; if immediate injection is True, injection of inputs is done at the start of the run, otherwise it is done at the end.
- **do_direct_injection** – True if direct injection into methods should be supported. This will allow any of the inputs or generated outputs to be injected into a method

- **use_unscanned_annotated_algorithms** – True if algorithms that have been detected outside of the packages argument specified above should be used
- **provenance_path** – Path to file to append full provenance data to If None no provenance is written

algorithm_timings

execute_mapping()
Executes the algorithms

Return type None

get_item(*item_type*)
Get an item from the outputs of the execution

Parameters **item_type** – the item from the internal type mapping to be returned

Returns the returned item

get_items()
Get all the outputs from a execution

Returns dictionary of types as keys and values.

pacman.model package

Subpackages

pacman.model.abstract_classes package

Submodules

pacman.model.abstract_classes.abstract_has_global_max_atoms module

class pacman.model.abstract_classes.abstract_has_global_max_atoms.**AbstractHasGlobalMaxAtoms**
Bases: object

Indicates an application vertex which has a global max atoms per core

static get_max_atoms_per_core()
The global maximum atoms per core

Module contents

class pacman.model.abstract_classes.**AbstractHasGlobalMaxAtoms**
Bases: object

Indicates an application vertex which has a global max atoms per core

static get_max_atoms_per_core()
The global maximum atoms per core

pacman.model.constraints package

Subpackages

pacman.model.constraints.key_allocator_constraints package

Submodules

pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint module

class pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.**Ab**

Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on key allocation

pacman.model.constraints.key_allocator_constraints.contiguous_key_range_constraint module

class pacman.model.constraints.key_allocator_constraints.contiguous_key_range_constraint.**Cont**

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Key allocator constraint that keeps the keys allocated to a contiguous range. Without this constraint, keys can be allocated across the key space.

pacman.model.constraints.key_allocator_constraints.fixed_key_and_mask_constraint module

class pacman.model.constraints.key_allocator_constraints.fixed_key_and_mask_constraint.**FixedK**

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Key allocator constraint that fixes the key and mask of an edge

Parameters

- **keys_and_masks** (iterable of *pacman.model.routing_info.BaseKeyAndMask*) – The key and mask combinations to fix
- **key_list_function** ((iterable of *pacman.model.routing_info.BaseKeyAndMask*, pacman.model.graph.machine.MachineEdge, int) -> iterable of int) – Optional function which will be called to translate the keys_and_masks list into individual keys. If missing, the keys will be generated by iterating through the keys_and_masks list directly. The function parameters are: * An iterable of keys and masks * A machine edge * Number of keys to generate (may be None)

key_list_function

A function to call to generate the keys

Returns A python function, or None if the default function can be used

keys_and_masks

The keys and masks to be fixed

Returns An iterable of key and mask combinations

Return type iterable of *pacman.model.routing_info.BaseKeyAndMask*

pacman.model.constraints.key_allocator_constraints.fixed_key_field_constraint module

class pacman.model.constraints.key_allocator_constraints.fixed_key_field_constraint.**FixedKeyF**

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Constraint that indicates fields in the mask of a key

Parameters **fields** (iterable of pacman.utilities.field.Field) – any fields that define regions in the mask with further limitations

Raises ***PacmanInvalidParameterException*** – if any of the fields are outside of the mask i.e. mask & field.mask != field.mask or if any of the field masks overlap i.e. field.mask & other_field.mask != 0

fields

Any fields in the mask - i.e. ranges of the mask that have further limitations

Returns Iterable of fields, ordered by mask with the highest bit range first

Return type iterable of pacman.utilities.field.Field

pacman.model.constraints.key_allocator_constraints.fixed_mask_constraint module

class pacman.model.constraints.key_allocator_constraints.fixed_mask_constraint.**FixedMaskConst**

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

A key allocator that fixes the mask to be assigned to an edge

Parameters **mask** (*int*) – the mask to be used during key allocation

mask

The mask to be used

Returns The mask to be used

Return type int

pacman.model.constraints.key_allocator_constraints.flexi_key_field_constraint module

class pacman.model.constraints.key_allocator_constraints.flexi_key_field_constraint.**FlexiKeyF**

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Constraint that indicates fields in the mask without a specific size or position

fields

Module contents

class pacman.model.constraints.key_allocator_constraints.**AbstractKeyAllocatorConstraint**

Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on key allocation

class pacman.model.constraints.key_allocator_constraints.**ContiguousKeyRangeContraint**

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Key allocator constraint that keeps the keys allocated to a contiguous range. Without this constraint, keys can be allocated across the key space.

class pacman.model.constraints.key_allocator_constraints.**FixedKeyFieldConstraint**(*fields=None*)

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Constraint that indicates fields in the mask of a key

Parameters **fields** (iterable of pacman.utilities.field.Field) – any fields that define regions in the mask with further limitations

Raises ***PacmanInvalidParameterException*** – if any of the fields are outside of the mask i.e. mask & field.mask != field.mask or if any of the field masks overlap i.e. field.mask & other_field.mask != 0

fields

Any fields in the mask - i.e. ranges of the mask that have further limitations

Returns Iterable of fields, ordered by mask with the highest bit range first

Return type iterable of pacman.utilities.field.Field

class pacman.model.constraints.key_allocator_constraints.**FixedKeyAndMaskConstraint**(*keys_and_mas key_list_functi*

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Key allocator constraint that fixes the key and mask of an edge

Parameters

- **keys_and_masks** (iterable of *pacman.model.routing_info.BaseKeyAndMask*) – The key and mask combinations to fix
- **key_list_function** ((iterable of *pacman.model.routing_info.BaseKeyAndMask*, pacman.model.graph.machine.MachineEdge, int) -> iterable of int) – Optional function which will be called to translate the keys_and_masks list into individual keys. If missing, the keys will be generated by iterating through the keys_and_masks list directly. The function parameters are: * An iterable of keys and masks * A machine edge * Number of keys to generate (may be None)

key_list_function

A function to call to generate the keys

Returns A python function, or None if the default function can be used

keys_and_masks

The keys and masks to be fixed

Returns An iterable of key and mask combinations

Return type iterable of *pacman.model.routing_info.BaseKeyAndMask*

class pacman.model.constraints.key_allocator_constraints.**FixedMaskConstraint**(*mask*)

Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

A key allocator that fixes the mask to be assigned to an edge

Parameters **mask** (*int*) – the mask to be used during key allocation

mask
The mask to be used

Returns The mask to be used

Return type int

class pacman.model.constraints.key_allocator_constraints.**FlexiKeyFieldConstraint**(*fields*)
Bases: *pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint.AbstractKeyAllocatorConstraint*

Constraint that indicates fields in the mask without a specific size or position

fields

pacman.model.constraints.partitioner_constraints package

Submodules

pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint module

class pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint.**Abstra**
Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on the partitioning of a graph

pacman.model.constraints.partitioner_constraints.max_vertex_atoms_constraint module

class pacman.model.constraints.partitioner_constraints.max_vertex_atoms_constraint.**MaxVertexA**
Bases: *pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint.AbstractPartitionerConstraint*

A constraint which limits the number of atoms on each division of a vertex

Parameters **size** (*int*) – The maximum number of atoms to split the vertex into

size
The maximum number of atoms to split the vertex into

Return type int

pacman.model.constraints.partitioner_constraints.same_atoms_as_vertex_constraint module

class pacman.model.constraints.partitioner_constraints.same_atoms_as_vertex_constraint.**SameAt**
Bases: *pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint.AbstractPartitionerConstraint*

A constraint which indicates that a vertex must be split in the same way as another vertex

Parameters **vertex** (pacman.model.graph.application.application_vertex.ApplicationVertex) – The vertex to which the constraint refers

Raises **None** – does not raise any known exceptions

vertex
The vertex to partition with

Returns the vertex

Return type pacman.model.graph.application.application_vertex.ApplicationVertex

Raises **None** – does not raise any known exceptions

Module contents

class pacman.model.constraints.partitioner_constraints.**AbstractPartitionerConstraint**

Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on the partitioning of a graph

class pacman.model.constraints.partitioner_constraints.**MaxVertexAtomsConstraint**(*size*)

Bases: *pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint.AbstractPartitionerConstraint*

A constraint which limits the number of atoms on each division of a vertex

Parameters **size** (*int*) – The maximum number of atoms to split the vertex into

size

The maximum number of atoms to split the vertex into

Return type int

class pacman.model.constraints.partitioner_constraints.**SameAtomsAsVertexConstraint**(*vertex*)

Bases: *pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint.AbstractPartitionerConstraint*

A constraint which indicates that a vertex must be split in the same way as another vertex

Parameters **vertex** (pacman.model.graph.application.application_vertex.ApplicationVertex) – The vertex to which the constraint refers

Raises **None** – does not raise any known exceptions

vertex

The vertex to partition with

Returns the vertex

Return type pacman.model.graph.application.application_vertex.ApplicationVertex

Raises **None** – does not raise any known exceptions

pacman.model.constraints.placer_constraints package

Submodules

pacman.model.constraints.placer_constraints.abstract_placer_constraint module

class pacman.model.constraints.placer_constraints.abstract_placer_constraint.**AbstractPlacerCo**

Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on placement

pacman.model.constraints.placer_constraints.board_constraint module

class pacman.model.constraints.placer_constraints.board_constraint.**BoardConstraint**(*board_address*

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

A constraint on the board on which a placement is made

Parameters **board_address** – The IP address of the Ethernet of the board to be used

board_address

The board of the constraint

pacman.model.constraints.placer_constraints.chip_and_core_constraint module

class pacman.model.constraints.placer_constraints.chip_and_core_constraint.**ChipAndCoreConstra**

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

A constraint to place a vertex on a specific chip and, optionally, a specific core on that chip

Parameters

- **x** (*int*) – the x-coordinate of the chip
- **y** (*int*) – the y-coordinate of the chip
- **p** (*int*) – the processor (if any) of the chip

Raises **None** – does not raise any known exceptions

location

The location as a dictionary with three keys: "x", "y" and "p"

Returns a dictionary containing the location

Return type dict of {"x": int, "y": int, "p": int}

Raises **None** – does not raise any known exceptions

p

The processor on the chip

Returns the processor id or None

Return type int

Raises **None** – does not raise any known exceptions

x

The x-coordinate of the chip

Returns the x-coordinate

Return type int

Raises **None** – does not raise any known exceptions

y

The y-coordinate of the chip

Returns the y-coordinate

Return type int

Raises **None** – does not raise any known exceptions

pacman.model.constraints.placer_constraints.radial_placement_from_chip_constraint module

class pacman.model.constraints.placer_constraints.radial_placement_from_chip_constraint.**Radia**

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

A constraint that attempts to place a vertex as close to a chip as possible (including on it)

Parameters

- **x** (*int*) – the x-coordinate of the chip
- **y** (*int*) – the y-coordinate of the chip

Raises **None** – does not raise any known exceptions

x

y

pacman.model.constraints.placer_constraints.same_chip_as_constraint module

class pacman.model.constraints.placer_constraints.same_chip_as_constraint.**SameChipAsConstrain**

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

Indicates that a vertex should be placed on the same chip as another vertex

Parameters **vertex** – The vertex to place on the same chip

vertex

The vertex to place on the same chip

Module contents

class pacman.model.constraints.placer_constraints.**AbstractPlacerConstraint**

Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on placement

class pacman.model.constraints.placer_constraints.**BoardConstraint**(*board_address*)

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

A constraint on the board on which a placement is made

Parameters **board_address** – The IP address of the Ethernet of the board to be used

board_address

The board of the constraint

class pacman.model.constraints.placer_constraints.**ChipAndCoreConstraint**(*x*, *y*, *p=None*)

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

A constraint to place a vertex on a specific chip and, optionally, a specific core on that chip

Parameters

- **x** (*int*) – the x-coordinate of the chip
- **y** (*int*) – the y-coordinate of the chip
- **p** (*int*) – the processor (if any) of the chip

Raises **None** – does not raise any known exceptions

location

The location as a dictionary with three keys: "x", "y" and "p"

Returns a dictionary containing the location

Return type dict of {"x": int, "y": int, "p": int}

Raises **None** – does not raise any known exceptions

p

The processor on the chip

Returns the processor id or None

Return type int

Raises **None** – does not raise any known exceptions

x

The x-coordinate of the chip

Returns the x-coordinate

Return type int

Raises **None** – does not raise any known exceptions

y

The y-coordinate of the chip

Returns the y-coordinate

Return type int

Raises **None** – does not raise any known exceptions

class pacman.model.constraints.placer_constraints.**RadialPlacementFromChipConstraint**(*x*, *y*)

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

A constraint that attempts to place a vertex as close to a chip as possible (including on it)

Parameters

- **x** (*int*) – the x-coordinate of the chip
- **y** (*int*) – the y-coordinate of the chip

Raises **None** – does not raise any known exceptions

x

y

class pacman.model.constraints.placer_constraints.**SameChipAsConstraint**(*vertex*)

Bases: *pacman.model.constraints.placer_constraints.abstract_placer_constraint.AbstractPlacerConstraint*

Indicates that a vertex should be placed on the same chip as another vertex

Parameters **vertex** – The vertex to place on the same chip

vertex

The vertex to place on the same chip

pacman.model.constraints.router_constraints package

Submodules

pacman.model.constraints.router_constraints.abstract_router_constraint module

class pacman.model.constraints.router_constraints.abstract_router_constraint.**AbstractRouterCo**

Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on routing

Module contents

class pacman.model.constraints.router_constraints.**AbstractRouterConstraint**

Bases: *pacman.model.constraints.abstract_constraint.AbstractConstraint*

A constraint on routing

Submodules

pacman.model.constraints.abstract_constraint module

class pacman.model.constraints.abstract_constraint.**AbstractConstraint**

Bases: object

A constraint of some sort which an algorithm might or might not support

Module contents

class pacman.model.constraints.**AbstractConstraint**

Bases: object

A constraint of some sort which an algorithm might or might not support

pacman.model.decorators package

Submodules

pacman.model.decorators.overrides module

class `pacman.model.decorators.overrides.`**`overrides`**(*super_class_method*, *extend_doc=True*, *additional_arguments=None*)

Bases: `object`

A decorator for indicating that a method overrides another method in a super class. 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`** – 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`** – Additional arguments taken by the subclass method over the superclass method e.g. that are to be injected

Module contents

class `pacman.model.decorators.`**`overrides`**(*super_class_method*, *extend_doc=True*, *additional_arguments=None*)

Bases: `object`

A decorator for indicating that a method overrides another method in a super class. 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`** – 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`** – Additional arguments taken by the subclass method over the superclass method e.g. that are to be injected

pacman.model.graphs package

Subpackages

pacman.model.graphs.application package

Submodules

pacman.model.graphs.application.application_edge module

class `pacman.model.graphs.application.application_edge.`**`ApplicationEdge`**(*pre_vertex, post_vertex, traffic_type=<EdgeTrafficType.MUL 1>, label=None, machine_edge_type=<class 'pacman.model.graphs.machine.mac*

Bases: `pacman.model.graphs.abstract_edge.AbstractEdge`

A simple implementation of an application edge

Parameters

- **pre_vertex** (`pacman.model.graph.application.application_vertex.ApplicationVertex`) – the application vertex at the start of the edge
- **post_vertex** (`pacman.model.graph.application.application_vertex.ApplicationVertex`) – the application vertex at the end of the edge
- **traffic_type** (`pacman.model.graph.edge_traffic_type.EdgeTrafficType`) – The type of the traffic on the edge
- **label** (*str*) – The name of the edge

create_machine_edge(*pre_vertex, post_vertex, label*)

Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (`pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex`) – The machine vertex at the start of the edge
- **post_vertex** (`pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex`) – The machine vertex at the end of the edge
- **label** (*str*) – label of the edge

Returns The created machine edge

Return type `pacman.model.graph.machine.machine_edge.MachineEdge`

label
The label of the edge

Returns The label

Return type str

Raises **None** – Raises no known exceptions

post_vertex
The vertex at the end of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

pre_vertex
The vertex at the start of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type
The traffic type of the edge

Return type py:class:*pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

pacman.model.graphs.application.application_fpga_vertex module

class pacman.model.graphs.application.application_fpga_vertex.**ApplicationFPGAVertex**(*n_atoms*, *fpga_id*, *fpga_link_id*, *board_addre*, *label=None*, *constraints=Non*, *max_atoms_*

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *pacman.model.graphs.abstract_fpga_vertex.AbstractFPGAVertex*

A virtual vertex on an FPGA link

board_address
The IP address of the board to which the device is connected, or None for the boot board

Return type str

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

fpga_id
The id of the FPGA to which the vertex is connected

Return type int

fpga_link_id
The link of the FPGA to which the vertex is connected

Return type int

get_resources_used_by_atoms (*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

set_virtual_chip_coordinates (*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

pacman.model.graphs.application.application_graph module

class pacman.model.graphs.application.application_graph.**ApplicationGraph** (*label*)
Bases: *pacman.model.graphs.impl.graph.Graph*

An application-level abstraction of a graph

pacman.model.graphs.application.application_outgoing_edge_partition module

class pacman.model.graphs.application.application_outgoing_edge_partition.**ApplicationOutgoing**

Bases: *pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition*

edge partition for the application graph.

pacman.model.graphs.application.application_spinnaker_link_vertex module

class pacman.model.graphs.application.application_spinnaker_link_vertex.**ApplicationSpiNNakerL**

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *pacman.model.graphs.abstract_spinnaker_link_vertex.AbstractSpiNNakerLinkVertex*

A virtual vertex on a SpiNNaker Link

board_address

The IP address of the board to which the device is connected, or None for the boot board

Return type str

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

set_virtual_chip_coordinates(*virtual_chip_x*, *virtual_chip_y*)

Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

spinnaker_link_id

The SpiNNaker Link that the vertex is connected to

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

pacman.model.graphs.application.application_vertex module

class pacman.model.graphs.application.application_vertex.**ApplicationVertex**(*label=None*, *constraints=None*, *max_atoms_per_core=922*

Bases: *pacman.model.graphs.common.constrained_object.ConstrainedObject*, *pacman.model.graphs.abstract_vertex.AbstractVertex*

A vertex that can be broken down into a number of smaller vertices based on the resources that the vertex requires

Parameters

- **label** (*str*) – The optional name of the vertex
- **constraints** (iterable of *pacman.model.constraints.abstract_constraint.AbstractConstraint*) – The optional initial constraints of the vertex
- **max_atoms_per_core** (*int*) – the max number of atoms that can be placed on a core, used in partitioning

Raises ***pacman.exceptions.PacmanInvalidParameterException*** –

- If one of the constraints is not valid

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

label
The label of the vertex

Return type str

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

Module contents

class pacman.model.graphs.application.**ApplicationEdge**(*pre_vertex*, *post_vertex*, *traffic_type=<EdgeTrafficType.MULTICAST: 1>*, *label=None*, *machine_edge_type=<class 'pacman.model.graphs.machine.machine_edge.MachineEdge'*)

Bases: *pacman.model.graphs.abstract_edge.AbstractEdge*

A simple implementation of an application edge

Parameters

- **pre_vertex** (pacman.model.graph.application.application_vertex.ApplicationVertex) – the application vertex at the start of the edge
- **post_vertex** (pacman.model.graph.application.application_vertex.ApplicationVertex) – the application vertex at the end of the edge
- **traffic_type** (pacman.model.graph.edge_traffic_type.EdgeTrafficType) – The type of the traffic on the edge
- **label** (*str*) – The name of the edge

create_machine_edge(*pre_vertex*, *post_vertex*, *label*)

Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the start of the edge
- **post_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the end of the edge
- **label** (*str*) – label of the edge

Returns The created machine edge

Return type pacman.model.graph.machine.machine_edge.MachineEdge

label

The label of the edge

Returns The label

Return type str

Raises **None** – Raises no known exceptions

post_vertex
The vertex at the end of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

pre_vertex
The vertex at the start of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type
The traffic type of the edge

Return type py:class:*pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

class pacman.model.graphs.application.**ApplicationFPGAVertex**(*n_atoms*, *fpga_id*, *fpga_link_id*, *board_address=None*, *label=None*, *constraints=None*, *max_atoms_per_core=9223372036854775807*)

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *pacman.model.graphs.abstract_fpga_vertex.AbstractFPGAVertex*

A virtual vertex on an FPGA link

board_address
The IP address of the board to which the device is connected, or None for the boot board

Return type str

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

fpga_id
The id of the FPGA to which the vertex is connected

Return type int

fpga_link_id
The link of the FPGA to which the vertex is connected

Return type int

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

set_virtual_chip_coordinates(*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

class pacman.model.graphs.application.**ApplicationGraph**(*label*)
Bases: *pacman.model.graphs.impl.graph.Graph*

An application-level abstraction of a graph

class pacman.model.graphs.application.**ApplicationOutgoingEdgePartition**(*identifier*, *constraints=None*, *label=None*)
Bases: *pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition*

edge partition for the application graph.

class pacman.model.graphs.application.**ApplicationVertex**(*label=None*, *constraints=None*, *max_atoms_per_core=9223372036854775807*)
Bases: *pacman.model.graphs.common.constrained_object.ConstrainedObject*, *pacman.model.graphs.abstract_vertex.AbstractVertex*

A vertex that can be broken down into a number of smaller vertices based on the resources that the vertex requires

Parameters

- **label** (*str*) – The optional name of the vertex
- **constraints** (iterable of *pacman.model.constraints.abstract_constraint.AbstractConstraint*) – The optional initial constraints of the vertex
- **max_atoms_per_core** (*int*) – the max number of atoms that can be placed on a core, used in partitioning

Raises ***pacman.exceptions.PacmanInvalidParameterException*** –

- If one of the constraints is not valid

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

label
The label of the vertex

Return type str

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

class pacman.model.graphs.application.**ApplicationSpiNNakerLinkVertex**(*n_atoms*, *spinnaker_link_id*, *board_address=None*, *label=None*, *constraints=None*, *max_atoms_per_core=9223372036*

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *pacman.model.graphs.abstract_spinnaker_link_vertex.AbstractSpiNNakerLinkVertex*

A virtual vertex on a SpiNNaker Link

board_address
The IP address of the board to which the device is connected, or None for the boot board

Return type str

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

get_resources_used_by_atoms (*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

set_virtual_chip_coordinates (*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

spinnaker_link_id
The SpiNNaker Link that the vertex is connected to

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

pacman.model.graphs.common package

Submodules

pacman.model.graphs.common.constrained_object module

class pacman.model.graphs.common.constrained_object.**ConstrainedObject** (*constraints=None*)
Bases: object

An implementation of an object which holds constraints

Parameters **constraints** – Any initial constraints

add_constraint (*constraint*)
Add a new constraint to the collection of constraints

Parameters **constraint** (*pacman.model.constraints.abstract_constraint.AbstractConstraint*) – constraint to add

Return type None

Raises *pacman.exceptions.PacmanInvalidParameterException* – If the constraint is not valid

add_constraints(*constraints*)
Add an iterable of constraints to the collection of constraints

Parameters **constraints** (iterable of *pacman.model.constraints.abstract_constraint.AbstractConstraint*) – iterable of constraints to add

Return type None

Raises *pacman.exceptions.PacmanInvalidParameterException* – If one of the constraints is not valid

constraints
An iterable of constraints

Returns iterable of constraints

Return type iterable of `pacman.model.constraints.abstract_constraint.AbstractConstraint`

Raises **None** – Raises no known exceptions

pacman.model.graphs.common.edge_traffic_type module

class `pacman.model.graphs.common.edge_traffic_type.`**EdgeTrafficType**
Bases: `enum.Enum`

Indicates the traffic type of an Edge in a graph

FIXED_ROUTE = 2

MULTICAST = 1

pacman.model.graphs.common.graph_mapper module

class `pacman.model.graphs.common.graph_mapper.`**GraphMapper**
Bases: `object`

A mapping between an Application Graph and a Machine Graph

add_edge_mapping(*machine_edge*, *application_edge*)
Add a mapping between a machine edge and an application edge

Parameters

- **machine_edge** – An edge from a Machine Graph
- **application_edge** – An edge from an Application Graph

add_vertex_mapping(*machine_vertex*, *vertex_slice*, *application_vertex*)
Add a mapping between application and machine vertices

Parameters

- **machine_vertex** – A vertex from a Machine Graph
- **vertex_slice** (*pacman.model.graphs.common.slice.Slice*) – The range of atoms from the application vertex that is going to be in the machine_vertex

- **application_vertex** – A vertex from an Application Graph

Raises *pacman.exceptions.PacmanValueError* – If atom selection is out of bounds.

get_application_edge(*machine_edge*)
Get the application edge mapped to a machine edge

Parameters **machine_edge** – An edge from a Machine Graph

Returns A machine edge, or None if none

get_application_vertex(*machine_vertex*)
Get the application vertex mapped to a machine vertex

Parameters **machine_vertex** – A vertex from a Machine Graph

Returns an application vertex, or None if none

get_machine_edges(*application_edge*)
Get all machine edges mapped to a given application edge

Parameters **application_edge** – An edge from an Application Graph

Returns An iterable of machine edges or None if none

get_machine_vertex_index(*machine_vertex*)
Get the index of a machine vertex within the list of such vertices associated with an application vertex

get_machine_vertices(*application_vertex*)
Get all machine vertices mapped to a given application vertex

Parameters **application_vertex** – A vertex from an Application Graph

Returns An iterable of machine vertices or None if none

get_slice(*machine_vertex*)
Get the slice mapped to a machine vertex

Parameters **machine_vertex** – A vertex in a Machine Graph

Returns a slice object containing the low and high atom or None if none

get_slices(*application_vertex*)
Get all the slices mapped to an application vertex

pacman.model.graphs.common.slice module

class pacman.model.graphs.common.slice.**Slice**
Bases: *pacman.model.graphs.common.slice.Slice*

Represents a slice of a vertex.

Attr int lo_atom The lowest atom represented in the slice.

Attr int hi_atom The highest atom represented in the slice.

Attr int n_atoms The number of atoms represented by the slice.

Attr as_slice This slice represented as a `slice()` object (for use in indexing lists, arrays, etc.)

Create a new Slice object.

Parameters

- **lo_atom** (*int*) – Index of the lowest atom to represent.

- **hi_atom** (*int*) – Index of the highest atom to represent.

Raises *PacmanValueError* – If the bounds of the slice are invalid.

Module contents

class pacman.model.graphs.common.**ConstrainedObject**(*constraints=None*)

Bases: object

An implementation of an object which holds constraints

Parameters **constraints** – Any initial constraints

add_constraint(*constraint*)

Add a new constraint to the collection of constraints

Parameters **constraint** (*pacman.model.constraints.abstract_constraint.AbstractConstraint*) – constraint to add

Return type None

Raises *pacman.exceptions.PacmanInvalidParameterException* – If the constraint is not valid

add_constraints(*constraints*)

Add an iterable of constraints to the collection of constraints

Parameters **constraints** (iterable of *pacman.model.constraints.abstract_constraint.AbstractConstraint*) – iterable of constraints to add

Return type None

Raises *pacman.exceptions.PacmanInvalidParameterException* – If one of the constraints is not valid

constraints

An iterable of constraints

Returns iterable of constraints

Return type iterable of pacman.model.constraints.abstract_constraint.AbstractConstraint

Raises **None** – Raises no known exceptions

class pacman.model.graphs.common.**EdgeTrafficType**

Bases: enum.Enum

Indicates the traffic type of an Edge in a graph

FIXED_ROUTE = 2

MULTICAST = 1

class pacman.model.graphs.common.**GraphMapper**

Bases: object

A mapping between an Application Graph and a Machine Graph

add_edge_mapping(*machine_edge*, *application_edge*)

Add a mapping between a machine edge and an application edge

Parameters

- **machine_edge** – An edge from a Machine Graph
- **application_edge** – An edge from an Application Graph

add_vertex_mapping(*machine_vertex*, *vertex_slice*, *application_vertex*)
Add a mapping between application and machine vertices

Parameters

- **machine_vertex** – A vertex from a Machine Graph
- **vertex_slice** (*pacman.model.graphs.common.slice.Slice*) – The range of atoms from the application vertex that is going to be in the machine_vertex
- **application_vertex** – A vertex from an Application Graph

Raises *pacman.exceptions.PacmanValueError* – If atom selection is out of bounds.

get_application_edge(*machine_edge*)
Get the application edge mapped to a machine edge

Parameters **machine_edge** – An edge from a Machine Graph

Returns A machine edge, or None if none

get_application_vertex(*machine_vertex*)
Get the application vertex mapped to a machine vertex

Parameters **machine_vertex** – A vertex from a Machine Graph

Returns an application vertex, or None if none

get_machine_edges(*application_edge*)
Get all machine edges mapped to a given application edge

Parameters **application_edge** – An edge from an Application Graph

Returns An iterable of machine edges or None if none

get_machine_vertex_index(*machine_vertex*)
Get the index of a machine vertex within the list of such vertices associated with an application vertex

get_machine_vertices(*application_vertex*)
Get all machine vertices mapped to a given application vertex

Parameters **application_vertex** – A vertex from an Application Graph

Returns An iterable of machine vertices or None if none

get_slice(*machine_vertex*)
Get the slice mapped to a machine vertex

Parameters **machine_vertex** – A vertex in a Machine Graph

Returns a slice object containing the low and high atom or None if none

get_slices(*application_vertex*)
Get all the slices mapped to an application vertex

class pacman.model.graphs.common.**Slice**
Bases: *pacman.model.graphs.common.slice.Slice*

Represents a slice of a vertex.

Attr int lo_atom The lowest atom represented in the slice.

Attr int hi_atom The highest atom represented in the slice.

Attr int n_atoms The number of atoms represented by the slice.

Attr as_slice This slice represented as a `slice()` object (for use in indexing lists, arrays, etc.)

Create a new Slice object.

Parameters

- **lo_atom** (*int*) – Index of the lowest atom to represent.
- **hi_atom** (*int*) – Index of the highest atom to represent.

Raises **PacmanValueError** – If the bounds of the slice are invalid.

pacman.model.graphs.impl package

Submodules

pacman.model.graphs.impl.graph module

class pacman.model.graphs.impl.graph.**Graph**(*allowed_vertex_types*, *allowed_edge_types*, *allowed_partition_types*, *label*)

Bases: *pacman.model.graphs.common.constrained_object.ConstrainedObject*, *pacman.model.graphs.abstract_graph.AbstractGraph*

A graph implementation that specifies the allowed types of the vertices and edges

Parameters

- **allowed_vertex_types** – A single or tuple of types of vertex to be allowed in the graph
- **allowed_edge_types** – A single or tuple of types of edges to be allowed in the graph
- **allowed_partition_types** – A single or tuple of types of partitions to be allowed in the graph
- **label** – The label on the graph, or None

add_edge(*edge*, *outgoing_edge_partition_name*)

Add an edge to the graph

Parameters

- **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – The edge to add
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition to add the edge to; each edge partition is the partition of edges that start at the same vertex

Raises **PacmanInvalidParameterException** – If the edge is not of a valid type or if edges have already been added to this partition that start at a different vertex to this one

add_outgoing_edge_partition(*outgoing_edge_partition*)

Add an outgoing edge partition to the graph

Parameters **outgoing_edge_partition** (*pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*) – The outgoing edge partition to add

Raises **PacmanAlreadyExistsException** – If a partition already exists with the same pre_vertex and identifier

add_vertex(*vertex*)

Add a vertex to the graph

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex to add

Raises ***PacmanInvalidParameterException*** – If the vertex is not of a valid type

edges

The edges in the graph

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_ending_at_vertex(*vertex*)

Get all the edges that end at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get end

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_ending_at_vertex_with_partition_name(*vertex*, *partition_name*)

Get all the edges that end at the given vertex, and reside in the correct partition id

Parameters

- **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get end
- **partition_name** (*str*) – the label for the partition

Returns iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_starting_at_vertex(*vertex*)

Get all the edges that start at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get start

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_outgoing_edge_partition_starting_at_vertex(*vertex*, *outgoing_edge_partition_name*)

Get the given outgoing edge partition that starts at the given vertex, or None if no such edge partition exists

Parameters

- **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at the start of the edges in the partition
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition

Return type *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

get_outgoing_edge_partitions_starting_at_vertex(*vertex*)

Get all the edge partitions that start at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edge partitions to find starts

Return type iterable of *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

label
The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_outgoing_edge_partitions
The number of outgoing edge partitions in the graph

Return type int

n_vertices
The number of vertices in the graph

Return type int

outgoing_edge_partitions
The outgoing edge partitions in the graph

Return type iterable of *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

vertices
The vertices in the graph

Return type iterable of *pacman.model.graphs.abstract_vertex.AbstractVertex*

pacman.model.graphs.impl.outgoing_edge_partition module

class pacman.model.graphs.impl.outgoing_edge_partition.**OutgoingEdgePartition**(*identifier*, *allowed_edge_types*, *constraints=None*, *label=None*, *traffic_weight=1*)

Bases: *pacman.model.graphs.common.constrained_object.ConstrainedObject*, *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

A collection of edges which start at a single vertex which have the same semantics and so can share a single key

Parameters

- **identifier** – The identifier of the partition
- **allowed_edge_types** – The types of edges allowed
- **constraints** – Any initial constraints
- **label** – An optional label of the partition
- **traffic_weight** – The weight of traffic going down this partition

add_edge(*edge*)
Add an edge to the partition

Parameters **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – the edge to add

Raises *pacman.exceptions.PacmanInvalidParameterException* if the starting vertex of the edge does not match that of the edges already in the partition

edges
The edges in this outgoing edge partition

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

identifier
The identifier of this outgoing edge partition

Return type str

label
The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_edges
The number of edges in the partition

Return type int

pre_vertex
The vertex at which all edges in this partition start

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type
The traffic type of all the edges in this partition

Return type *pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

traffic_weight
The weight of the traffic in this partition compared to other partitions

Return type int

Module contents

class pacman.model.graphs.impl.**Graph**(*allowed_vertex_types*, *allowed_edge_types*, *allowed_partition_types*, *label*)

Bases: *pacman.model.graphs.common.constrained_object.ConstrainedObject*, *pacman.model.graphs.abstract_graph.AbstractGraph*

A graph implementation that specifies the allowed types of the vertices and edges

Parameters

- **allowed_vertex_types** – A single or tuple of types of vertex to be allowed in the graph
- **allowed_edge_types** – A single or tuple of types of edges to be allowed in the graph

- **allowed_partition_types** – A single or tuple of types of partitions to be allowed in the graph
- **label** – The label on the graph, or None

add_edge (*edge*, *outgoing_edge_partition_name*)

Add an edge to the graph

Parameters

- **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – The edge to add
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition to add the edge to; each edge partition is the partition of edges that start at the same vertex

Raises ***PacmanInvalidParameterException*** – If the edge is not of a valid type or if edges have already been added to this partition that start at a different vertex to this one

add_outgoing_edge_partition (*outgoing_edge_partition*)

Add an outgoing edge partition to the graph

Parameters **outgoing_edge_partition** (*pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*) – The outgoing edge partition to add

Raises ***PacmanAlreadyExistsException*** – If a partition already exists with the same pre_vertex and identifier

add_vertex (*vertex*)

Add a vertex to the graph

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex to add

Raises ***PacmanInvalidParameterException*** – If the vertex is not of a valid type

edges

The edges in the graph

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_ending_at_vertex (*vertex*)

Get all the edges that end at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get end

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_ending_at_vertex_with_partition_name (*vertex*, *partition_name*)

Get all the edges that end at the given vertex, and reside in the correct partition id

Parameters

- **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get end
- **partition_name** (*str*) – the label for the partition

Returns iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_starting_at_vertex (*vertex*)

Get all the edges that start at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get start

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_outgoing_edge_partition_starting_at_vertex(*vertex*, *outgoing_edge_partition_name*)

Get the given outgoing edge partition that starts at the given vertex, or None if no such edge partition exists

Parameters

- **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at the start of the edges in the partition
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition

Return type *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

get_outgoing_edge_partitions_starting_at_vertex(*vertex*)

Get all the edge partitions that start at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edge partitions to find starts

Return type iterable of *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

label

The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_outgoing_edge_partitions

The number of outgoing edge partitions in the graph

Return type int

n_vertices

The number of vertices in the graph

Return type int

outgoing_edge_partitions

The outgoing edge partitions in the graph

Return type iterable of *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

vertices

The vertices in the graph

Return type iterable of *pacman.model.graphs.abstract_vertex.AbstractVertex*

class pacman.model.graphs.impl.**OutgoingEdgePartition**(*identifier*, *allowed_edge_types*, *constraints=None*, *label=None*, *traffic_weight=1*)

Bases: *pacman.model.graphs.common.constrained_object.ConstrainedObject*, *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

A collection of edges which start at a single vertex which have the same semantics and so can share a single key

Parameters

- **identifier** – The identifier of the partition
- **allowed_edge_types** – The types of edges allowed
- **constraints** – Any initial constraints
- **label** – An optional label of the partition
- **traffic_weight** – The weight of traffic going down this partition

add_edge(*edge*)

Add an edge to the partition

Parameters **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – the edge to add

Raises *pacman.exceptions.PacmanInvalidParameterException* if the starting vertex of the edge does not match that of the edges already in the partition

edges

The edges in this outgoing edge partition

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

identifier

The identifier of this outgoing edge partition

Return type str

label

The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_edges

The number of edges in the partition

Return type int

pre_vertex

The vertex at which all edges in this partition start

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type

The traffic type of all the edges in this partition

Return type *pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

traffic_weight

The weight of the traffic in this partition compared to other partitions

Return type int

pacman.model.graphs.machine package

Submodules

pacman.model.graphs.machine.machine_edge module

class `pacman.model.graphs.machine.machine_edge.`**`MachineEdge`**(*pre_vertex, post_vertex, traffic_type=<EdgeTrafficType.MULTICAST: 1>, label=None, traffic_weight=1*)

Bases: `pacman.model.graphs.abstract_edge.AbstractEdge`

A simple implementation of a machine edge

Parameters

- **pre_vertex** (`pacman.model.graphs.machine.abstract_machine_vertex.impl.MachineVertex`) – the vertex at the start of the edge
- **post_vertex** (`pacman.model.graphs.machine.abstract_machine_vertex.impl.MachineVertex`) – the vertex at the end of the edge
- **traffic_type** (`pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType`) – The type of traffic that this edge will carry
- **label** (`str`) – The name of the edge
- **traffic_weight** (`int`) – the optional weight of traffic expected to travel down this edge relative to other edges (default is 1)

label

The label of the edge

Returns The label

Return type str

Raises **None** – Raises no known exceptions

post_vertex

The vertex at the end of the edge

Return type `pacman.model.graphs.abstract_vertex.AbstractVertex`

pre_vertex

The vertex at the start of the edge

Return type `pacman.model.graphs.abstract_vertex.AbstractVertex`

traffic_type

The traffic type of the edge

Return type py:class:*pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

traffic_weight

The amount of traffic expected to go down this edge relative to other edges

pacman.model.graphs.machine.machine_fpga_vertex module

class pacman.model.graphs.machine.machine_fpga_vertex.**MachineFPGAVertex**(*fpga_id*, *fpga_link_id*, *board_address=None*, *label=None*, *constraints=None*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *pacman.model.graphs.abstract_fpga_vertex.AbstractFPGAVertex*

A virtual vertex on an FPGA link

board_address
The IP address of the board to which the device is connected, or None for the boot board

Return type str

fpga_id
The id of the FPGA to which the vertex is connected

Return type int

fpga_link_id
The link of the FPGA to which the vertex is connected

Return type int

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

set_virtual_chip_coordinates(*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

pacman.model.graphs.machine.machine_graph module

class pacman.model.graphs.machine.machine_graph.**MachineGraph**(*label*)

Bases: *pacman.model.graphs.impl.graph.Graph*

A graph whose vertices can fit on the chips of a machine

pacman.model.graphs.machine.machine_outgoing_edge_partition module

class pacman.model.graphs.machine.machine_outgoing_edge_partition.**MachineOutgoingEdgePartitio**

Bases: *pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition*

An outgoing edge partition for a Machine Graph

Parameters

- **identifier** – The identifier of the partition
- **constraints** – Any initial constraints
- **label** – An optional label of the partition
- **traffic_weight** – the weight of this partition in relation to other partitions

pacman.model.graphs.machine.machine_spinnaker_link_vertex module

class pacman.model.graphs.machine.machine_spinnaker_link_vertex.**MachineSpiNNakerLinkVertex**(*sp bo la- be co str*

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *pacman.model.graphs.abstract_spinnaker_link_vertex.AbstractSpiNNakerLinkVertex*

A virtual vertex on a SpiNNaker Link

board_address

The IP address of the board to which the device is connected, or None for the boot board

Return type str

resources_required

The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

set_virtual_chip_coordinates(*virtual_chip_x*, *virtual_chip_y*)

Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

spinnaker_link_id

The SpiNNaker Link that the vertex is connected to

virtual_chip_x

The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

pacman.model.graphs.machine.machine_vertex module

class pacman.model.graphs.machine.machine_vertex.**MachineVertex**(*label=None, constraints=None*)

Bases: *pacman.model.graphs.common.constrained_object.ConstrainedObject*, *pacman.model.graphs.abstract_vertex.AbstractVertex*

A machine graph vertex

Parameters

- **label** (*str*) – The optional name of the vertex
- **constraints** (iterable of *pacman.model.constraints.AbstractConstraint*) – The optional initial constraints of the vertex

Raises ***pacman.exceptions.PacmanInvalidParameterException*** –

- If one of the constraints is not valid

label
The label of the vertex

Return type str

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

pacman.model.graphs.machine.simple_machine_vertex module

class pacman.model.graphs.machine.simple_machine_vertex.**SimpleMachineVertex**(*resources, label=None, constraints=None*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*

A MachineVertex that stores its own resources

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

Module contents

class pacman.model.graphs.machine.**MachineEdge**(*pre_vertex, post_vertex, traffic_type=<EdgeTrafficType.MULTICAST: 1>, label=None, traffic_weight=1*)

Bases: *pacman.model.graphs.abstract_edge.AbstractEdge*

A simple implementation of a machine edge

Parameters

- **pre_vertex** (pacman.model.graphs.machine.abstract_machine_vertex.impl.MachineVertex) – the vertex at the start of the edge
- **post_vertex** (pacman.model.graphs.machine.abstract_machine_vertex.impl.MachineVertex) – the vertex at the end of the edge
- **traffic_type** (*pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*) – The type of traffic that this edge will carry
- **label** (*str*) – The name of the edge
- **traffic_weight** (*int*) – the optional weight of traffic expected to travel down this edge relative to other edges (default is 1)

label

The label of the edge

Returns The label

Return type str

Raises **None** – Raises no known exceptions

post_vertex

The vertex at the end of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

pre_vertex

The vertex at the start of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type

The traffic type of the edge

Return type py:class:*pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

traffic_weight

The amount of traffic expected to go down this edge relative to other edges

class pacman.model.graphs.machine.**MachineFPGAVertex**(*fpga_id*, *fpga_link_id*, *board_address=None*, *label=None*, *constraints=None*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *pacman.model.graphs.abstract_fpga_vertex.AbstractFPGAVertex*

A virtual vertex on an FPGA link

board_address

The IP address of the board to which the device is connected, or None for the boot board

Return type str

fpga_id

The id of the FPGA to which the vertex is connected

Return type int

fpga_link_id
The link of the FPGA to which the vertex is connected

Return type int

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

set_virtual_chip_coordinates(*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

class pacman.model.graphs.machine.**MachineGraph**(*label*)
Bases: *pacman.model.graphs.impl.graph.Graph*

A graph whose vertices can fit on the chips of a machine

class pacman.model.graphs.machine.**MachineOutgoingEdgePartition**(*identifier*, *constraints=None*, *label=None*, *traffic_weight=1*)
Bases: *pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition*

An outgoing edge partition for a Machine Graph

Parameters

- **identifier** – The identifier of the partition
- **constraints** – Any initial constraints
- **label** – An optional label of the partition
- **traffic_weight** – the weight of this partition in relation to other partitions

class pacman.model.graphs.machine.**MachineSpiNNakerLinkVertex**(*spinnaker_link_id*, *board_address=None*, *label=None*, *constraints=None*)
Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *pacman.model.graphs.abstract_spinnaker_link_vertex.AbstractSpiNNakerLinkVertex*

A virtual vertex on a SpiNNaker Link

board_address
The IP address of the board to which the device is connected, or None for the boot board

Return type str

resources_required
The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

set_virtual_chip_coordinates(*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

spinnaker_link_id
The SpiNNaker Link that the vertex is connected to

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

class pacman.model.graphs.machine.**MachineVertex**(*label=None*, *constraints=None*)
Bases: pacman.model.graphs.common.constrained_object.ConstrainedObject, pacman.model.graphs.abstract_vertex.AbstractVertex

A machine graph vertex

Parameters

- **label** (*str*) – The optional name of the vertex
- **constraints** (iterable of pacman.model.constraints.AbstractConstraint) – The optional initial constraints of the vertex

Raises **pacman.exceptions.PacmanInvalidParameterException** –

- If one of the constraints is not valid

label
The label of the vertex

Return type str

resources_required
The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

class pacman.model.graphs.machine.**SimpleMachineVertex**(*resources*, *label=None*, *constraints=None*)
Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex

A MachineVertex that stores its own resources

resources_required
The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

Submodules

pacman.model.graphs.abstract_edge module

class pacman.model.graphs.abstract_edge.**AbstractEdge**

Bases: object

A directed edge in a graph between two vertices

label

The label of the edge

Returns The label

Return type str

Raises **None** – Raises no known exceptions

post_vertex

The vertex at the end of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

pre_vertex

The vertex at the start of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type

The traffic type of the edge

Return type py:class:*pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

pacman.model.graphs.abstract_fpga_vertex module

class pacman.model.graphs.abstract_fpga_vertex.**AbstractFPGAVertex**

Bases: *pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex*

A vertex connected to an FPGA

fpga_id

The id of the FPGA to which the vertex is connected

Return type int

fpga_link_id

The link of the FPGA to which the vertex is connected

Return type int

pacman.model.graphs.abstract_graph module

class pacman.model.graphs.abstract_graph.**AbstractGraph**

Bases: object

A graph

add_constraint (*constraint*)

Add a constraint

Parameters **constraint** (AbstractConstraint) – The constraint to add

add_constraints(*constraints*)
Add a list of constraints

Parameters **constraints** (list of `AbstractConstraint`) – The list of constraints to add

add_edge(*edge*, *outgoing_edge_partition_name*)
Add an edge to the graph

Parameters

- **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – The edge to add
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition to add the edge to; each edge partition is the partition of edges that start at the same vertex

Raises ***PacmanInvalidParameterException*** – If the edge is not of a valid type or if edges have already been added to this partition that start at a different vertex to this one

add_edges(*edges*, *outgoing_edge_partition_name*)
Add a collection of edges to the graph

Parameters

- **edges** (an iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*) – The edges to add
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition to add the edges to; each edge partition is the partition of edges that start at the same vertex

Raises ***PacmanInvalidParameterException*** – If any edge is not of a valid type or if edges have already been added to this partition that start at a different vertex to this one

add_outgoing_edge_partition(*outgoing_edge_partition*)
Add an outgoing edge partition to the graph

Parameters **outgoing_edge_partition** (*pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*) – The outgoing edge partition to add

Raises ***PacmanAlreadyExistsException*** – If a partition already exists with the same pre_vertex and identifier

add_vertex(*vertex*)
Add a vertex to the graph

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex to add

Raises ***PacmanInvalidParameterException*** – If the vertex is not of a valid type

add_vertices(*vertices*)
Add a collection of vertices to the graph.

Parameters **vertices** (an iterable of *pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertices to add

Raises ***PacmanInvalidParameterException*** – If any vertex is not of a valid type

constraints
The constraints of the vertex

Return type iterable of `AbstractConstraint`

edges

The edges in the graph

Return type iterable of `pacman.model.graphs.abstract_edge.AbstractEdge`

get_edges_ending_at_vertex(*vertex*)

Get all the edges that end at the given vertex

Parameters **vertex** (`pacman.model.graphs.abstract_vertex.AbstractVertex`) – The vertex at which the edges to get end

Return type iterable of `pacman.model.graphs.abstract_edge.AbstractEdge`

get_edges_ending_at_vertex_with_partition_name(*vertex*, *partition_name*)

Get all the edges that end at the given vertex, and reside in the correct partition id

Parameters

- **vertex** (`pacman.model.graphs.abstract_vertex.AbstractVertex`) – The vertex at which the edges to get end
- **partition_name** (`str`) – the label for the partition

Returns iterable of `pacman.model.graphs.abstract_edge.AbstractEdge`

get_edges_starting_at_vertex(*vertex*)

Get all the edges that start at the given vertex

Parameters **vertex** (`pacman.model.graphs.abstract_vertex.AbstractVertex`) – The vertex at which the edges to get start

Return type iterable of `pacman.model.graphs.abstract_edge.AbstractEdge`

get_outgoing_edge_partition_starting_at_vertex(*vertex*, *outgoing_edge_partition_name*)

Get the given outgoing edge partition that starts at the given vertex, or None if no such edge partition exists

Parameters

- **vertex** (`pacman.model.graphs.abstract_vertex.AbstractVertex`) – The vertex at the start of the edges in the partition
- **outgoing_edge_partition_name** (`str`) – The name of the edge partition

Return type `pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition`

get_outgoing_edge_partitions_starting_at_vertex(*vertex*)

Get all the edge partitions that start at the given vertex

Parameters **vertex** (`pacman.model.graphs.abstract_vertex.AbstractVertex`) – The vertex at which the edge partitions to find starts

Return type iterable of `pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition`

label

The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_outgoing_edge_partitions
The number of outgoing edge partitions in the graph

Return type int

n_vertices
The number of vertices in the graph

Return type int

outgoing_edge_partitions
The outgoing edge partitions in the graph

Return type iterable of *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

vertices
The vertices in the graph

Return type iterable of *pacman.model.graphs.abstract_vertex.AbstractVertex*

pacman.model.graphs.abstract_outgoing_edge_partition module

class pacman.model.graphs.abstract_outgoing_edge_partition.**AbstractOutgoingEdgePartition**
Bases: object

A group of edges that start at the same vertex and share the same semantics; used to group edges that can use the same multicast key

add_constraint(*constraint*)
Add a constraint

Parameters **constraint** (AbstractConstraint) – The constraint to add

add_constraints(*constraints*)
Add a list of constraints

Parameters **constraints** (list of AbstractConstraint) – The list of constraints to add

add_edge(*edge*)
Add an edge to the partition

Parameters **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – the edge to add

Raises *pacman.exceptions.PacmanInvalidParameterException* if the starting vertex of the edge does not match that of the edges already in the partition

constraints
The constraints of the vertex

Return type iterable of AbstractConstraint

edges
The edges in this outgoing edge partition

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

identifier
The identifier of this outgoing edge partition

Return type str

label
The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_edges
The number of edges in the partition

Return type int

pre_vertex
The vertex at which all edges in this partition start

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type
The traffic type of all the edges in this partition

Return type *pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

traffic_weight
The weight of the traffic in this partition compared to other partitions

Return type int

pacman.model.graphs.abstract_spinnaker_link_vertex module

class pacman.model.graphs.abstract_spinnaker_link_vertex.**AbstractSpiNNakerLinkVertex**
Bases: *pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex*

A vertex connected to a SpiNNaker Link

spinnaker_link_id
The SpiNNaker Link that the vertex is connected to

pacman.model.graphs.abstract_vertex module

class pacman.model.graphs.abstract_vertex.**AbstractVertex**
Bases: object

A vertex in a graph

add_constraint (*constraint*)
Add a constraint

Parameters **constraint** (AbstractConstraint) – The constraint to add

add_constraints (*constraints*)
Add a list of constraints

Parameters **constraints** (list of AbstractConstraint) – The list of constraints to add

constraints
The constraints of the vertex

Return type iterable of AbstractConstraint

label
The label of the vertex

Return type str

pacman.model.graphs.abstract_virtual_vertex module

class pacman.model.graphs.abstract_virtual_vertex.**AbstractVirtualVertex**
Bases: *pacman.model.graphs.abstract_vertex.AbstractVertex*

A vertex which exists outside of the machine

board_address
The IP address of the board to which the device is connected, or None for the boot board

Return type str

set_virtual_chip_coordinates (*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

Module contents

class pacman.model.graphs.**AbstractEdge**
Bases: object

A directed edge in a graph between two vertices

label
The label of the edge

Returns The label

Return type str

Raises **None** – Raises no known exceptions

post_vertex
The vertex at the end of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

pre_vertex
The vertex at the start of the edge

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type
The traffic type of the edge

Return type py:class:*pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

class pacman.model.graphs.**AbstractFPGAVertex**
Bases: *pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex*

A vertex connected to an FPGA

fpga_id
The id of the FPGA to which the vertex is connected

Return type int

fpga_link_id
The link of the FPGA to which the vertex is connected

Return type int

class pacman.model.graphs.**AbstractGraph**
Bases: object

A graph

add_constraint(*constraint*)
Add a constraint

Parameters **constraint** (AbstractConstraint) – The constraint to add

add_constraints(*constraints*)
Add a list of constraints

Parameters **constraints** (list of AbstractConstraint) – The list of constraints to add

add_edge(*edge*, *outgoing_edge_partition_name*)
Add an edge to the graph

Parameters

- **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – The edge to add
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition to add the edge to; each edge partition is the partition of edges that start at the same vertex

Raises ***PacmanInvalidParameterException*** – If the edge is not of a valid type or if edges have already been added to this partition that start at a different vertex to this one

add_edges(*edges*, *outgoing_edge_partition_name*)
Add a collection of edges to the graph

Parameters

- **edges** (an iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*) – The edges to add
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition to add the edges to; each edge partition is the partition of edges that start at the same vertex

Raises ***PacmanInvalidParameterException*** – If any edge is not of a valid type or if edges have already been added to this partition that start at a different vertex to this one

add_outgoing_edge_partition(*outgoing_edge_partition*)
Add an outgoing edge partition to the graph

Parameters **outgoing_edge_partition** (*pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*) – The outgoing edge partition to add

Raises ***PacmanAlreadyExistsException*** – If a partition already exists with the same pre_vertex and identifier

add_vertex(*vertex*)

Add a vertex to the graph

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex to add

Raises ***PacmanInvalidParameterException*** – If the vertex is not of a valid type

add_vertices(*vertices*)

Add a collection of vertices to the graph.

Parameters **vertices** (an iterable of *pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertices to add

Raises ***PacmanInvalidParameterException*** – If any vertex is not of a valid type

constraints

The constraints of the vertex

Return type iterable of `AbstractConstraint`

edges

The edges in the graph

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_ending_at_vertex(*vertex*)

Get all the edges that end at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get end

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_ending_at_vertex_with_partition_name(*vertex*, *partition_name*)

Get all the edges that end at the given vertex, and reside in the correct partition id

Parameters

- **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get end
- **partition_name** (*str*) – the label for the partition

Returns iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_edges_starting_at_vertex(*vertex*)

Get all the edges that start at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edges to get start

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

get_outgoing_edge_partition_starting_at_vertex(*vertex*, *outgoing_edge_partition_name*)

Get the given outgoing edge partition that starts at the given vertex, or None if no such edge partition exists

Parameters

- **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at the start of the edges in the partition
- **outgoing_edge_partition_name** (*str*) – The name of the edge partition

Return type *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

get_outgoing_edge_partitions_starting_at_vertex(*vertex*)

Get all the edge partitions that start at the given vertex

Parameters **vertex** (*pacman.model.graphs.abstract_vertex.AbstractVertex*) – The vertex at which the edge partitions to find starts

Return type iterable of *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

label

The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_outgoing_edge_partitions

The number of outgoing edge partitions in the graph

Return type int

n_vertices

The number of vertices in the graph

Return type int

outgoing_edge_partitions

The outgoing edge partitions in the graph

Return type iterable of *pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition*

vertices

The vertices in the graph

Return type iterable of *pacman.model.graphs.abstract_vertex.AbstractVertex*

class pacman.model.graphs.**AbstractOutgoingEdgePartition**

Bases: object

A group of edges that start at the same vertex and share the same semantics; used to group edges that can use the same multicast key

add_constraint(*constraint*)

Add a constraint

Parameters **constraint** (AbstractConstraint) – The constraint to add

add_constraints(*constraints*)

Add a list of constraints

Parameters **constraints** (list of AbstractConstraint) – The list of constraints to add

add_edge(*edge*)
Add an edge to the partition

Parameters **edge** (*pacman.model.graphs.abstract_edge.AbstractEdge*) – the edge to add

Raises *pacman.exceptions.PacmanInvalidParameterException* if the starting vertex of the edge does not match that of the edges already in the partition

constraints
The constraints of the vertex

Return type iterable of `AbstractConstraint`

edges
The edges in this outgoing edge partition

Return type iterable of *pacman.model.graphs.abstract_edge.AbstractEdge*

identifier
The identifier of this outgoing edge partition

Return type str

label
The label of the item

Returns The label

Return type str

Raises **None** – Raises no known exceptions

n_edges
The number of edges in the partition

Return type int

pre_vertex
The vertex at which all edges in this partition start

Return type *pacman.model.graphs.abstract_vertex.AbstractVertex*

traffic_type
The traffic type of all the edges in this partition

Return type *pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType*

traffic_weight
The weight of the traffic in this partition compared to other partitions

Return type int

class pacman.model.graphs.**AbstractSpiNNakerLinkVertex**
Bases: *pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex*

A vertex connected to a SpiNNaker Link

spinnaker_link_id
The SpiNNaker Link that the vertex is connected to

class pacman.model.graphs.**AbstractVertex**
Bases: `object`

A vertex in a graph

add_constraint (*constraint*)
Add a constraint

Parameters **constraint** (AbstractConstraint) – The constraint to add

add_constraints (*constraints*)
Add a list of constraints

Parameters **constraints** (list of AbstractConstraint) – The list of constraints to add

constraints
The constraints of the vertex

Return type iterable of AbstractConstraint

label
The label of the vertex

Return type str

class pacman.model.graphs.**AbstractVirtualVertex**
Bases: *pacman.model.graphs.abstract_vertex.AbstractVertex*

A vertex which exists outside of the machine

board_address
The IP address of the board to which the device is connected, or None for the boot board

Return type str

set_virtual_chip_coordinates (*virtual_chip_x*, *virtual_chip_y*)
Set the details of the virtual chip that has been added to the machine for this vertex

Parameters

- **virtual_chip_x** – The x-coordinate of the added chip
- **virtual_chip_y** – The y-coordinate of the added chip

virtual_chip_x
The x-coordinate of the virtual chip where this vertex is to be placed

Return type int

virtual_chip_y
The y-coordinate of the virtual chip where this vertex is to be placed

Return type int

pacman.model.placements package

Submodules

pacman.model.placements.placement module

class pacman.model.placements.placement.**Placement** (*vertex*, *x*, *y*, *p*)
Bases: object

The placement of a vertex on to a machine chip and core

Parameters

- **vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The vertex that has been placed
- **x** (*int*) – the x-coordinate of the chip on which the vertex is placed
- **y** (*int*) – the y-coordinate of the chip on which the vertex is placed
- **p** (*int or None*) – the id of the processor on which the vertex is placed

p

The id of the processor of the chip where the vertex is placed

Returns The processor id

Return type int

vertex

The vertex that was placed

Returns a vertex

Return type pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex

Raises **None** – does not raise any known exceptions

x

The x-coordinate of the chip where the vertex is placed

Returns The x-coordinate

Return type int

y

The y-coordinate of the chip where the vertex is placed

Returns The y-coordinate

Return type int

pacman.model.placements.placements module

class pacman.model.placements.placements.**Placements**(*placements=None*)

Bases: object

The placements of vertices on the chips of the machine

Parameters **placements** (iterable of *pacman.model.placements.placement.Placement*) – Any initial placements

Raises

- ***PacmanAlreadyPlacedError*** – If there is any vertex with more than one placement.
- ***PacmanProcessorAlreadyOccupiedError*** – If two placements are made to the same processor.

add_placement(*placement*)

Add a placement

Parameters **placement** (*pacman.model.placements.placement.Placement*) – The placement to add

Raises

- **PacmanAlreadyPlacedError** – If there is any vertex with more than one placement.
- **PacmanProcessorAlreadyOccupiedError** – If two placements are made to the same processor.

add_placements(*placements*)

Add some placements

Parameters **placements** (iterable of *pacman.model.placements.placement.Placement*) – The placements to add

get_placed_processors()

Return an iterable of processors with assigned vertices.

Returns Iterable of (x, y, p) tuples

Return type iterable of (int, int, int)

get_placement_of_vertex(*vertex*)

Return the placement information for a vertex

Parameters **vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The vertex to find the placement of

Returns The placement

Return type *pacman.model.placements.placement.Placement*

Raises **PacmanNotPlacedError** – If the vertex has not been placed.

get_vertex_on_processor(*x*, *y*, *p*)

Return the vertex on a specific processor or None if the processor has not been allocated

Parameters

- **x** (*int*) – the x coordinate of the chip
- **y** (*int*) – the y coordinate of the chip
- **p** (*int*) – the processor on the chip

Returns the vertex placed on the given processor

Return type pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex

Raises **PacmanProcessorNotOccupiedError** – If the processor is not occupied

is_processor_occupied(*x*, *y*, *p*)

Determine if a processor has a vertex on it

Parameters

- **x** (*int*) – x coordinate of processor.
- **y** (*int*) – y coordinate of processor.
- **p** (*int*) – Index of processor.

Return bool Whether the processor has an assigned vertex.

n_placements

The number of placements

Return type int

placements
All of the placements

Returns iterable of placements

Return type iterable of *pacman.model.placements.placement.Placement*

Raises **None** – does not raise any known exceptions

Module contents

class pacman.model.placements.**Placement**(*vertex*, *x*, *y*, *p*)
Bases: object

The placement of a vertex on to a machine chip and core

Parameters

- **vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The vertex that has been placed
- **x** (*int*) – the x-coordinate of the chip on which the vertex is placed
- **y** (*int*) – the y-coordinate of the chip on which the vertex is placed
- **p** (*int or None*) – the id of the processor on which the vertex is placed

p
The id of the processor of the chip where the vertex is placed

Returns The processor id

Return type int

vertex
The vertex that was placed

Returns a vertex

Return type pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex

Raises **None** – does not raise any known exceptions

x
The x-coordinate of the chip where the vertex is placed

Returns The x-coordinate

Return type int

y
The y-coordinate of the chip where the vertex is placed

Returns The y-coordinate

Return type int

class pacman.model.placements.**Placements**(*placements=None*)
Bases: object

The placements of vertices on the chips of the machine

Parameters **placements** (iterable of *pacman.model.placements.placement.Placement*) – Any initial placements

Raises

- **PacmanAlreadyPlacedError** – If there is any vertex with more than one placement.
- **PacmanProcessorAlreadyOccupiedError** – If two placements are made to the same processor.

add_placement(*placement*)

Add a placement

Parameters **placement** (*pacman.model.placements.placement.Placement*) – The placement to add

Raises

- **PacmanAlreadyPlacedError** – If there is any vertex with more than one placement.
- **PacmanProcessorAlreadyOccupiedError** – If two placements are made to the same processor.

add_placements(*placements*)

Add some placements

Parameters **placements** (iterable of *pacman.model.placements.placement.Placement*) – The placements to add

get_placed_processors()

Return an iterable of processors with assigned vertices.

Returns Iterable of (x, y, p) tuples

Return type iterable of (int, int, int)

get_placement_of_vertex(*vertex*)

Return the placement information for a vertex

Parameters **vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The vertex to find the placement of

Returns The placement

Return type *pacman.model.placements.placement.Placement*

Raises **PacmanNotPlacedError** – If the vertex has not been placed.

get_vertex_on_processor(*x*, *y*, *p*)

Return the vertex on a specific processor or None if the processor has not been allocated

Parameters

- **x** (*int*) – the x coordinate of the chip
- **y** (*int*) – the y coordinate of the chip
- **p** (*int*) – the processor on the chip

Returns the vertex placed on the given processor

Return type pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex

Raises **PacmanProcessorNotOccupiedError** – If the processor is not occupied

is_processor_occupied(*x*, *y*, *p*)

Determine if a processor has a vertex on it

Parameters

- **x** (*int*) – x coordinate of processor.
- **y** (*int*) – y coordinate of processor.
- **p** (*int*) – Index of processor.

Return bool Whether the processor has an assigned vertex.

n_placements

The number of placements

Return type int

placements

All of the placements

Returns iterable of placements

Return type iterable of *pacman.model.placements.placement.Placement*

Raises **None** – does not raise any known exceptions

pacman.model.resources package

Submodules

pacman.model.resources.abstract_resource module

class pacman.model.resources.abstract_resource.**AbstractResource**

Bases: object

Represents some finite resource

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

pacman.model.resources.core_resource module

class pacman.model.resources.core_resource.**CoreResource**(*chip*, *n_cores*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the number of cores that need to be allocated

Parameters

- **n_cores** (*int*) – The number of cores to allocate
- **chip** (*SpiNNMachine.chip.Chip*) – chip of where these cores are to be allocated

Raises **None** – No known exceptions are raised

chip

get_value()

n_cores

pacman.model.resources.cpu_cycles_per_tick_resource module

class pacman.model.resources.cpu_cycles_per_tick_resource.**CPUCyclesPerTickResource**(*cycles*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the number of CPU clock cycles per tick used or available on a core of a chip in the machine

Parameters **cycles** (*int*) – The number of CPU clock cycles

Raises **None** – No known exceptions are raised

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

pacman.model.resources.dtcm_resource module

class pacman.model.resources.dtcm_resource.**DTCMResource**(*dtcm*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the amount of local core memory available or used on a core on a chip of the machine

Parameters **dtcm** (*int*) – The amount of DTCM in bytes

Raises **None** – No known exceptions are raised

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

pacman.model.resources.element_free_space module

class pacman.model.resources.element_free_space.**ElementFreeSpace**(*start_address*, *size*)

Bases: object

size

start_address

pacman.model.resources.iptag_resource module

class pacman.model.resources.iptag_resource.**IPtagResource**(*ip_address*, *port*, *strip_sdp*, *tag=None*, *traffic_identifier='DEFAULT'*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the amount of local core memory available or used on a core on a chip of the machine

Parameters

- **ip_address** (*str*) – The ip address of the host that will receive data from this tag
- **port** (*int or None*) – The port that will

- **strip_sdp** (*bool*) – Whether the tag requires that SDP headers are stripped before transmission of data
- **tag** (*int*) – A fixed tag id to assign, or None if any tag is OK
- **traffic_identifier** (*str*) – The traffic to be sent using this tag; traffic with the same traffic_identifier can be sent using the same tag

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

ip_address

The ip address to assign to the tag

Returns An ip address

Return type str

port

The port of the tag

Returns The port of the tag

Return type int

strip_sdp

Whether SDP headers should be stripped for this tag

Returns True if the headers should be stripped, False otherwise

Return type bool

tag

The tag required, or None if any tag is OK

Returns The tag or None

Return type int

traffic_identifier

the traffic identifier for this iptag

pacman.model.resources.pre_allocated_resource_container module

class pacman.model.resources.pre_allocated_resource_container.**PreAllocatedResourceContainer**(*s*
s
c
c
s
c
s
c

Bases: object

container object for pre-allocated resources

Container object for the types of resources

Parameters

- **specific_sdram_usage** (*iterable of pacman.model.resources.SpecificSDRAMResource*) – iterable of SpecificSDRAMResource which states that specific chips have missing SDRAM
- **specific_core_resources** (*iterable of pacman.model.resources.SpecificCoreResource*) – states which cores have been preallocated
- **core_resources** (*iterable of pacman.model.resources.CoreResource*) – states a number of cores have been pre allocated but don't care which ones they are

core_resources

extend(*other*)

specific_core_resources

specific_iptag_resources

specific_reverse_iptag_resources

specific_sdram_usage

pacman.model.resources.resource_container module

class pacman.model.resources.resource_container.**ResourceContainer**(*dtcm=None*, *sdram=None*, *cpu_cycles=None*, *iptags=None*, *reverse_iptags=None*)

Bases: object

container object for the types of resources so that ordering is no longer a risk

Container object for the types of resources

Parameters

- **dtcm** (None or pacman.models.resources.dtcm_resource.DTCMResource) – the amount of dtcm used
- **sdram** (None or pacman.models.resources.sdram_resource.SDRAMResource) – the amount of sdram used
- **cpu_cycles** (None or pacman.models.resources.cpu_cycles_per_tick_resource.CPUCyclesPerTickResource) – the amount of cpu used
- **iptags** (None or list of pacman.models.resources.iptag_resource.IPtagResource) – the iptags required
- **reverse_iptags** (None or list of pacman.models.resources.reverse_iptag_resource.ReverseIPtagResource) – the reverse iptags required

Return type pacman.models.resources.resource_container.ResourceContainer

Raises **None** – does not raise any known exception

cpu_cycles

dtcm

extend(*other*)

iptags

reverse_iptags

sdram

sdram_tags

pacman.model.resources.reverse_iptag_resource module

class pacman.model.resources.reverse_iptag_resource.**ReverseIPtagResource**(*port=None*, *sdp_port=1*, *tag=None*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the amount of local core memory available or used on a core on a chip of the machine

Parameters

- **port** (*int*) – The target port of the tag or None to assign elsewhere
- **port** – The UDP port to listen to on the board for this tag
- **sdp_port** (*int*) – The SDP port number to be used when constructing SDP packets from the received UDP packets for this tag
- **tag** (*int or None*) – A fixed tag id to assign, or None if any tag is OK

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

port

The port of the tag

Returns The port of the tag

Return type int

sdp_port

The SDP port to use when constructing the SDP message from the received UDP message

tag

The tag required, or None if any tag is OK

Returns The tag or None

Return type int

pacman.model.resources.sdram_resource module

class pacman.model.resources.sdram_resource.**SDRAMResource**(*sdram*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents an amount of SDRAM used or available on a chip in the machine

Parameters **sdram** (*int*) – The amount of SDRAM in bytes

Raises **None** – No known exceptions are raised

get_value()
Get the amount of the resource used or available

Returns The amount of the resource

Return type int See `pacman.model.resources.abstract_resource.AbstractResource.get_value()`

pacman.model.resources.specific_board_iptag_resource module

class `pacman.model.resources.specific_board_iptag_resource.`**SpecificBoardTagResource**(*board*, *ip_address*, *port*, *strip_sdp*, *tag=None*, *traffic_identifier*)

Bases: `pacman.model.resources.abstract_resource.AbstractResource`

A resource that allocates a tag on a specific board before the class needing it has been built.

Parameters

- **board** (*str*) – The IP address of the board to which this tag is to be associated with
- **ip_address** (*str*) – The IP address of the host that will receive data from this tag
- **port** (*int or None*) – The port that will
- **strip_sdp** (*bool*) – Whether the tag requires that SDP headers are stripped before transmission of data
- **tag** (*int*) – A fixed tag id to assign, or None if any tag is OK
- **traffic_identifier** (*str*) – The traffic to be sent using this tag; traffic with the same traffic_identifier can be sent using the same tag

board
The board IP address that this tag is to reside on

Returns IP address

get_value()
Get the amount of the resource used or available

Returns The amount of the resource

Return type int

ip_address
The IP address to assign to the tag.

Returns An IP address

Return type str

port
The port of the tag

Returns The port of the tag

Return type int

strip_sdp
Whether SDP headers should be stripped for this tag

Returns True if the headers should be stripped, False otherwise

Return type bool

tag
The tag required, or None if any tag is OK.

Returns The tag or None

Return type int

traffic_identifier
The traffic identifier for this IPTag

pacman.model.resources.specific_board_reverse_iptag_resource module

class `pacman.model.resources.specific_board_reverse_iptag_resource.`**`ReverseIPtagResource`**(*board, port=N sdp_po tag=N*

Bases: `pacman.model.resources.abstract_resource.AbstractResource`

Represents the amount of local core memory available or used on a core on a chip of the machine

Parameters

- **board** (*str*) – A board IP address which is where this reverse IPTag is to be placed
- **port** (*int*) – The target port of the tag or None to assign elsewhere
- **port** – The UDP port to listen to on the board for this tag
- **sdp_port** (*int*) – The SDP port number to be used when constructing SDP packets from the received UDP packets for this tag.
- **tag** (*int or None*) – A fixed tag id to assign, or None if any tag is OK

board
A board IP address which is where this reverse IPTag is to be placed

Returns str

get_value()
Get the amount of the resource used or available

Returns The amount of the resource

Return type int

port
The port of the tag

Returns The port of the tag

Return type int

sdp_port
The SDP port to use when constructing the SDP message from the received UDP message.

tag
The tag required, or None if any tag is OK

Returns The tag or None

Return type int

pacman.model.resources.specific_chip_sdram_resource module

class pacman.model.resources.specific_chip_sdram_resource.**SpecificChipSDRAMResource**(*chip*, *sdram_usage*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the number of cores that need to be allocated

Parameters

- **sdram_usage** (*int*) – The amount of SDRAM in bytes needed to be pre-allocated
- **chip** (*SpiNNMachine.chip.Chip*) – chip of where the SDRAM is to be allocated

Raises **None** – No known exceptions are raised

chip

get_value()

sdram_usage

pacman.model.resources.specific_core_resource module

class pacman.model.resources.specific_core_resource.**SpecificCoreResource**(*chip*, *cores*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents specific cores that need to be allocated

Parameters

- **cores** (*iterable of int*) – The specific cores that need to be allocated (list of processor ids)
- **chip** (*SpiNNMachine.chip.Chip*) – chip of where these cores are to be allocated

Raises **None** – No known exceptions are raised

chip

cores

get_value()

Module contents

class pacman.model.resources.**AbstractResource**

Bases: object

Represents some finite resource

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

class pacman.model.resources.**CPUCyclesPerTickResource**(*cycles*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the number of CPU clock cycles per tick used or available on a core of a chip in the machine

Parameters **cycles** (*int*) – The number of CPU clock cycles

Raises **None** – No known exceptions are raised

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

class pacman.model.resources.**DTCMResource**(*dtcm*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the amount of local core memory available or used on a core on a chip of the machine

Parameters **dtcm** (*int*) – The amount of DTCM in bytes

Raises **None** – No known exceptions are raised

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

class pacman.model.resources.**ElementFreeSpace**(*start_address*, *size*)

Bases: object

size

start_address

class pacman.model.resources.**IPtagResource**(*ip_address*, *port*, *strip_sdp*, *tag=None*, *traffic_identifier='DEFAULT'*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the amount of local core memory available or used on a core on a chip of the machine

Parameters

- **ip_address** (*str*) – The ip address of the host that will receive data from this tag
- **port** (*int or None*) – The port that will
- **strip_sdp** (*bool*) – Whether the tag requires that SDP headers are stripped before transmission of data
- **tag** (*int*) – A fixed tag id to assign, or None if any tag is OK
- **traffic_identifier** (*str*) – The traffic to be sent using this tag; traffic with the same traffic_identifier can be sent using the same tag

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

ip_address

The ip address to assign to the tag

Returns An ip address

Return type str

port

The port of the tag

Returns The port of the tag

Return type int

strip_sdp

Whether SDP headers should be stripped for this tag

Returns True if the headers should be stripped, False otherwise

Return type bool

tag

The tag required, or None if any tag is OK

Returns The tag or None

Return type int

traffic_identifier

the traffic identifier for this iptag

class `pacman.model.resources.`**`ResourceContainer`**(*dtcm=None*, *sdram=None*, *cpu_cycles=None*, *iptags=None*, *reverse_iptags=None*)

Bases: `object`

container object for the types of resources so that ordering is no longer a risk

Container object for the types of resources

Parameters

- **dtcm** (None or `pacman.models.resources.dtcm_resource.DTCMResource`) – the amount of dtcm used
- **sdram** (None or `pacman.models.resources.sdram_resource.SDRAMResource`) – the amount of sdram used
- **cpu_cycles** (None or `pacman.models.resources.cpu_cycles_per_tick_resource.CPUCyclesPerTickResource`) – the amount of cpu used
- **iptags** (None or list of `pacman.models.resources.iptag_resource.IPtagResource`) – the iptags required
- **reverse_iptags** (None or list of `pacman.models.resources.reverse_iptag_resource.ReverseIPtagResource`) – the reverse iptags required

Return type pacman.models.resources.resource_container.ResourceContainer

Raises **None** – does not raise any known exception

cpu_cycles

dtcm

extend(*other*)

iptags

reverse_iptags

sdram

sdram_tags

class pacman.model.resources.**ReverseIPtagResource**(*port=None*, *sdp_port=1*, *tag=None*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the amount of local core memory available or used on a core on a chip of the machine

Parameters

- **port** (*int*) – The target port of the tag or None to assign elsewhere
- **port** – The UDP port to listen to on the board for this tag
- **sdp_port** (*int*) – The SDP port number to be used when constructing SDP packets from the received UDP packets for this tag
- **tag** (*int or None*) – A fixed tag id to assign, or None if any tag is OK

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int

port

The port of the tag

Returns The port of the tag

Return type int

sdp_port

The SDP port to use when constructing the SDP message from the received UDP message

tag

The tag required, or None if any tag is OK

Returns The tag or None

Return type int

class pacman.model.resources.**SDRAMResource**(*sdram*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents an amount of SDRAM used or available on a chip in the machine

Parameters **sdram** (*int*) – The amount of SDRAM in bytes

Raises **None** – No known exceptions are raised

get_value()

Get the amount of the resource used or available

Returns The amount of the resource

Return type int See pacman.model.resources.abstract_resource.AbstractResource.get_value()

class pacman.model.resources.**CoreResource**(*chip*, *n_cores*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the number of cores that need to be allocated

Parameters

- **n_cores** (*int*) – The number of cores to allocate
- **chip** (*SpiNNMachine.chip.Chip*) – chip of where these cores are to be allocated

Raises **None** – No known exceptions are raised

chip

get_value()

n_cores

class pacman.model.resources.**PreAllocatedResourceContainer**(*specific_sdram_usage=None, specific_core_resources=None, core_resources=None, specific_iptag_resources=None, specific_reverse_iptag_resources=None*)

Bases: object

container object for pre-allocated resources

Container object for the types of resources

Parameters

- **specific_sdram_usage** (*iterable of pacman.model.resources.SpecificSDRAMResource*) – iterable of SpecificSDRAMResource which states that specific chips have missing SDRAM
- **specific_core_resources** (*iterable of pacman.model.resources.SpecificCoreResource*) – states which cores have been preallocated
- **core_resources** (*iterable of pacman.model.resources.CoreResource*) – states a number of cores have been pre allocated but don't care which ones they are

core_resources

extend(*other*)

specific_core_resources

specific_iptag_resources

specific_reverse_iptag_resources

specific_sdram_usage

class pacman.model.resources.**SpecificChipSDRAMResource**(*chip, sdram_usage*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents the number of cores that need to be allocated

Parameters

- **sdram_usage** (*int*) – The amount of SDRAM in bytes needed to be pre-allocated
- **chip** (*SpiNNMachine.chip.Chip*) – chip of where the SDRAM is to be allocated

Raises **None** – No known exceptions are raised

chip

get_value()

sdram_usage

class pacman.model.resources.**SpecificCoreResource**(*chip*, *cores*)

Bases: *pacman.model.resources.abstract_resource.AbstractResource*

Represents specific cores that need to be allocated

Parameters

- **cores** (*iterable of int*) – The specific cores that need to be allocated (list of processor ids)
- **chip** (*SpiNNMachine.chip.Chip*) – chip of where these cores are to be allocated

Raises **None** – No known exceptions are raised

chip

cores

get_value()

pacman.model.routing_info package

Submodules

pacman.model.routing_info.abstract_machine_partition_n_keys_map module

class pacman.model.routing_info.abstract_machine_partition_n_keys_map.**AbstractMachinePartitio**

Bases: object

A map that provides the number of keys required by each partition

n_keys_for_partition(*partition*)

The number of keys required by the given partition

Parameters **partition** (pacman.model.graph.simple_outgoing_edge_partition.OutgoingEdgePartition) – The partition to set the number of keys for

Returns The number of keys required by the partition

Return type int

pacman.model.routing_info.base_key_and_mask module

class pacman.model.routing_info.base_key_and_mask.**BaseKeyAndMask**(*base_key*, *mask*)

Bases: object

A Key and Mask to be used for routing

Parameters

- **base_key** (*int*) – The routing key
- **mask** (*int*) – The routing mask

Raises *PacmanConfigurationException* – If key & mask != key i.e. the key is not valid for the given mask

get_keys (*key_array=None*, *offset=0*, *n_keys=None*)
Get the ordered list of keys that the combination allows

Parameters

- **key_array** (*array-like of int*) – Optional array into which the returned keys will be placed
- **offset** (*int*) – Optional offset into the array at which to start placing keys
- **n_keys** (*int*) – Optional limit on the number of keys returned. If less than this number of keys are available, only the keys available will be added

Returns A tuple of an array of keys and the number of keys added to the array

Return type (array-like of int, int)

key
The base key

Returns The base key

Return type int

key_combo
The key combined with the mask

mask
The mask

Returns The mask

Return type int

n_keys
The total number of keys that can be generated given the mask

Returns The number of keys

Return type int

pacman.model.routing_info.dict_based_machine_partition_n_keys_map module

class pacman.model.routing_info.dict_based_machine_partition_n_keys_map.**DictBasedMachineParti**
Bases: *pacman.model.routing_info.abstract_machine_partition_n_keys_map.AbstractMachinePartitionNKeysMap*

A python dict-based implementation of the AbstractMachinePartitionNKeysMap

n_keys_for_partition (*partition*)

Parameters **partition** (pacman.model.graph.simple_outgoing_edge_partition.OutgoingEdgePartition) – The partition to set the number of keys for

set_n_keys_for_partition (*partition*, *n_keys*)
Set the number of keys required by a machine outgoing edge partition

Parameters

- **partition** (pacman.model.graph.simple_outgoing_edge_partition.OutgoingEdgePartition) – The partition to set the number of keys for
- **n_keys** (*int*) – The number of keys required by the edge

pacman.model.routing_info.partition_routing_info module

class pacman.model.routing_info.partition_routing_info.**PartitionRoutingInfo**(*keys_and_masks*, *partition*)

Bases: object

Associates a partition to its routing information (keys and masks)

Parameters

- **keys_and_masks** (iterable of *pacman.model.routing_info.BaseKeyAndMask*) – The keys allocated to the machine partition
- **partition** (pacman.model.graph.OutgoingEdgePartition) – The partition to set the number of keys for

first_key

The first key (or only one if there is only one)

first_key_and_mask

The first key and mask (or only one if there is only one)

first_mask

The first mask (or only one if there is only one)

get_keys(*n_keys=None*)

Get the ordered list of individual keys allocated to the edge

Parameters **n_keys** (*int*) – Optional limit on the number of keys to return

Returns An array of keys

Return type array-like of int

keys_and_masks

partition

pacman.model.routing_info.routing_info module

class pacman.model.routing_info.routing_info.**RoutingInfo**(*partition_info_items=None*)

Bases: object

An association of a set of edges to a non-overlapping set of keys and masks

Parameters **partition_info_items** (iterable of *pacman.model.routing_info.PartitionRoutingInfo* or none) – The partition information items to add

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If there are two partition information objects with the same partition

add_partition_info(*partition_info*)

Add a partition information item

Parameters **partition_info** (*pacman.model.routing_info.PartitionRoutingInfo*) – The partition information item to add

Return type None

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If the partition is already in the set of edges

get_first_key_for_edge(*edge*)
Get routing key for an edge

Parameters **edge** – The edge to search for

get_first_key_from_partition(*partition*)
Get the first key associated with a particular partition

Parameters **partition** (pacman.model.graph.OutgoingEdgePartition) – The partition to get the first key of

Returns The routing key or None if the partition does not exist

Return type int

Raises **None** – does not raise any known exceptions

get_first_key_from_pre_vertex(*vertex*, *partition_id*)
Get the first key for the partition starting at a vertex

Parameters

- **vertex** – The vertex which the partition starts at
- **partition_id** – The id of the partition for which to get the routing information

Returns The routing key of the partition

Return type int

get_routing_info_for_edge(*edge*)
Get routing information for an edge

Parameters **edge** – The edge to search for

get_routing_info_from_partition(*partition*)

Parameters **partition** (pacman.model.graph.OutgoingEdgePartition) – The partition to set the number of keys for

Returns the partition_routing_info for the partition

get_routing_info_from_pre_vertex(*vertex*, *partition_id*)
Get routing information for edges with a given partition_id from a pre vertex

Parameters

- **vertex** – The pre_vertex to search for
- **partition_id** – The id of the partition for which to get the routing information

Module contents

class pacman.model.routing_info.**AbstractMachinePartitionNKeysMap**
Bases: object

A map that provides the number of keys required by each partition

n_keys_for_partition(*partition*)
The number of keys required by the given partition

Parameters **partition** (pacman.model.graph.simple_outgoing_edge_partition.OutgoingEdgePartition) – The partition to set the number of keys for

Returns The number of keys required by the partition

Return type int

class pacman.model.routing_info.**BaseKeyAndMask**(*base_key*, *mask*)

Bases: object

A Key and Mask to be used for routing

Parameters

- **base_key** (*int*) – The routing key
- **mask** (*int*) – The routing mask

Raises *PacmanConfigurationException* – If key & mask != key i.e. the key is not valid for the given mask

get_keys(*key_array=None*, *offset=0*, *n_keys=None*)

Get the ordered list of keys that the combination allows

Parameters

- **key_array** (*array-like of int*) – Optional array into which the returned keys will be placed
- **offset** (*int*) – Optional offset into the array at which to start placing keys
- **n_keys** (*int*) – Optional limit on the number of keys returned. If less than this number of keys are available, only the keys available will be added

Returns A tuple of an array of keys and the number of keys added to the array

Return type (array-like of int, int)

key

The base key

Returns The base key

Return type int

key_combo

The key combined with the mask

mask

The mask

Returns The mask

Return type int

n_keys

The total number of keys that can be generated given the mask

Returns The number of keys

Return type int

class pacman.model.routing_info.**DictBasedMachinePartitionNKeysMap**

Bases: *pacman.model.routing_info.abstract_machine_partition_n_keys_map.AbstractMachinePartitionNKeysMap*

A python dict-based implementation of the AbstractMachinePartitionNKeysMap

n_keys_for_partition(*partition*)

Parameters **partition** (pacman.model.graph.simple_outgoing_edge_partition.OutgoingEdgePartition) – The partition to set the number of keys for

set_n_keys_for_partition(*partition*, *n_keys*)
Set the number of keys required by a machine outgoing edge partition

Parameters

- **partition** (pacman.model.graph.simple_outgoing_edge_partition.OutgoingEdgePartition) – The partition to set the number of keys for
- **n_keys** (*int*) – The number of keys required by the edge

class pacman.model.routing_info.**PartitionRoutingInfo**(*keys_and_masks*, *partition*)
Bases: object

Associates a partition to its routing information (keys and masks)

Parameters

- **keys_and_masks** (iterable of *pacman.model.routing_info.BaseKeyAndMask*) – The keys allocated to the machine partition
- **partition** (pacman.model.graph.OutgoingEdgePartition) – The partition to set the number of keys for

first_key
The first key (or only one if there is only one)

first_key_and_mask
The first key and mask (or only one if there is only one)

first_mask
The first mask (or only one if there is only one)

get_keys(*n_keys=None*)
Get the ordered list of individual keys allocated to the edge

Parameters **n_keys** (*int*) – Optional limit on the number of keys to return

Returns An array of keys

Return type array-like of int

keys_and_masks

partition

class pacman.model.routing_info.**RoutingInfo**(*partition_info_items=None*)
Bases: object

An association of a set of edges to a non-overlapping set of keys and masks

Parameters **partition_info_items** (iterable of *pacman.model.routing_info.PartitionRoutingInfo* or none) – The partition information items to add

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If there are two partition information objects with the same partition

add_partition_info(*partition_info*)
Add a partition information item

Parameters **partition_info** (*pacman.model.routing_info.PartitionRoutingInfo*) – The partition information item to add

Return type None

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If the partition is already in the set of edges

get_first_key_for_edge(*edge*)
Get routing key for an edge

Parameters **edge** – The edge to search for

get_first_key_from_partition(*partition*)
Get the first key associated with a particular partition

Parameters **partition** (pacman.model.graph.OutgoingEdgePartition) – The partition to get the first key of

Returns The routing key or None if the partition does not exist

Return type int

Raises **None** – does not raise any known exceptions

get_first_key_from_pre_vertex(*vertex*, *partition_id*)
Get the first key for the partition starting at a vertex

Parameters

- **vertex** – The vertex which the partition starts at
- **partition_id** – The id of the partition for which to get the routing information

Returns The routing key of the partition

Return type int

get_routing_info_for_edge(*edge*)
Get routing information for an edge

Parameters **edge** – The edge to search for

get_routing_info_from_partition(*partition*)

Parameters **partition** (pacman.model.graph.OutgoingEdgePartition) – The partition to set the number of keys for

Returns the partition_routing_info for the partition

get_routing_info_from_pre_vertex(*vertex*, *partition_id*)
Get routing information for edges with a given partition_id from a pre vertex

Parameters

- **vertex** – The pre_vertex to search for
- **partition_id** – The id of the partition for which to get the routing information

pacman.model.routing_table_by_partition package

Submodules

pacman.model.routing_table_by_partition.multicast_routing_table_by_partition module

class pacman.model.routing_table_by_partition.multicast_routing_table_by_partition.**MulticastR**
Bases: object

A set of multicast routing path objects

add_path_entry(*entry*, *router_x*, *router_y*, *partition*)
Adds a multicast routing path entry

Parameters

- **entry** – the entry to add
- **router_x** – the x coord of the router
- **router_y** – the y coord of the router
- **partition** (*pacman.model.graphs.AbstractOutgoingEdgePartition*) – the partition containing the machine edge

get_entries_for_router(*router_x*, *router_y*)

Get the set of multicast path entries assigned to this router

Parameters

- **router_x** – the x coord of the router
- **router_y** – the y coord of the router

Returns return all router_path_entries for the router.

get_entry_on_coords_for_edge(*partition*, *router_x*, *router_y*)

Get an entry from a specific coordinate

get_routers()

Get the coordinates of all stored routers

pacman.model.routing_table_by_partition.multicast_routing_table_by_partition_entry module

class pacman.model.routing_table_by_partition.multicast_routing_table_by_partition_entry.**Mult**

Bases: object

An entry in a path of a multicast route

Parameters

- **out_going_links** (*iterable of ints between 0 and 5*) – the edges this path entry goes down
- **outgoing_processors** (*iterable of ints between 0 and 17*) – the processors this path entry goes to
- **incoming_processor** (*int between 0 and 17*) – the direction this entry came from
- **incoming_link** (*int between 0 and 5*) – the direction this entry came from in link

defaultable

The defaultable status of the entry

incoming_link

The source link for this path entry

incoming_processor
The source processor

merge_entry(*other*)
Merges the another entry with this one and returns a new MulticastRoutingTableByPartitionEntry

Parameters **other** – the MulticastRoutingTableByPartitionEntry to merge into this one

Returns a merged MulticastRoutingTableByPartitionEntry

out_going_links
The destination links of the entry

out_going_processors
The destination processors of the entry

Module contents

class pacman.model.routing_table_by_partition.**MulticastRoutingTableByPartition**
Bases: object

A set of multicast routing path objects

add_path_entry(*entry*, *router_x*, *router_y*, *partition*)
Adds a multicast routing path entry

Parameters

- **entry** – the entry to add
- **router_x** – the x coord of the router
- **router_y** – the y coord of the router
- **partition** (*pacman.model.graphs.AbstractOutgoingEdgePartition*) – the partition containing the machine edge

get_entries_for_router(*router_x*, *router_y*)
Get the set of multicast path entries assigned to this router

Parameters

- **router_x** – the x coord of the router
- **router_y** – the y coord of the router

Returns return all router_path_entries for the router.

get_entry_on_coords_for_edge(*partition*, *router_x*, *router_y*)
Get an entry from a specific coordinate

get_routers()
Get the coordinates of all stored routers

class pacman.model.routing_table_by_partition.**MulticastRoutingTableByPartitionEntry**(*out_going_li*
out-
go-
ing_processo
in-
com-
ing_processo
in-
com-
ing_link=No

Bases: object

An entry in a path of a multicast route

Parameters

- **out_going_links** (*iterable of ints between 0 and 5*) – the edges this path entry goes down
- **outgoing_processors** (*iterable of ints between 0 and 17*) – the processors this path entry goes to
- **incoming_processor** (*int between 0 and 17*) – the direction this entry came from
- **incoming_link** (*int between 0 and 5*) – the direction this entry came from in link

defaultable
The defaultable status of the entry

incoming_link
The source link for this path entry

incoming_processor
The source processor

merge_entry(*other*)
Merges the another entry with this one and returns a new MulticastRoutingTableByPartitionEntry

Parameters **other** – the MulticastRoutingTableByPartitionEntry to merge into this one

Returns a merged MulticastRoutingTableByPartitionEntry

out_going_links
The destination links of the entry

out_going_processors
The destination processors of the entry

pacman.model.routing_tables package

Submodules

pacman.model.routing_tables.multicast_routing_table module

class pacman.model.routing_tables.multicast_routing_table.**MulticastRoutingTable**(*x*, *y*, *multicast_routing_entri*

Bases: object

Represents a routing table for a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip for which this is the routing table
- **y** (*int*) – The y-coordinate of the chip for which this is the routing tables
- **multicast_routing_entries** (iterable of *spinn_machine.MulticastRoutingEntry*) – An iterable of routing entries to add to the table

Raises *pacman.exceptions.PacmanAlreadyExistsException* – If any two routing entries contain the same key-mask combination

add_multicast_routing_entry(*multicast_routing_entry*)

Adds a routing entry to this table

Parameters **multicast_routing_entry** (*spinn_machine.MulticastRoutingEntry*) – The route to add

Return type None

Raises *pacman.exceptions.PacmanAlreadyExistsException* – If a routing entry with the same key-mask combination already exists

get_multicast_routing_entry_by_routing_entry_key(*routing_entry_key*, *mask*)

Get the routing entry associated with the specified key_combo-mask combination or None if the routing table does not match the key_combo

Parameters

- **routing_entry_key** (*int*) – the routing key to be searched
- **mask** (*int*) – the routing mask to be searched

Returns the routing entry associated with the routing key_combo or None if no such entry exists

Return type *spinn_machine.MulticastRoutingEntry*

multicast_routing_entries

The multicast routing entries in the table

Returns an iterable of multicast routing entries

Return type iterable of *spinn_machine.MulticastRoutingEntry*

Raises **None** – does not raise any known exceptions

number_of_defaultable_entries

The number of multi-cast routing entries that are set to be defaultable within this multicast routing table

Returns int

number_of_entries
The number of multi-cast routing entries there are in the multicast routing table

x
The x-coordinate of the chip of this table

Returns The x-coordinate

Return type int

y
The y-coordinate of the chip of this table

Returns The y-coordinate

Return type int

pacman.model.routing_tables.multicast_routing_tables module

class pacman.model.routing_tables.multicast_routing_tables.**MulticastRoutingTables**(*routing_tables=*
Bases: object

Represents the multicast routing tables for a number of chips

Parameters **routing_tables** (iterable of *pacman.model.routing_tables.MulticastRoutingTable*) – The routing tables to add

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If any two routing tables are for the same chip

add_routing_table(*routing_table*)
Add a routing table

Parameters **routing_table** (*pacman.model.routing_tables.MulticastRoutingTable*) – a routing table to add

Return type None

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If a routing table already exists for the chip

get_routing_table_for_chip(*x*, *y*)
Get a routing table for a particular chip

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip

Returns The routing table, or None if no such table exists

Return type *pacman.model.routing_tables.MulticastRoutingTable* or None

Raises **None** – No known exceptions are raised

routing_tables
The routing tables stored within

Returns an iterable of routing tables

Return type iterable of *pacman.model.routing_tables.MulticastRoutingTable*

Raises **None** – does not raise any known exceptions

Module contents

class pacman.model.routing_tables.**MulticastRoutingTable**(*x*, *y*, *multicast_routing_entries=None*)

Bases: object

Represents a routing table for a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip for which this is the routing table
- **y** (*int*) – The y-coordinate of the chip for which this is the routing tables
- **multicast_routing_entries** (iterable of *spinn_machine.MulticastRoutingEntry*) – An iterable of routing entries to add to the table

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If any two routing entries contain the same key-mask combination

add_multicast_routing_entry(*multicast_routing_entry*)

Adds a routing entry to this table

Parameters **multicast_routing_entry** (*spinn_machine.MulticastRoutingEntry*) – The route to add

Return type None

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If a routing entry with the same key-mask combination already exists

get_multicast_routing_entry_by_routing_entry_key(*routing_entry_key*, *mask*)

Get the routing entry associated with the specified key_combo-mask combination or None if the routing table does not match the key_combo

Parameters

- **routing_entry_key** (*int*) – the routing key to be searched
- **mask** (*int*) – the routing mask to be searched

Returns the routing entry associated with the routing key_combo or None if no such entry exists

Return type *spinn_machine.MulticastRoutingEntry*

multicast_routing_entries

The multicast routing entries in the table

Returns an iterable of multicast routing entries

Return type iterable of *spinn_machine.MulticastRoutingEntry*

Raises **None** – does not raise any known exceptions

number_of_defaultable_entries

The number of multi-cast routing entries that are set to be defaultable within this multicast routing table

Returns int

number_of_entries

The number of multi-cast routing entries there are in the multicast routing table

x

The x-coordinate of the chip of this table

Returns The x-coordinate

Return type int

y

The y-coordinate of the chip of this table

Returns The y-coordinate

Return type int

class `pacman.model.routing_tables.`**`MulticastRoutingTables`**(*routing_tables=None*)

Bases: `object`

Represents the multicast routing tables for a number of chips

Parameters **routing_tables** (iterable of *pacman.model.routing_tables.MulticastRoutingTable*) – The routing tables to add

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If any two routing tables are for the same chip

add_routing_table(*routing_table*)

Add a routing table

Parameters **routing_table** (*pacman.model.routing_tables.MulticastRoutingTable*) – a routing table to add

Return type None

Raises ***pacman.exceptions.PacmanAlreadyExistsException*** – If a routing table already exists for the chip

get_routing_table_for_chip(*x*, *y*)

Get a routing table for a particular chip

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip

Returns The routing table, or None if no such table exists

Return type *pacman.model.routing_tables.MulticastRoutingTable* or None

Raises **None** – No known exceptions are raised

routing_tables

The routing tables stored within

Returns an iterable of routing tables

Return type iterable of *pacman.model.routing_tables.MulticastRoutingTable*

Raises **None** – does not raise any known exceptions

pacman.model.tags package

Submodules

pacman.model.tags.tags module

class pacman.model.tags.tags.**Tags**

Bases: object

Represents assigned IP Tag and Reverse IP Tags

add_ip_tag(*ip_tag*, *vertex*)

Add an IP tag

Parameters

- **ip_tag** (*spinn_machine.tags.IPTag*) – The tag to add
- **vertex** (pacman.model.graph.machine.MachineVertex) – The machine vertex by which the tag is to be used

Raises *PacmanInvalidParameterException* –

- If the combination of (board-address, tag) has already been assigned to an IP tag with different properties
- If the combination of (board-address, tag) has already been assigned to a reverse IP tag

add_reverse_ip_tag(*reverse_ip_tag*, *vertex*)

Add a reverse iptag

Parameters

- **reverse_ip_tag** (*spinn_machine.tags.ReverseIPTag*) – The tag to add
- **vertex** (pacman.model.graph.machine.MachineVertex) – The vertex by which the tag is to be used

Raises *PacmanInvalidParameterException* –

- If the combination of (board-address, tag) has already been assigned to an IP tag or Reverse IP tag
- If the port of the tag has already been assigned on the given board-address

get_ip_tags_for_vertex(*vertex*)

Get the IP Tags assigned to a given machine vertex

Parameters **vertex** (pacman.model.graph.machine.MachineVertex) – The vertex to get the tags for

Returns An iterable of IPTag or None if the vertex has no tags

Return type iterable of *spinn_machine.tags.IPTag* or None

get_reverse_ip_tags_for_vertex(*vertex*)

Get the Reverse IP Tags assigned to a given machine vertex

Parameters **vertex** (pacman.model.graph.AbstractVertex) – The vertex to get the tags for

Returns An iterable of ReverseIPTag or None if the vertex has no tags

Return type iterable of *spinn_machine.tags.ReverseIPTag* or None

ip_tags
The IPTags assigned

Returns iterable of IPTag

Return type iterable of *spinn_machine.tags.IPTag*

reverse_ip_tags
The ReverseIPTags assigned

Returns iterable of ReverseIPTag

Return type iterable of *spinn_machine.tags.ReverseIPTag*

Module contents

class pacman.model.tags.**Tags**
Bases: object

Represents assigned IP Tag and Reverse IP Tags

add_ip_tag(*ip_tag*, *vertex*)
Add an IP tag

Parameters

- **ip_tag** (*spinn_machine.tags.IPTag*) – The tag to add
- **vertex** (pacman.model.graph.machine.MachineVertex) – The machine vertex by which the tag is to be used

Raises ***PacmanInvalidParameterException*** –

- If the combination of (board-address, tag) has already been assigned to an IP tag with different properties
- If the combination of (board-address, tag) has already been assigned to a reverse IP tag

add_reverse_ip_tag(*reverse_ip_tag*, *vertex*)
Add a reverse iptag

Parameters

- **reverse_ip_tag** (*spinn_machine.tags.ReverseIPTag*) – The tag to add
- **vertex** (pacman.model.graph.machine.MachineVertex) – The vertex by which the tag is to be used

Raises ***PacmanInvalidParameterException*** –

- If the combination of (board-address, tag) has already been assigned to an IP tag or Reverse IP tag
- If the port of the tag has already been assigned on the given board-address

get_ip_tags_for_vertex(*vertex*)
Get the IP Tags assigned to a given machine vertex

Parameters **vertex** (pacman.model.graph.machine.MachineVertex) – The vertex to get the tags for

Returns An iterable of IPTag or None if the vertex has no tags

Return type iterable of *spinn_machine.tags.IPTag* or None

get_reverse_ip_tags_for_vertex(*vertex*)
Get the Reverse IP Tags assigned to a given machine vertex

Parameters **vertex** (pacman.model.graph.AbstractVertex) – The vertex to get the tags for

Returns An iterable of ReverseIPTag or None if the vertex has no tags

Return type iterable of *spinn_machine.tags.ReverseIPTag* or None

ip_tags
The IPTags assigned

Returns iterable of IPTag

Return type iterable of *spinn_machine.tags.IPTag*

reverse_ip_tags
The ReverseIPTags assigned

Returns iterable of ReverseIPTag

Return type iterable of *spinn_machine.tags.ReverseIPTag*

Module contents

pacman.operations package

Subpackages

pacman.operations.algorithm_reports package

Submodules

pacman.operations.algorithm_reports.network_specification module

class pacman.operations.algorithm_reports.network_specification.**NetworkSpecification**
Bases: object

Generate report on the user's network specification.

pacman.operations.algorithm_reports.reports module

pacman.operations.algorithm_reports.reports.**generate_comparison_router_report**(*report_folder*, *routing_tables*, *compressed_routing_ta*

Make a report on comparison of the compressed and uncompressed routing tables

Parameters

- **report_folder** – the folder to store the resulting report
- **routing_tables** – the original routing tables
- **compressed_routing_tables** – the compressed routing tables

Return type None

`pacman.operations.algorithm_reports.reports.`**`partitioner_report`**(*report_folder*, *hostname*, *graph*, *graph_mapper*)

Generate report on the placement of vertices onto cores.

`pacman.operations.algorithm_reports.reports.`**`placement_report_with_application_graph_by_core`**

Generate report on the placement of vertices onto cores by core.

Parameters

- **`report_folder`** – the folder to which the reports are being written
- **`hostname`** – the machine's hostname to which the placer worked on
- **`graph_mapper`** – the mapping between application and machine graphs
- **`machine`** – the spinnaker machine object
- **`placements`** – the placements objects built by the placer.

`pacman.operations.algorithm_reports.reports.`**`placement_report_with_application_graph_by_vert`**

Generate report on the placement of vertices onto cores by vertex.

Parameters

- **`report_folder`** – the folder to which the reports are being written
- **`hostname`** – the machine's hostname to which the placer worked on
- **`graph`** – the graph to which placements were built
- **`graph_mapper`** – the mapping between graphs
- **`placements`** – the placements objects built by the placer.

`pacman.operations.algorithm_reports.reports.`**`placement_report_without_application_graph_by_c`**

Generate report on the placement of vertices onto cores by core.

Parameters

- **`report_folder`** – the folder to which the reports are being written

- **hostname** – the machine's hostname to which the placer worked on
- **machine** – the spinnaker machine object
- **placements** – the placements objects built by the placer.

pacman.operations.algorithm_reports.reports.**placement_report_without_application_graph_by_v**

Generate report on the placement of vertices onto cores by vertex.

Parameters

- **report_folder** – the folder to which the reports are being written
- **hostname** – the machine's hostname to which the placer worked on
- **placements** – the placements objects built by the placer.
- **machine_graph** – the machine graph generated by the end user

pacman.operations.algorithm_reports.reports.**placer_reports_with_application_graph**(*report_folde host-name, graph, graph_mapp place-ments, ma-chine*)

Reports that can be produced from placement given a application graph's existence

Parameters

- **report_folder** – the folder to which the reports are being written
- **hostname** – the machine's hostname to which the placer worked on
- **graph** – the application graph to which placements were built
- **graph_mapper** – the mapping between application and machine graphs
- **placements** – the placements objects built by the placer.
- **machine** – the python machine object

Return type None

pacman.operations.algorithm_reports.reports.**placer_reports_without_application_graph**(*report_f host-name, ma-chine_g place-ments, ma-chine*)

Parameters

- **report_folder** – the folder to which the reports are being written

- **hostname** – the machine's hostname to which the placer worked on
- **placements** – the placements objects built by the placer.
- **machine** – the python machine object
- **machine_graph** – the machine graph to which the reports are to operate on

Return type None

pacman.operations.algorithm_reports.reports.**router_report_from_compressed_router_tables**(*report_folder*, *routing_tables*)

Parameters

- **report_folder** –
- **routing_tables** –

Return type None

pacman.operations.algorithm_reports.reports.**router_report_from_paths**(*report_folder*, *routing_tables*, *routing_infos*, *hostname*, *machine_graph*, *placements*, *machine*)

Generates a text file of routing paths

Parameters

- **routing_tables** –
- **report_folder** –
- **hostname** –
- **routing_infos** –
- **machine_graph** –
- **placements** –
- **machine** –

Return type None

pacman.operations.algorithm_reports.reports.**router_report_from_router_tables**(*report_folder*, *routing_tables*)

Parameters

- **report_folder** –
- **routing_tables** –

Return type None

pacman.operations.algorithm_reports.reports.**routing_info_report**(*report_folder*, *machine_graph*, *routing_infos*)

Generates a report which says which keys is being allocated to each vertex

Parameters

- **report_folder** – the report folder to store this value
- **machine_graph** –
- **routing_infos** –

pacman.operations.algorithm_reports.reports.**sdram_usage_report_per_chip**(*report_folder*, *hostname*, *placements*, *machine*)

Reports the SDRAM used per chip

Parameters

- **report_folder** – the folder to which the reports are being written
- **hostname** – the machine's hostname to which the placer worked on
- **placements** – the placements objects built by the placer.
- **machine** – the python machine object

Return type None

pacman.operations.algorithm_reports.reports.**tag_allocator_report**(*report_folder*, *tag_infos*)

Reports the tags that are being used by the tool chain for this simulation

Parameters

- **report_folder** – the folder to which the reports are being written
- **tag_infos** – the tags container generated by the tools.

Return type None

Module contents

pacman.operations.chip_id_allocator_algorithms package

Submodules

pacman.operations.chip_id_allocator_algorithms.malloc_based_chip_id_allocator module

class pacman.operations.chip_id_allocator_algorithms.malloc_based_chip_id_allocator.**MallocBas**

Bases: *pacman.utilities.algorithm_utilities.element_allocator_algorithm.ElementAllocatorAlgorithm*

A Chip id Allocation Allocator algorithm that keeps track of chip ids and attempts to allocate them as requested

Module contents

class pacman.operations.chip_id_allocator_algorithms.**MallocBasedChipIdAllocator**

Bases: *pacman.utilities.algorithm_utilities.element_allocator_algorithm.ElementAllocatorAlgorithm*

A Chip id Allocation Allocator algorithm that keeps track of chip ids and attempts to allocate them as requested

pacman.operations.fixed_route_router package

Submodules

pacman.operations.fixed_route_router.fixed_route_router module

class pacman.operations.fixed_route_router.fixed_route_router.**FixedRouteRouter**

Bases: object

fixed router that makes a mirror path on every board based off the below diagram. It assumed there's a core on the ethernet connected chip that is of the destination class.

[] [] [] [] / / / /

[] [] [] [] [] / / /

[] [] [] [] [] [] / / / / / /

[] [] [] [] [] [] [] / / / / / / /

[] [] [] [] [] [] [] [] | / / / / / / / [] [] [] [] [] [] [] [] | / / / / / / [] []-[] [] [] [] | / / / / []-[]-[]-[]-[]

joins = {(0, 1): [5], (2, 0): [3], (5, 6): [5], (6, 6): [5], (1, 0): [3], (0, 3): [5], (3, 0): [3], (0, 2): [5], (2, 1): [3], (4, 0): [3]}

router_path_chips = {0: [(1, 0), (2, 0), (3, 0), (4, 0), (3, 1), (4, 1), (5, 1), (4, 2), (5, 2), (6, 2), (5, 3), (6, 3), (7, 3), (6, 4), (7,

Module contents

pacman.operations.multi_cast_router_check_functionality package

Submodules

pacman.operations.multi_cast_router_check_functionality.valid_routes_checker module

Collection of functions which together validate routes.

class pacman.operations.multi_cast_router_check_functionality.valid_routes_checker.**PlacementT**

Bases: tuple

Create new instance of PlacementTuple(x, y, p)

p

Alias for field number 2

x

Alias for field number 0

y
Alias for field number 1

pacman.operations.multi_cast_router_check_functionality.valid_routes_checker.**validate_route**

Go though the placements given and check that the routing entries within the routing tables support reach the correction destinations as well as not producing any cycles.

Parameters

- **machine_graph** – the graph
- **placements** – the placements container
- **routing_infos** – the routing info container
- **routing_tables** – the routing tables generated by the routing algorithm
- **graph_mapper** – the mapping between graphs or none if only using a machine graph
- **machine** (*spinnMachine.machine.Machine object*) – the python machine object

Return type None

Raises *PacmanRoutingException* – when either no routing table entry is found by the search on a given router, or a cycle is detected

Module contents

pacman.operations.partition_algorithms package

Submodules

pacman.operations.partition_algorithms.basic_partitioner module

class pacman.operations.partition_algorithms.basic_partitioner.**BasicPartitioner**
Bases: object

An basic algorithm that can partition an application graph based on the number of atoms in the vertices.

pacman.operations.partition_algorithms.partition_and_place_partitioner module

class pacman.operations.partition_algorithms.partition_and_place_partitioner.**PartitionAndPlac**
Bases: object

A partitioner that tries to ensure that SDRAM is not overloaded by keeping track of the SDRAM usage on the various chips

Module contents

class pacman.operations.partition_algorithms.**BasicPartitioner**
Bases: object

An basic algorithm that can partition an application graph based on the number of atoms in the vertices.

class pacman.operations.partition_algorithms.**PartitionAndPlacePartitioner**
Bases: object

A partitioner that tries to ensure that SDRAM is not overloaded by keeping track of the SDRAM usage on the various chips

pacman.operations.placer_algorithms package

Submodules

pacman.operations.placer_algorithms.basic_placer module

class pacman.operations.placer_algorithms.basic_placer.**BasicPlacer**
Bases: object

A basic placement algorithm that can place a machine graph onto a machine using the chips as they appear in the machine

pacman.operations.placer_algorithms.connective_based_placer module

class pacman.operations.placer_algorithms.connective_based_placer.**ConnectiveBasedPlacer**
Bases: *pacman.operations.placer_algorithms.radial_placer.RadialPlacer*

A radial algorithm that can place a machine graph onto a machine using a circle out behaviour from a Ethernet at a given point and which will place things that are most connected closest to each other

pacman.operations.placer_algorithms.one_to_one_placer module

class pacman.operations.placer_algorithms.one_to_one_placer.**OneToOnePlacer**
Bases: *pacman.operations.placer_algorithms.radial_placer.RadialPlacer*

Placer that puts vertices which are directly connected to only its destination on the same chip

pacman.operations.placer_algorithms.radial_placer module

class pacman.operations.placer_algorithms.radial_placer.**RadialPlacer**
Bases: object

A placement algorithm that can place a machine graph onto a machine choosing chips radiating in a circle from the boot chip

Module contents

class pacman.operations.placer_algorithms.**RadialPlacer**
Bases: object

A placement algorithm that can place a machine graph onto a machine choosing chips radiating in a circle from the boot chip

class pacman.operations.placer_algorithms.**BasicPlacer**
Bases: object

A basic placement algorithm that can place a machine graph onto a machine using the chips as they appear in the machine

class pacman.operations.placer_algorithms.**ConnectiveBasedPlacer**
Bases: *pacman.operations.placer_algorithms.radial_placer.RadialPlacer*

A radial algorithm that can place a machine graph onto a machine using a circle out behaviour from a Ethernet at a given point and which will place things that are most connected closest to each other

pacman.operations.rig_algorithms package

Submodules

pacman.operations.rig_algorithms.rig_mc_route module

class pacman.operations.rig_algorithms.rig_mc_route.**RigMCRoute**
Bases: object

Performs routing using rig algorithm

pacman.operations.rig_algorithms.rig_place module

class pacman.operations.rig_algorithms.rig_place.**RigPlace**
Bases: object

Performs placement and routing using rig algorithms; both are done to save conversion time

pacman.operations.rig_algorithms.rig_place_and_route module

class pacman.operations.rig_algorithms.rig_place_and_route.**RigPlaceAndRoute**
Bases: object

Performs placement and routing using rig algorithms; both are done to save conversion time

pacman.operations.rig_algorithms.rig_route module

class pacman.operations.rig_algorithms.rig_route.**RigRoute**
Bases: object

Performs routing using rig algorithm

Module contents

pacman.operations.rigged_algorithms package

Submodules

pacman.operations.rigged_algorithms.hilbert_placer module

class `pacman.operations.rigged_algorithms.hilbert_placer.`**`HilbertPlacer`**

Bases: `object`

A simple placing algorithm using the Hilbert space-filling curve, translated from RIG.

pacman.operations.rigged_algorithms.hilbert_state module

class `pacman.operations.rigged_algorithms.hilbert_state.`**`HilbertState`**(*xpos=0*, *ypos=0*, *xchange=1*, *ychange=0*)

Bases: `object`

A mutable self object for the hilbert placer algorithm.

Constructor :param xpos: the x coordinate on the generated curve :param ypos: the y coordinate on the generated curve :param xchange: the change in x coordinate on the generated curve :param ychange: the change in y coordinate on the generated curve :type xpos: int :type ypos: int :type xchange: int :type ychange: int

`move_forward`()

method to move forward in the generation of a hilbert curve :return: the x and y coordinates on the generated curve

`turn_left`(*angle*)

method to turn left in the generation of a hilbert curve :param angle: determines the direction in which the curve turns :type angle: int :return: the x and y coordinates on the generated curve

`turn_right`(*angle*)

method to turn right in the generation of a hilbert curve :param angle: determines the direction in which the curve turns :type angle: int :return: the x and y coordinates on the generated curve

`x_pos`

`y_pos`

pacman.operations.rigged_algorithms.isomorph_check module

class `pacman.operations.rigged_algorithms.isomorph_check.`**`IsomorphicChecker`**

Bases: `object`

A short algorithm to check if there is an isomorphism of the placement of vertices by two separate placement algorithms. One of the algorithms must output to memory placements_copy in its method and <param_type>MemoryPlacements2</param_type> in algorithms_metadata.xml.

`check`(*placements*, *placements_copy*)

Checks if the placements on each processor are the same for two placement algorithms.

Parameters **`placements`** – Placements of vertices on the machine

:type *pacman.model.placements.placements.Placements* :param placements_copy: memory copy of placements of vertices on the machine :type *pacman.model.placements.placements.Placements* :return True if the placements are the same :rtype bool

pacman.operations.rigged_algorithms.random_placer module

class pacman.operations.rigged_algorithms.random_placer.**RandomPlacer**

Bases: object

THRESHOLD = 3

Module contents

pacman.operations.router_algorithms package

Submodules

pacman.operations.router_algorithms.basic_dijkstra_routing module

class pacman.operations.router_algorithms.basic_dijkstra_routing.**BasicDijkstraRouting**

Bases: object

An routing algorithm that can find routes for edges between vertices in a machine graph that have been placed on a machine by the use of a Dijkstra shortest path algorithm

BW_PER_ROUTE_ENTRY = 0.01

MAX_BW = 250

Module contents

class pacman.operations.router_algorithms.**BasicDijkstraRouting**

Bases: object

An routing algorithm that can find routes for edges between vertices in a machine graph that have been placed on a machine by the use of a Dijkstra shortest path algorithm

BW_PER_ROUTE_ENTRY = 0.01

MAX_BW = 250

pacman.operations.router_compressors package

Subpackages

pacman.operations.router_compressors.mundys_router_compressor package

Submodules

pacman.operations.router_compressors.mundys_router_compressor.routing_table_condenser module

class `pacman.operations.router_compressors.mundys_router_compressor.routing_table_condenser.M`

Bases: `object`

compressor from rig which has been tied into the main tool chain stack.

class **`KeyMask`** (*key*, *mask*)

Bases: `tuple`

Create new instance of KeyMask(key, mask)

`key`

Alias for field number 0

`mask`

Alias for field number 1

class **`RoutingEntry`** (*key*, *mask*, *route*, *defaultable*)

Bases: `tuple`

Create new instance of RoutingEntry(key, mask, route, defaultable)

`defaultable`

Alias for field number 3

`key`

Alias for field number 0

`mask`

Alias for field number 1

`route`

Alias for field number 2

`max_supported_length` = **1023**

Module contents

Submodules

pacman.operations.router_compressors.basic_route_merger module

class `pacman.operations.router_compressors.basic_route_merger.`**`BasicRouteMerger`**

Bases: `object`

functionality to merge routing tables entries via different masks and a exploration process

pacman.operations.router_compressors.malloc_based_route_merger module

class pacman.operations.router_compressors.malloc_based_route_merger.**MallocBasedRouteMerger**
Bases: object

a routing table entry merging function, that merges based off a malloc memory style.

Module contents

pacman.operations.routing_info_allocator_algorithms package

Subpackages

pacman.operations.routing_info_allocator_algorithms.field_based_routing_allocator package

Submodules

pacman.operations.routing_info_allocator_algorithms.field_based_routing_allocator.vertex_based_routing_info_a module

class pacman.operations.routing_info_allocator_algorithms.field_based_routing_allocator.verte
Bases: object

allocator of routing keys based off the vertex requirements

static add_field_constraints (*partition*, *graph_mapper*, *graph*, *n_keys_map*)
Search though the graph adding field constraints for the key allocator

Module contents

pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator package

Submodules

pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.compressible_malloc_base module

class pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.comp
Bases: *pacman.utilities.algorithm_utilities.element_allocator_algorithm.ElementAllocatorAlgorithm*

A Routing Info Allocation Allocator algorithm that keeps track of free keys and attempts to allocate them as requested, but that also looks at routing tables in an attempt to make things more compressible

pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.key_field_generator module

class pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.key_

Bases: object

functionality to handle fields in a routing key.

is_next_key

next()

next_key

pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.malloc_based_routing_info module

class pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.mall

Bases: *pacman.utilities.algorithm_utilities.element_allocator_algorithm.ElementAllocatorAlgorithm*

A Routing Info Allocation Allocator algorithm that keeps track of free keys and attempts to allocate them as requested

Module contents

Submodules

pacman.operations.routing_info_allocator_algorithms.basic_routing_info_allocator module

class pacman.operations.routing_info_allocator_algorithms.basic_routing_info_allocator.**BasicR**

Bases: object

An basic algorithm that can produce routing keys and masks for edges in a graph based on the x,y,p of the placement of the preceding vertex. Note that no constraints are supported, and that the number of keys required by each edge must be 2048 or less, and that all edges coming out of a vertex will be given the same key/mask assignment.

pacman.operations.routing_info_allocator_algorithms.destination_based_key_allocator module

class pacman.operations.routing_info_allocator_algorithms.destination_based_key_allocator.**Des**

Bases: object

A routing key allocator that operates for people who wish to have a separate key for each destination (making a mc into a point-to-point cast.

MASK = 4294965248

MAX_KEYS_SUPPORTED = 2048

Module contents

class pacman.operations.routing_info_allocator_algorithms.**BasicRoutingInfoAllocator**
Bases: object

An basic algorithm that can produce routing keys and masks for edges in a graph based on the x,y,p of the placement of the preceding vertex. Note that no constraints are supported, and that the number of keys required by each edge must be 2048 or less, and that all edges coming out of a vertex will be given the same key/mask assignment.

class pacman.operations.routing_info_allocator_algorithms.**MallocBasedRoutingInfoAllocator**
Bases: *pacman.utilities.algorithm_utilities.element_allocator_algorithm.ElementAllocatorAlgorithm*

A Routing Info Allocation Allocator algorithm that keeps track of free keys and attempts to allocate them as requested

pacman.operations.routing_table_generators package

Submodules

pacman.operations.routing_table_generators.basic_routing_table_generator module

class pacman.operations.routing_table_generators.basic_routing_table_generator.**BasicRoutingTa**
Bases: object

An basic algorithm that can produce routing tables

Module contents

pacman.operations.tag_allocator_algorithms package

Submodules

pacman.operations.tag_allocator_algorithms.basic_tag_allocator module

class pacman.operations.tag_allocator_algorithms.basic_tag_allocator.**BasicTagAllocator**
Bases: object

Basic tag allocator that goes though the boards available and applies the ip tags and reverse ip tags as needed.

Module contents

class pacman.operations.tag_allocator_algorithms.**BasicTagAllocator**
Bases: object

Basic tag allocator that goes though the boards available and applies the ip tags and reverse ip tags as needed.

Module contents

pacman.utilities package

Subpackages

pacman.utilities.algorithm_utilities package

Submodules

pacman.utilities.algorithm_utilities.element_allocator_algorithm module

class pacman.utilities.algorithm_utilities.element_allocator_algorithm.**ElementAllocatorAlgori**

Bases: object

Abstract element allocator algorithm which allocates elements from a pool of a given size

pacman.utilities.algorithm_utilities.field_based_system_utilities module

class pacman.utilities.algorithm_utilities.field_based_system_utilities.**TYPES_OF_FIELDS**

Bases: enum.Enum

FIXED_FIELD = 2

FIXED_KEY = 1

FIXED_MASK = 0

pacman.utilities.algorithm_utilities.field_based_system_utilities.**convert_mask_into_fields**

Parameters **entity** –

pacman.utilities.algorithm_utilities.field_based_system_utilities.**deduce_types**(*graph*)

deducing the number of applications required for this key space

Parameters **graph** –

pacman.utilities.algorithm_utilities.field_based_system_utilities.**handle_flexi_field**(*constrai*
seen_fie
*known_*

Parameters

- **constraint** –
- **seen_fields** –
- **known_fields** –

Return type None:

pacman.utilities.algorithm_utilities.machine_algorithm_utilities module

pacman.utilities.algorithm_utilities.machine_algorithm_utilities.**create_virtual_chip**(*machine*
link_dat
vir-
tual_chi
vir-
tual_chi

pacman.utilities.algorithm_utilities.partition_algorithm_utilities module

A collection of methods which support partitioning algorithms.

pacman.utilities.algorithm_utilities.partition_algorithm_utilities.**generate_machine_edges**(*m*
g
a
p
a
t

Generate the machine edges for the vertices in the graph

Parameters

- **machine_graph** (pacman.model.graph.machine.MachineGraph) – the machine graph to add edges to
- **graph_mapper** (pacman.model.GraphMapper) – the mapper graphs
- **application_graph** (pacman.model.graph.application.ApplicationGraph) – the application graph to work with

pacman.utilities.algorithm_utilities.partition_algorithm_utilities.**get_remaining_constraint**

Gets the rest of the constraints from a vertex after removing partitioning constraints

pacman.utilities.algorithm_utilities.partition_algorithm_utilities.**get_same_size_vertex_gro**

Get a dictionary of vertex to vertex that must be partitioned the same size

pacman.utilities.algorithm_utilities.placer_algorithm_utilities module

pacman.utilities.algorithm_utilities.placer_algorithm_utilities.**get_same_chip_vertex_groups**

Get a dictionary of vertex to vertex that must be placed on the same chip

pacman.utilities.algorithm_utilities.placer_algorithm_utilities.**sort_vertices_by_known_cons**

Sort vertices to be placed by constraint so that those with more restrictive constraints come first.

pacman.utilities.algorithm_utilities.routing_info_allocator_utilities module

pacman.utilities.algorithm_utilities.routing_info_allocator_utilities.**check_types_of_edge_c**

Go through the graph for operations and checks that the constraints are compatible.

Parameters **machine_graph** – the graph to search through

Return type None:

pacman.utilities.algorithm_utilities.routing_info_allocator_utilities.**get_edge_groups**(*machi traf- fic_typ*

Utility method to get groups of edges using any pacman.model.constraints.key_allocator_constraints.KeyAllocatorSameKeyConstraint constraints. Note that no checking is done here about conflicts related to other constraints.

Parameters

- **machine_graph** – the machine graph
- **traffic_type** – the traffic type to group

pacman.utilities.algorithm_utilities.routing_info_allocator_utilities.**get_fixed_mask**(*same_ke*

Get a fixed mask from a group of edges if a *pacman.model.constraints.key_allocator_constraints.FixedMaskConstraint* constraint exists in any of the edges in the group.

Parameters **same_key_group** (iterable of pacman.model.graph.machine.MachineEdge) – Set of edges that are to be assigned the same keys and masks

Returns The fixed mask if found, or None

Raises *PacmanValueError* – If two edges conflict in their requirements

Module contents

class pacman.utilities.algorithm_utilities.**ElementAllocatorAlgorithm**(*size_begin, size_end*)

Bases: object

Abstract element allocator algorithm which allocates elements from a pool of a given size

pacman.utilities.file_format_converters package

Submodules

pacman.utilities.file_format_converters.convert_to_file_core_allocations module

class pacman.utilities.file_format_converters.convert_to_file_core_allocations.**ConvertToFileC**

Bases: object

Converts placements to core allocations

pacman.utilities.file_format_converters.convert_to_file_machine module

class pacman.utilities.file_format_converters.convert_to_file_machine.**ConvertToFileMachine**

Bases: object

Converter from memory machine to file machine

pacman.utilities.file_format_converters.convert_to_file_machine_graph module

class pacman.utilities.file_format_converters.convert_to_file_machine_graph.**ConvertToFileMach**

Bases: object

Converts a memory based graph into a file based graph

pacman.utilities.file_format_converters.convert_to_file_machine_graph_pure_multicast module

class pacman.utilities.file_format_converters.convert_to_file_machine_graph_pure_multicast.**Co**
Bases: object

Converts a memory based graph into a file based graph

pacman.utilities.file_format_converters.convert_to_file_placement module

class pacman.utilities.file_format_converters.convert_to_file_placement.**ConvertToFilePlacemen**
Bases: object

Converts memory placements to file placements

pacman.utilities.file_format_converters.convert_to_memory_multi_cast_routes module

class pacman.utilities.file_format_converters.convert_to_memory_multi_cast_routes.**ConvertToMe**
Bases: object

Converts between file routing paths and the pacman representation of the routes

route_translation = {'SOUTH': (False, 5), 'NORTH': (False, 2), 'CORE_15': (True, 15), 'CORE_14': (True, 14), 'N

pacman.utilities.file_format_converters.convert_to_memory_placements module

class pacman.utilities.file_format_converters.convert_to_memory_placements.**ConvertToMemoryPla**
Bases: object

Takes the file-based placements, machine, machine graph and constraints and builds a memory placements object

pacman.utilities.file_format_converters.create_file_constraints module

class pacman.utilities.file_format_converters.create_file_constraints.**CreateConstraintsToFile**
Bases: object

Creates constraints file from the machine and machine graph

Module contents

pacman.utilities.file_format_schemas package

Module contents

This code has no python scripts.

pacman.utilities.utility_objs package

Submodules

pacman.utilities.utility_objs.field module

class pacman.utilities.utility_objs.field.**Field**(*lo*, *hi*, *value*, *tag=<SUPPORTED_TAGS.ROUTING: 1>*, *name=None*)

Bases: object

Field object used in a field constraint for key allocation

hi

lo

name

tag

value

pacman.utilities.utility_objs.flexi_field module

class pacman.utilities.utility_objs.flexi_field.**FlexiField**(*flexi_field_name*, *value=None*, *instance_n_keys=None*, *tag=None*, *nested_level=0*)

Bases: object

field who's location is not fixed in key allocation

instance_n_keys

name

The name for this Flexible field

tag

value

class pacman.utilities.utility_objs.flexi_field.**SUPPORTED_TAGS**

Bases: enum.Enum

APPLICATION = 0

ROUTING = 1

pacman.utilities.utility_objs.resource_tracker module

class pacman.utilities.utility_objs.resource_tracker.**ResourceTracker**(*machine*, *chips=None*, *preallocated_resources=None*)

Bases: object

Tracks the usage of resources of a machine

Parameters

- **machine** (*spinn_machine.Machine*) – The machine to track the usage of
- **chips** (*iterable of (int, int) tuples of coordinates of chips*) – If specified, this list of chips will be used instead of the list from the machine. Note that the order will be maintained, so this can be used either to reduce the set of chips used, or to re-order the chips. Note also that on deallocation, the order is no longer guaranteed.

allocate_constrained_group_resources(*resource_and_constraint_list*, *chips=None*)

Allocates a group of cores on the same chip for these resources

Parameters

- **resource_and_constraint_list** – A list of tuples of (resources, list of constraints) to allocate
- **chips** – a list of chips that can be used

Returns list of The x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type iterable of (int, int, int, list((int, int)), list((int, int)))

allocate_constrained_resources(*resources*, *constraints*, *chips=None*)

Attempts to use the given resources of the machine, constrained by the given placement constraints.

Parameters

- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to be allocated
- **constraints** (list of *pacman.model.constraints.AbstractConstraint*) – the constraints to consider
- **chips** (*iterable of (int, int)*) – The optional list of (x, y) tuples of chip coordinates of chips that can be used. Note that any chips passed in previously will be ignored

Returns The x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type (int, int, int, list((int, int)), list((int, int)))

Raises ***PacmanValueError*** – If the constraints cannot be met given the current allocation of resources

allocate_group_resources(*group_resources*, *chips=None*, *processor_ids=None*, *board_address=None*, *group_ip_tags=None*, *group_reverse_ip_tags=None*)

Attempts to use the given group of resources on a single chip of the machine. Can be given specific place to use the resources, or else it will allocate them on the first place that the resources of the group fit together.

Parameters

- **group_resources** (list of *pacman.model.resources.ResourceContainer*) – The resources to be allocated
- **chips** (*iterable of (int, int)*) – An iterable of (x, y) tuples of chips that are to be used
- **processor_ids** (*list of (int or None)*) – The specific processor to use on any chip for each resource of the group
- **board_address** (*str*) – the board address to allocate resources of a chip

- **group_ip_tags** (list of lists of `pacman.model.resources.IptagResource`) – list of lists of IP tag resources
- **group_reverse_ip_tags** (list of lists of `pacman.model.resources.ReverseIptagResource`) – list of lists of reverse IP tag resources

Returns An iterable of tuples of the x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type iterable of (int, int, int, list((int, int)), list((int, int)))

allocate_resources(*resources*, *chips=None*, *processor_id=None*, *board_address=None*, *ip_tags=None*, *reverse_ip_tags=None*)

Attempts to use the given resources of the machine. Can be given specific place to use the resources, or else it will allocate them on the first place that the resources fit.

Parameters

- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to be allocated
- **chips** (*iterable of (int, int)*) – An iterable of (x, y) tuples of chips that are to be used
- **processor_id** (*int*) – The specific processor to use on any chip.
- **board_address** (*str*) – the board address to allocate resources of a chip
- **ip_tags** (iterable of `pacman.model.resources.IptagResource`) – iterable of IP tag resources
- **reverse_ip_tags** (iterable of *pacman.model.resources.ReverseIPtagResource*) – iterable of reverse IP tag resources

Returns The x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type (int, int, int, list((int, int, int, int)), list((int, int)))

static **check_constraints**(*vertices*, *additional_placement_constraints=None*)

Check that the constraints on the given vertices are supported by the resource tracker

Parameters

- **vertices** – The vertices to check the constraints of
- **additional_placement_constraints** – Additional placement constraints supported by the algorithm doing this check

chips_available

The chips currently available

chips_used

deduce the number of chips used in this allocation

Returns the number of chips used during the allocation.

static **get_chip_and_core**(*constraints*, *chips=None*)

Get an assigned chip and core from a set of constraints

Parameters

- **constraints** (iterable of *pacman.model.constraints.AbstractConstraint*) – The set of constraints to get the values from. Note that any type of constraint can be in the list but only those relevant will be used

- **chips** (*iterable of (int, int)*) – Optional list of tuples of (x, y) coordinates of chips, restricting the allowed chips

Returns tuple of a chip x and y coordinates, and processor id, any of which might be None

Return type (tuple of (int, int, int)

static get_ip_tag_info(*resources*, *constraints*)

Get the IP tag resource information

Parameters

- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to get the values from
- **constraints** (list of *pacman.model.constraints.AbstractConstraint*) – A list of constraints

Returns A tuple of board address, iterable of IP tag resources and iterable of reverse IP tag resources

Return type (str, iterable of `pacman.model.resources.IptagResource`, iterable of `pacman.model.resources.ReverseIPtabResource`)

get_maximum_constrained_resources_available(*resources*, *constraints*, *chips=None*)

Get the maximum resources available given the constraints

Parameters

- **resources** (*pacman.model.resources.AbstractResource*) – The resources of the item to check
- **chips** (*iterable of spinn_machine.Chip*) – the chips to locate the max available resources of

get_maximum_resources_available(*chips=None*, *processor_id=None*, *board_address=None*, *ip_tags=None*, *reverse_ip_tags=None*)

Get the maximum resources available

Parameters

- **chips** (*iterable of (int, int)*) – An iterable of (x, y) tuples of chips that are to be used
- **processor_id** (*int*) – the processor id
- **board_address** (*str*) – the board address for locating max resources from
- **ip_tags** (iterable of `pacman.model.resources.IptagResource`) – iterable of IP tag resources
- **reverse_ip_tags** (iterable of `pacman.model.resources.ReverseIptagResource`) – iterable of reverse IP tag resources

Returns a resource which shows max resources available

Return type *pacman.model.resources.ResourceContainer*

is_chip_available(*chip_x*, *chip_y*)

Check if a given chip is available

Parameters

- **chip_x** (*int*) – the x coord of the chip
- **chip_y** (*int*) – the y coord of the chip

Returns True if the chip is available, False otherwise

Return type bool

keys

The chip coordinates assigned

sdram_avilable_on_chip(*chip_x*, *chip_y*)

Get the available SDRAM on the chip at coordinates chip_x, chip_y

Parameters

- **chip_x** – x coord of the chip in question
- **chip_y** – y coord of the chip in question

Returns the SDRAM remaining

unallocate_resources(*chip_x*, *chip_y*, *processor_id*, *resources*, *ip_tags*, *reverse_ip_tags*)

Undo the allocation of resources

Parameters

- **chip_x** (*int*) – the x coord of the chip allocated
- **chip_y** (*int*) – the y coord of the chip allocated
- **processor_id** (*int*) – the processor id
- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to be unallocated
- **ip_tags** (*iterable of (str, int) or None*) – the details of the IP tags allocated
- **reverse_ip_tags** (*iterable of (str, int) or None*) – the details of the reverse IP tags allocated

Return type None

Module contents

class pacman.utilities.utility_objs.**Field**(*lo*, *hi*, *value*, *tag=<SUPPORTED_TAGS.ROUTING: 1>*, *name=None*)

Bases: object

Field object used in a field constraint for key allocation

hi

lo

name

tag

value

class pacman.utilities.utility_objs.**FlexiField**(*flexi_field_name*, *value=None*, *instance_n_keys=None*, *tag=None*, *nested_level=0*)

Bases: object

field who's location is not fixed in key allocation

instance_n_keys

name
The name for this Flexible field

tag

value

class pacman.utilities.utility_objs.**ResourceTracker**(*machine*, *chips=None*, *preallocated_resources=None*)

Bases: object

Tracks the usage of resources of a machine

Parameters

- **machine** (*spinn_machine.Machine*) – The machine to track the usage of
- **chips** (*iterable of (int, int) tuples of coordinates of chips*) – If specified, this list of chips will be used instead of the list from the machine. Note that the order will be maintained, so this can be used either to reduce the set of chips used, or to re-order the chips. Note also that on deallocation, the order is no longer guaranteed.

allocate_constrained_group_resources(*resource_and_constraint_list*, *chips=None*)

Allocates a group of cores on the same chip for these resources

Parameters

- **resource_and_constraint_list** – A list of tuples of (resources, list of constraints) to allocate
- **chips** – a list of chips that can be used

Returns list of The x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type iterable of (int, int, int, list((int, int)), list((int, int)))

allocate_constrained_resources(*resources*, *constraints*, *chips=None*)

Attempts to use the given resources of the machine, constrained by the given placement constraints.

Parameters

- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to be allocated
- **constraints** (list of *pacman.model.constraints.AbstractConstraint*) – the constraints to consider
- **chips** (*iterable of (int, int)*) – The optional list of (x, y) tuples of chip coordinates of chips that can be used. Note that any chips passed in previously will be ignored

Returns The x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type (int, int, int, list((int, int)), list((int, int)))

Raises ***PacmanValueError*** – If the constraints cannot be met given the current allocation of resources

allocate_group_resources(*group_resources*, *chips=None*, *processor_ids=None*, *board_address=None*, *group_ip_tags=None*, *group_reverse_ip_tags=None*)

Attempts to use the given group of resources on a single chip of the machine. Can be given specific place to use the resources, or else it will allocate them on the first place that the resources of the group fit together.

Parameters

- **group_resources** (list of *pacman.model.resources.ResourceContainer*) – The resources to be allocated
- **chips** (*iterable of (int, int)*) – An iterable of (x, y) tuples of chips that are to be used
- **processor_ids** (*list of (int or None)*) – The specific processor to use on any chip for each resource of the group
- **board_address** (*str*) – the board address to allocate resources of a chip
- **group_ip_tags** (list of lists of `pacman.model.resources.IptagResource`) – list of lists of IP tag resources
- **group_reverse_ip_tags** (list of lists of `pacman.model.resources.ReverseIptagResource`) – list of lists of reverse IP tag resources

Returns An iterable of tuples of the x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type iterable of (int, int, int, list((int, int)), list((int, int)))

allocate_resources(*resources*, *chips=None*, *processor_id=None*, *board_address=None*, *ip_tags=None*, *reverse_ip_tags=None*)

Attempts to use the given resources of the machine. Can be given specific place to use the resources, or else it will allocate them on the first place that the resources fit.

Parameters

- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to be allocated
- **chips** (*iterable of (int, int)*) – An iterable of (x, y) tuples of chips that are to be used
- **processor_id** (*int*) – The specific processor to use on any chip.
- **board_address** (*str*) – the board address to allocate resources of a chip
- **ip_tags** (iterable of `pacman.model.resources.IptagResource`) – iterable of IP tag resources
- **reverse_ip_tags** (iterable of *pacman.model.resources.ReverseIPtagResource*) – iterable of reverse IP tag resources

Returns The x and y coordinates of the used chip, the processor_id, and the IP tag and reverse IP tag allocation tuples

Return type (int, int, int, list((int, int, int, int)), list((int, int)))

static **check_constraints**(*vertices*, *additional_placement_constraints=None*)

Check that the constraints on the given vertices are supported by the resource tracker

Parameters

- **vertices** – The vertices to check the constraints of
- **additional_placement_constraints** – Additional placement constraints supported by the algorithm doing this check

chips_available

The chips currently available

chips_used

deduce the number of chips used in this allocation

Returns the number of chips used during the allocation.

static get_chip_and_core(*constraints*, *chips=None*)

Get an assigned chip and core from a set of constraints

Parameters

- **constraints** (iterable of *pacman.model.constraints.AbstractConstraint*) – The set of constraints to get the values from. Note that any type of constraint can be in the list but only those relevant will be used
- **chips** (*iterable of (int, int)*) – Optional list of tuples of (x, y) coordinates of chips, restricting the allowed chips

Returns tuple of a chip x and y coordinates, and processor id, any of which might be None

Return type (tuple of (int, int, int)

static get_ip_tag_info(*resources*, *constraints*)

Get the IP tag resource information

Parameters

- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to get the values from
- **constraints** (list of *pacman.model.constraints.AbstractConstraint*) – A list of constraints

Returns A tuple of board address, iterable of IP tag resources and iterable of reverse IP tag resources

Return type (str, iterable of `pacman.model.resources.IptagResource`, iterable of `pacman.model.resources.ReverseIPtabResource`)

get_maximum_constrained_resources_available(*resources*, *constraints*, *chips=None*)

Get the maximum resources available given the constraints

Parameters

- **resources** (*pacman.model.resources.AbstractResource*) – The resources of the item to check
- **chips** (*iterable of spinn_machine.Chip*) – the chips to locate the max available resources of

get_maximum_resources_available(*chips=None*, *processor_id=None*, *board_address=None*, *ip_tags=None*, *reverse_ip_tags=None*)

Get the maximum resources available

Parameters

- **chips** (*iterable of (int, int)*) – An iterable of (x, y) tuples of chips that are to be used
- **processor_id** (*int*) – the processor id
- **board_address** (*str*) – the board address for locating max resources from
- **ip_tags** (iterable of `pacman.model.resources.IptagResource`) – iterable of IP tag resources
- **reverse_ip_tags** (iterable of `pacman.model.resources.ReverseIptagResource`) – iterable of reverse IP tag resources

Returns a resource which shows max resources available

Return type *pacman.model.resources.ResourceContainer*

is_chip_available(*chip_x*, *chip_y*)
Check if a given chip is available

Parameters

- **chip_x** (*int*) – the x coord of the chip
- **chip_y** (*int*) – the y coord of the chip

Returns True if the chip is available, False otherwise

Return type bool

keys
The chip coordinates assigned

sdram_avilable_on_chip(*chip_x*, *chip_y*)
Get the available SDRAM on the chip at coordinates chip_x, chip_y

Parameters

- **chip_x** – x coord of the chip in question
- **chip_y** – y coord of the chip in question

Returns the SDRAM remaining

unallocate_resources(*chip_x*, *chip_y*, *processor_id*, *resources*, *ip_tags*, *reverse_ip_tags*)
Undo the allocation of resources

Parameters

- **chip_x** (*int*) – the x coord of the chip allocated
- **chip_y** (*int*) – the y coord of the chip allocated
- **processor_id** (*int*) – the processor id
- **resources** (*pacman.model.resources.ResourceContainer*) – The resources to be unallocated
- **ip_tags** (*iterable of (str, int) or None*) – the details of the IP tags allocated
- **reverse_ip_tags** (*iterable of (str, int) or None*) – the details of the reverse IP tags allocated

Return type None

Submodules

pacman.utilities.constants module

class pacman.utilities.constants.**EDGES**
Bases: enum.Enum

EAST = 0

NORTH = 2

NORTH_EAST = 1

SOUTH = 5

SOUTH_WEST = 4

WEST = 3

pacman.utilities.rig_converters module

pacman.utilities.rig_converters.**convert_from_rig_placements**(*rig_placements*, *rig_allocations*, *machine_graph*)

pacman.utilities.rig_converters.**convert_from_rig_routes**(*rig_routes*)

pacman.utilities.rig_converters.**convert_to_rig_graph**(*machine_graph*)

pacman.utilities.rig_converters.**convert_to_rig_graph_pure_mc**(*machine_graph*)

pacman.utilities.rig_converters.**convert_to_rig_machine**(*machine*)

pacman.utilities.rig_converters.**convert_to_rig_placements**(*placements*, *machine*)

pacman.utilities.rig_converters.**create_rig_graph_constraints**(*machine_graph*, *machine*)

pacman.utilities.rig_converters.**create_rig_machine_constraints**(*machine*)

pacman.utilities.utility_calls module

pacman.utilities.utility_calls.**check_algorithm_can_support_constraints**(*constrained_vertices*, *supported_constraints*, *abstract_constraint_type*)

Helper method to find out if an algorithm can support all the constraints given the objects its expected to work on

Parameters

- **constrained_vertices** (iterable of *pacman.model.constraints.AbstractConstraint*) – a list of constrained vertices which each has constraints given to the algorithm
- **supported_constraints** (iterable of *pacman.model.constraints.AbstractConstraint*) – The constraints supported
- **abstract_constraint_type** (*pacman.model.constraints.AbstractConstraint*) – The overall abstract c type supported

Returns Nothing is returned

Return type None

Raises *pacman.exceptions.PacmanInvalidParameterException* – when the algorithm cannot support the constraints demanded of it

pacman.utilities.utility_calls.**check_constrained_value**(*value*, *current_value*)

Checks that the current value and a new value match

Parameters

- **value** – The value to check
- **current_value** – The existing value

`pacman.utilities.utility_calls.`**`compress_bits_from_bit_array`**(*bit_array*, *bit_positions*)

Compress specific positions from a bit array of 32 uint8 value, where is a 1 or 0, into a 32-bit value.

Parameters

- **bit_array** (*[uint8]*) – The array to extract the value from
- **bit_positions** (*[int]*) – The positions of the bits to extract, each value being between 0 and 31

Return type int

`pacman.utilities.utility_calls.`**`compress_from_bit_array`**(*bit_array*)

Compress a bit array of 32 uint8 values, where each is a 1 or 0, into a 32-bit value

Parameters **bit_array** (*[uint8]*) – The array to compress

Return type int

`pacman.utilities.utility_calls.`**`expand_to_bit_array`**(*value*)

Expand a 32-bit value in to an array of length 32 of uint8 values, each of which is a 1 or 0

Parameters **value** (*int*) – The value to expand

Return type [uint8]

`pacman.utilities.utility_calls.`**`is_equal_or_None`**(*a*, *b*)

If a and b are both not None, return True iff they are equal, otherwise return True

`pacman.utilities.utility_calls.`**`is_single`**(*iterable*)

Test if there is exactly one item in the iterable

`pacman.utilities.utility_calls.`**`locate_constraints_of_type`**(*constraints*, *constraint_type*)

Locates all constraints of a given type out of a list

Parameters

- **constraints** (iterable of *pacman.model.constraints.AbstractConstraint*) – The constraints to filter
- **constraint_type** (pacman.model.constraints.partitioner_constraints.AbstractPartitionConstraint) – The type of constraints to return

Returns The constraints of constraint_type that are found in the constraints given

Return type iterable of *pacman.model.constraints.AbstractConstraint*

Raises **None** – no known exceptions

`pacman.utilities.utility_calls.`**`locate_first_constraint_of_type`**(*constraints*, *constraint_type*)

Locates the first constraint of a given type out of a list

Parameters

- **constraints** (iterable of *pacman.model.constraints.AbstractConstraint*) – The constraints to select from
- **constraint_type** (pacman.model.constraints.partitioner_constraints.AbstractPartitionConstraint) – The type of constraints to return

Returns The first constraint of constraint_type that was found in the constraints given

Return type `pacman.model.constraints.AbstractConstraint`

Raises ***pacman.exceptions.PacmanInvalidParameterException*** – if no such constraint is present

pacman.utilities.vertex_sorter module

class `pacman.utilities.vertex_sorter.`**`ConstraintOrder`**(*constraint_class*, *relative_order*, *required_optional_properties=None*)

Bases: `object`

A constraint order definition for sorting

Parameters

- **`constraint_class`** – The class of the constraint
- **`relative_order`** – The order of the constraint relative to other constraints to be sorted
- **`required_optional_properties`** – Properties of the constraint instances that must not be None for the constraint to match this ordering

`constraint_class`
property method for the constraint class

`relative_order`
property method for the relative order

`required_optional_properties`
property method for the required optional properties

class `pacman.utilities.vertex_sorter.`**`VertexSorter`**(*constraint_order*)

Bases: `object`

Sorts vertices based on constraints with given criteria

Parameters **`constraint_order`** (`list of ConstraintOrder`) – The order in which the constraints are to be sorted

`sort`(*vertices*)
Sort the given set of vertices by the constraint ordering

Parameters **`vertices`** – The vertices to sort

Returns The sorted list of vertices

Module contents

class `pacman.utilities.`**`ConstraintOrder`**(*constraint_class*, *relative_order*, *required_optional_properties=None*)

Bases: `object`

A constraint order definition for sorting

Parameters

- **`constraint_class`** – The class of the constraint
- **`relative_order`** – The order of the constraint relative to other constraints to be sorted

- **required_optional_properties** – Properties of the constraint instances that must not be None for the constraint to match this ordering

constraint_class
property method for the constraint class

relative_order
property method for the relative order

required_optional_properties
property method for the required optional properties

class pacman.utilities.**VertexSorter**(*constraint_order*)
Bases: object

Sorts vertices based on constraints with given criteria

Parameters **constraint_order** (*list of ConstraintOrder*) – The order in which the constraints are to be sorted

sort(*vertices*)
Sort the given set of vertices by the constraint ordering

Parameters **vertices** – The vertices to sort

Returns The sorted list of vertices

Submodules

pacman.exceptions module

exception pacman.exceptions.**PacmanAlgorithmFailedToCompleteException**(*algorithm*, *exception*, *tb*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that a pacman algorithm ran from inside the software stack has failed to complete for some unknown reason.

algorithm
The algorithm that raised the exception

exception
The exception that caused this exception

traceback
The traceback of the exception that caused this exception

exception pacman.exceptions.**PacmanAlgorithmFailedToGenerateOutputsException**(**args*, ***kwargs*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that an algorithm has not generated the correct outputs for some unknown reason

exception pacman.exceptions.**PacmanAlreadyExistsException**(*item_type*, *item_id*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something already exists and that adding another would be a conflict

Parameters

- **item_type** (*str*) – The type of the item that already exists

- **item_id** (*str*) – The id of the item which is in conflict

exception pacman.exceptions.**PacmanAlreadyPlacedError**
Bases: exceptions.ValueError

Indicates multiple placements are being made for a vertex.

exception pacman.exceptions.**PacmanConfigurationException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something went wrong with configuring some part of pacman

Parameters **problem** (*str*) – The problem with the routing

exception pacman.exceptions.**PacmanElementAllocationException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something went wrong with element allocation

Parameters **problem** (*str*) – The problem with the allocation

exception pacman.exceptions.**PacmanException**(**args*, ***kwargs*)
Bases: exceptions.Exception

Indicates a general exception from Pacman

exception pacman.exceptions.**PacmanExternalAlgorithmFailedToCompleteException**(**args*, ***kwargs*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that an algorithm ran from outside the software stack has failed to complete for some unknown reason.

exception pacman.exceptions.**PacmanInvalidParameterException**(*parameter*, *value*, *problem*)
Bases: *pacman.exceptions.PacmanException*

An exception which indicates that a parameter has an invalid value

Parameters

- **parameter** (*str*) – The name of the parameter
- **value** (*str*) – The value of the parameter
- **problem** (*str*) – The problem with the value of the parameter

exception pacman.exceptions.**PacmanNoMergeException**(**args*, ***kwargs*)
Bases: *pacman.exceptions.PacmanException*

Exception to indicate that there are no merges worth performing.

exception pacman.exceptions.**PacmanNotExistException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that a routing table entry was attempted to be removed from a routing table which didn't have such an entry

Parameters **problem** (*str*) – The problem with the routing

exception pacman.exceptions.**PacmanNotFoundError**
Bases: exceptions.KeyError, *pacman.exceptions.PacmanException*

Indicates that some object has not been found when requested.

exception pacman.exceptions.**PacmanNotPlacedError**
Bases: exceptions.KeyError

Indicates no placements are made for a vertex.

exception pacman.exceptions.**PacmanPartitionException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something went wrong with partitioning

Parameters **problem** (*str*) – The problem with the partitioning

exception pacman.exceptions.**PacmanPlaceException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something went wrong with placement

Parameters **problem** (*str*) – The problem with the placement

exception pacman.exceptions.**PacmanProcessorAlreadyOccupiedError**
Bases: exceptions.ValueError

Indicates multiple placements are being made to a processor.

exception pacman.exceptions.**PacmanProcessorNotOccupiedError**
Bases: exceptions.KeyError

Indicates that no placement has been made to a processor.

exception pacman.exceptions.**PacmanPruneException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something went wrong with pruning

Parameters **problem** (*str*) – The problem with the pruning

exception pacman.exceptions.**PacmanRouteInfoAllocationException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something went wrong with route info allocation

Parameters **problem** (*str*) – The problem with the allocation

exception pacman.exceptions.**PacmanRoutingException**(*problem*)
Bases: *pacman.exceptions.PacmanException*

An exception that indicates that something went wrong with routing

Parameters **problem** (*str*) – The problem with the routing

exception pacman.exceptions.**PacmanTypeError**
Bases: exceptions.TypeError, *pacman.exceptions.PacmanException*

Indicates that an object is of incorrect type.

exception pacman.exceptions.**PacmanValueError**
Bases: exceptions.ValueError, *pacman.exceptions.PacmanException*

Indicates that a value is invalid for some reason.

Module contents

Provides various functions which together can be used to take a graph and split it into pieces that can be loaded on to a machine, along with routes between the pieces.

Functional Requirements

- Creation of an Application Graph of Vertices indicating points of computation within the graph and Edges between the vertices indicating a directional communication between the vertices; and a similar Machine Graph.
  - Vertices in the Application Graph will have a number of *atoms* - an atom cannot be broken down in to anything smaller.
  - Vertices in the Application Graph must be able to indicate what machine resources are required by any given subset of the atoms.
  - Vertices in the Machine Graph must be able to fit on a single chip of the machine in terms of resource usage.
  - A Vertex can have a number of constraints which must be respected by any algorithm which uses the graph. Algorithms must check that they can support the given constraints and must fail if they cannot. Provided constraints include support for:
    * The maximum number of atoms which any Machine Graph Vertex can contain for a given Application Graph vertex
    * The chip and/or processor on to which a Machine Graph Vertex should be placed.
    * A set of Application Graph Vertices whose corresponding Machine Graph vertices should contain the same number of atoms.
    * A set of Application Graph Vertices whose corresponding Machine Graph vertices should be placed on the same chip if they contain the same atom.
  - It should be possible to create new constraints as the need arises.
  - Multiple edges can exist between the same two vertices.
  - It must be possible to build the Machine Graph directly without requiring that it is created by one of the other modules.
  - It is *not* required that there is a Machine Graph Edge between every pair of Machine Graph Vertex from the same Application Graph Vertex.
  - Where a Machine Graph is created from an Application Graph, it should be possible to find the corresponding Vertices and Edges from one graph to the other.
- Creation of multicast routing info consisting of key/mask combinations assigned to Edges of the Machine Graph.
  - It must be possible to build this information directly without requiring that it is created by one of the other modules.
  - There should be exactly one key/mask combination for each Edge in the Machine Graph, which will represent all the keys which will be sent in all packets from the Vertex at the start of the Edge down that Edge.
  - It is possible for a Vertex to send several different keys down several different Edges, but only one per Edge (but note that it is acceptable for different keys to be assigned to different Edges between the same two Vertices).
  - There should be no overlap between the key/mask combinations of Edges which come from different Vertices i.e. no two Edges which start at different Vertices should have the same key/mask combination.
- Partitioning of an Application graph with respect to a machine, such that the resources consumed by each Vertex does not exceed those provided by each chip on the machine.
  - It should be possible to select from a range of partitioning algorithms or provide one, although a default should be provided in the absence of such a choice .

  - Any partitioning constraints should be met; if there are any that cannot, or that are not understood by the algorithm in use an exception should be thrown. Non-partitioning constraints can be ignored, although these can be used if it makes sense for the given algorithm.
  - It must be possible to create at least one grouping of the generated Vertices so that each group fits within the resources provided by a single chip on the machine.
  - It should not be assumed that a given grouping of Vertices will be the final grouping on the machine, although it is acceptable to make hints through additional constraints about what is likely to work.
  - The machine itself must not be altered by the partitioning, so that it can be used in further processing.
  - The graph itself must not be altered by the partitioning, so that it can be used in further processing.
  - No two Machine Graph Vertices created from a single Application Graph Vertex can contain the same atom.
  - Any Edges in the Application Graph must be split with the Vertices to create a number of Machine Graph edges, such that where there was a vertex v connected to a vertex w by a single edge in the Application Graph, there should be an Edge in the Machine Graph between every Vertex of Application Graph Vertex v and every Vertex of Application Graph Vertex w; for example, if there are 2 Machine Graph Vertices for each of v and w, and one Edge between them in the Application Graph, then there will be 4 new Edges in the Machine Graph for this Edge.
- Placement of a Machine Graph on a given machine, such that the resources required by any combination of Vertices placed on any chip in the machine does not exceed the resources provided by that chip.
  - It should be possible to choose from a range of placement algorithms or provide one, although a default should be provided in the absence of such a choice.
  - Any placement constraints should be met; if there are any that cannot, or that are not understood by placement algorithm, an exception should be thrown. Non-placement constraints can be ignored, although these can be used if it makes sense for the given algorithm.
  - The machine itself should not be altered by placement so that it can be used in further processing.
  - The graph itself should not be altered by placement so that it can be used in further processing.
  - The returned placements should only contain a single placement for each vertex.
  - The placements should be such that the vertices with edges between them must be able to communicate with each other.
- Allocation of multicast routing keys and masks to a Machine Graph such that each vertex sends out packets with a different key/mask combination.
  - This can use the placement information if required. If an algorithm requires placement information but none is provided an exception is thrown.
- Routing of edges between vertices with a given allocation of routing keys and masks with respect to a given machine.
  - It should be possible to choose from a range of routing algorithms, or provide one, although a default should be provided in the absence of such a choice
  - For any vertex, following the routes from the placement of the vertex should result exactly in the set of placements of the destination vertices described by all the edges which start at that vertex. No additional destination should be reached, and no fewer than this set of destinations should be reached.
- It should be possible to call each of the modules independently. There should be no assumption that one of the other modules has produced the data input for any other module.
- There should be no assumption about how the inputs and outputs are stored.

- Any utility functions that provide access to internal structures within a data structure should operate in approximately O(1) time; for example, where an object of type *obj* holds a number of objects of type *subobj* with property *prop*, requesting a list of *subobj* objects contained within *obj* with property value *prop* = *value* should not iterate through a list of such objects, but should instead maintain a mapping that allows access to such objects in O(1) time. If this is not possible, *obj* should only provide access to a list of *subobj* objects, allowing the caller to filter these themselves. This will ensure that no misunderstanding can be made about the speed of operation of a function.

4.1.3 Indices and tables

- genindex
- modindex
- search

PACMAN_github

PACMAN_individual_docs

CHAPTER 5

SpiNNMan

This package provides utilities for interacting with a SpiNNaker machine.

5.1 SpiNNMan

Used to communicate with a SpiNNaker Board. The main part of this package is the `spinnman.transceiver.Transceiver` class. This can be used to send and receive packets in various SpiNNaker formats, depending on what connections are available.

5.1.1 Functional Requirements

- Connect to and communicate with a machine using a number of different connections.
- Boot a machine with the expected version of the software.
  - If the machine is already booted but the version is not the version expected, an exception will be thrown.
- Check the version of the software which the machine is booted with.
- Query the state of the machine to determine:
  - What the current state of the machine is in terms of the chips and cores available, the SDRAM available on the chips and the which links are available between which chips.
  - What external links to the host exist (and separately add the discovered links to the set of links used to communicate with the machine).
  - What is running on the machine and where, and what the current status of those processes are.
  - How many cores are in a given state.
  - What is in the IOBUF buffers.
  - What the current routing entries for a given router are.
  - What the routing status counter values are.

- Load application binaries on to the machine, either to individual cores or via a "flood-fill" mechanism to multiple cores simultaneously (which may be a subset of the cores on a subset of the chips).
- Write data to SDRAM, either on an individual chip, or via a "flood-fill" mechanism to multiple chips simultaneously.
- Send a signal to an application.
- Read data from SDRAM on an individual chip.
- Send and receive SpiNNaker packets where the connections allow this.
  - If no connection supports this packet type, an exception is thrown.
  - The user should be able to select which connection is used. Selection of a connection which does not support the traffic type will also result in an exception.
- Send and receive SCP and SDP packets where the connections allow this.
  - If no connection supports the packet type, an exception is thrown.
  - The user should be able to select which connection is used. Selection of a connection which does not support the traffic type will also result in an exception.
- It should be possible to call any of the functions simultaneously, including the same function more than once.
  - Where possible, multiple connections should be used to overlap calls.
  - The functions should not return until they have confirmed that any messages sent have been received, and any responses have been received.
  - Functions should not respond with the result of a different function.
  - Functions can further sub-divide the call into a number of separate calls that can be divided across the available connections, so long as the other requirements are met.
- More than one machine can be connected to the same host.
  - Once the subset of connections has been worked out for each machine, the operation of these machines should be independent.

5.1.2 Use Cases

- *boot_board()* and *get_scamp_version()* are used to ensure that the board is booted correctly before starting a simulation.
- *get_machine_details()* is used to get a representation of the current state of the machine, which is used to decide where executables are to be run on the board for a particular simulation, where any external peripherals are connected, and how messages between the executables and/or the external peripherals are to be routed
- *write_memory()* and *execute()* are used to write parameters and execute executables on the board
- *send_signal()* is used to send a signal which starts, stops or pauses a simulation.
- `get_core_status_count()` is used to determine if a simulation is complete or has gone into an error state.
- *get_iobuf()*, *get_cpu_information()* and *get_router_diagnostics()* are used to diagnose a problem with a simulation
- *read_memory()* is used to read some statistics recorded in SDRAM after a simulation

5.1.3 Contents

spinnman

spinnman package

Subpackages

spinnman.connections package

Subpackages

spinnman.connections.abstract_classes package

Submodules

spinnman.connections.abstract_classes.connection module

class spinnman.connections.abstract_classes.connection.**Connection**

Bases: object

An abstract connection to the SpiNNaker board over some medium

close()

Closes the connection

Returns Nothing is returned

Return type None

Raises **None** – No known exceptions are raised

is_connected()

Determines if the medium is connected at this point in time

Returns True if the medium is connected, False otherwise

Return type bool

Raises *spinnman.exceptions.SpinnmanIOException* – If there is an error when determining the connectivity of the medium

spinnman.connections.abstract_classes.eieio_receiver module

class spinnman.connections.abstract_classes.eieio_receiver.**EIEIOReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of EIEIO data or commands

receive_eieio_message(*timeout=None*)

Receives an EIEIO message from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns an EIEIO message

Return type spinnman.messages.eieio.abstract_messages.abstractMulticastMessage

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If the received packet is not a valid EIEIO message
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If one of the fields of the EIEIO message is invalid

spinnman.connections.abstract_classes.eieio_sender module

class spinnman.connections.abstract_classes.eieio_sender.**EIEIOSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of EIEIO messages

send_eieio_message(*eieio_message*)

Sends an EIEIO message down this connection

Parameters **eieio_message** (spinnman.messages.eieio.abstract_messages.abstract_eieio_message.AbstractEIEIOMessage) – The eieio message to be sent

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error sending the message

spinnman.connections.abstract_classes.listenable module

class spinnman.connections.abstract_classes.listenable.**Listenable**

Bases: object

get_receive_method()

Get the method that receives for this connection

is_ready_to_receive(*timeout=0*)

Determines if there is an SCP packet to be read without blocking

Parameters **timeout** (*int*) – The time to wait before returning if the connection is not ready

Returns True if there is an SCP packet to be read

Return type bool

spinnman.connections.abstract_classes.multicast_receiver module

class spinnman.connections.abstract_classes.multicast_receiver.**MulticastReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of Multicast messages

get_input_chips()
Get a list of chips which identify the chips from which this receiver can receive receive packets directly

Returns An iterable of tuples of (x, y) where x is the x-coordinate of the chip and y is the y-coordinate of the chip

Return type iterable of (int, int)

Raises **None** – No known exceptions are raised

receive_multicast_message(*timeout=None*)
Receives a multicast message from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns a multicast message

Return type *spinnman.messages.multicast_message.MulticastMessage*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If the received packet is not a valid multicast message
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If one of the fields of the multicast message is invalid

spinnman.connections.abstract_classes.multicast_sender module

class spinnman.connections.abstract_classes.multicast_sender.**MulticastSender**
Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of Multicast messages

get_input_chips()
Get a list of chips which identify the chips to which this sender can send multicast packets directly

Returns An iterable of tuples of (x, y) where x is the x-coordinate of the chip and y is the y-coordinate of the chip

Return type iterable of (int, int)

Raises **None** – No known exceptions are raised

send_multicast_message(*multicast_message*)
Sends a SpiNNaker multicast message using this connection

Parameters **multicast_message** (*spinnman.messages.multicast_message.MulticastMessage*) – The message to be sent

Returns Nothing is returned

Return type None

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error sending the message

spinnman.connections.abstract_classes.scp_receiver module

class spinnman.connections.abstract_classes.scp_receiver.**SCPReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of SCP messages

is_ready_to_receive(*timeout=0*)

Determines if there is an SCP packet to be read without blocking

Parameters **timeout** (*int*) – The time to wait before returning if the connection is not ready

Returns True if there is an SCP packet to be read

Return type bool

receive_scp_response(*timeout=1.0*)

Receives an SCP response from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns The SCP result, the sequence number, the data of the response and the offset at which the data starts (i.e. where the SDP header starts)

Return type :py:class: *spinnman.messages.scp.scp_result.SCPResult*, int, bytestring, int

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received

spinnman.connections.abstract_classes.scp_sender module

class spinnman.connections.abstract_classes.scp_sender.**SCPSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of SCP messages

chip_x

The x-coordinate of the chip at which messages sent down this connection will arrive at first

Return type int

chip_y

The y-coordinate of the chip at which messages sent down this connection will arrive at first

Return type int

get_scp_data(*scp_request*)

Returns the data of an SCP request as it would be sent down this connection

send_scp_request(*scp_request*)

Sends an SCP request down this connection

Messages must have the following properties:

- source_port is None or 7
- source_cpu is None or 31

- source_chip_x is None or 0
- source_chip_y is None or 0

tag in the message is optional - if not set the default set in the constructor will be used. sequence in the message is optional - if not set (sequence number last assigned + 1) % 65536 will be used

Parameters **scp_request** (spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest) – message packet to send

Returns Nothing is returned

Return type None

Raises *spinnman.exceptions.SpinnmanIOException* – If there is an error sending the message

spinnman.connections.abstract_classes.sdp_receiver module

class spinnman.connections.abstract_classes.sdp_receiver.**SDPReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of SDP messages

receive_sdp_message(*timeout=None*)

Receives an SDP message from this connection. Blocks until the message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns The received SDP message

Return type *spinnman.messages.sdp.sdp_message.SDPMessage*

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error receiving the message
- *spinnman.exceptions.SpinnmanTimeoutException* – If there is a timeout before a message is received
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If the received packet is not a valid SDP message
- *spinnman.exceptions.SpinnmanInvalidParameterException* – If one of the fields of the SDP message is invalid

spinnman.connections.abstract_classes.sdp_sender module

class spinnman.connections.abstract_classes.sdp_sender.**SDPSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of SDP messages

send_sdp_message(*sdp_message*)

Sends an SDP message down this connection

Parameters **sdp_message** (spinnman.messages.sdp.sdp_message.SDPMessage) – The SDP message to be sent

Returns Nothing is returned

Return type None

Raises *spinnman.exceptions.SpinnmanIOException* – If there is an error sending the message

spinnman.connections.abstract_classes.spinnaker_boot_receiver module

class spinnman.connections.abstract_classes.spinnaker_boot_receiver.**SpinnakerBootReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of Spinnaker boot messages

receive_boot_message(*timeout=None*)

Receives a boot message from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns a boot message

Return type *spinnman.messages.spinnaker_boot.spinnaker_boot_message.SpinnakerBootMessage*

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error receiving the message
- *spinnman.exceptions.SpinnmanTimeoutException* – If there is a timeout before a message is received
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If the received packet is not a valid spinnaker boot message
- *spinnman.exceptions.SpinnmanInvalidParameterException* – If one of the fields of the spinnaker boot message is invalid

spinnman.connections.abstract_classes.spinnaker_boot_sender module

class spinnman.connections.abstract_classes.spinnaker_boot_sender.**SpinnakerBootSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of Spinnaker Boot messages

send_boot_message(*boot_message*)

Sends a SpiNNaker boot message using this connection

Parameters **boot_message** (*spinnman.messages.spinnaker_boot.spinnaker_boot_message.SpinnakerBootMessage*) – The message to be sent

Returns Nothing is returned

Return type None

Raises *spinnman.exceptions.SpinnmanIOException* – If there is an error sending the message

Module contents

class spinnman.connections.abstract_classes.**Connection**

Bases: object

An abstract connection to the SpiNNaker board over some medium

close()

Closes the connection

Returns Nothing is returned

Return type None

Raises **None** – No known exceptions are raised

is_connected()

Determines if the medium is connected at this point in time

Returns True if the medium is connected, False otherwise

Return type bool

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error when determining the connectivity of the medium

class spinnman.connections.abstract_classes.**EIEIOReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of EIEIO data or commands

receive_eieio_message(*timeout=None*)

Receives an EIEIO message from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns an EIEIO message

Return type spinnman.messages.eieio.abstract_messages.abstractMulticastMessage

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If the received packet is not a valid EIEIO message
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If one of the fields of the EIEIO message is invalid

class spinnman.connections.abstract_classes.**EIEIOSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of EIEIO messages

send_eieio_message(*eieio_message*)

Sends an EIEIO message down this connection

Parameters **eieio_message** (spinnman.messages.eieio.abstract_messages.abstract_eieio_message.AbstractEIEIOMessage) – The eieio message to be sent

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error sending the message

class spinnman.connections.abstract_classes.**Listenable**

Bases: object

get_receive_method()

Get the method that receives for this connection

is_ready_to_receive(*timeout=0*)

Determines if there is an SCP packet to be read without blocking

Parameters **timeout** (*int*) – The time to wait before returning if the connection is not ready

Returns True if there is an SCP packet to be read

Return type bool

class spinnman.connections.abstract_classes.**MulticastReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of Multicast messages

get_input_chips()

Get a list of chips which identify the chips from which this receiver can receive receive packets directly

Returns An iterable of tuples of (x, y) where x is the x-coordinate of the chip and y is the y-coordinate of the chip

Return type iterable of (int, int)

Raises **None** – No known exceptions are raised

receive_multicast_message(*timeout=None*)

Receives a multicast message from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns a multicast message

Return type *spinnman.messages.multicast_message.MulticastMessage*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If the received packet is not a valid multicast message
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If one of the fields of the multicast message is invalid

class spinnman.connections.abstract_classes.**MulticastSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of Multicast messages

get_input_chips()

Get a list of chips which identify the chips to which this sender can send multicast packets directly

Returns An iterable of tuples of (x, y) where x is the x-coordinate of the chip and y is the y-coordinate of the chip

Return type iterable of (int, int)

Raises **None** – No known exceptions are raised

send_multicast_message(*multicast_message*)

Sends a SpiNNaker multicast message using this connection

Parameters **multicast_message** (*spinnman.messages.multicast_message.MulticastMessage*) – The message to be sent

Returns Nothing is returned

Return type None

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error sending the message

class spinnman.connections.abstract_classes.**SCPReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of SCP messages

is_ready_to_receive(*timeout=0*)

Determines if there is an SCP packet to be read without blocking

Parameters **timeout** (*int*) – The time to wait before returning if the connection is not ready

Returns True if there is an SCP packet to be read

Return type bool

receive_scp_response(*timeout=1.0*)

Receives an SCP response from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns The SCP result, the sequence number, the data of the response and the offset at which the data starts (i.e. where the SDP header starts)

Return type :py:class: *spinnman.messages.scp.scp_result.SCPResult*, int, bytestring, int

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received

class spinnman.connections.abstract_classes.**SCPSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of SCP messages

chip_x

The x-coordinate of the chip at which messages sent down this connection will arrive at first

Return type int

chip_y
The y-coordinate of the chip at which messages sent down this connection will arrive at first

Return type int

get_scp_data (*scp_request*)
Returns the data of an SCP request as it would be sent down this connection

send_scp_request (*scp_request*)
Sends an SCP request down this connection

Messages must have the following properties:

- source_port is None or 7
- source_cpu is None or 31
- source_chip_x is None or 0
- source_chip_y is None or 0

tag in the message is optional - if not set the default set in the constructor will be used. sequence in the message is optional - if not set (sequence number last assigned + 1) % 65536 will be used

Parameters **scp_request** (`spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest`) – message packet to send

Returns Nothing is returned

Return type None

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error sending the message

class `spinnman.connections.abstract_classes.`**SDPReceiver**
Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of SDP messages

receive_sdp_message (*timeout=None*)
Receives an SDP message from this connection. Blocks until the message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns The received SDP message

Return type *spinnman.messages.sdp.sdp_message.SDPMessage*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If the received packet is not a valid SDP message
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If one of the fields of the SDP message is invalid

class spinnman.connections.abstract_classes.**SDPSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of SDP messages

send_sdp_message(*sdp_message*)

Sends an SDP message down this connection

Parameters **sdp_message** (spinnman.messages.sdp.sdp_message.SDPMessage) – The SDP message to be sent

Returns Nothing is returned

Return type None

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error sending the message

class spinnman.connections.abstract_classes.**SpinnakerBootReceiver**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A receiver of Spinnaker boot messages

receive_boot_message(*timeout=None*)

Receives a boot message from this connection. Blocks until a message has been received, or a timeout occurs.

Parameters **timeout** (*int*) – The time in seconds to wait for the message to arrive; if not specified, will wait forever, or until the connection is closed

Returns a boot message

Return type *spinnman.messages.spinnaker_boot.spinnaker_boot_message.SpinnakerBootMessage*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error receiving the message
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If the received packet is not a valid spinnaker boot message
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If one of the fields of the spinnaker boot message is invalid

class spinnman.connections.abstract_classes.**SpinnakerBootSender**

Bases: *spinnman.connections.abstract_classes.connection.Connection*

A sender of Spinnaker Boot messages

send_boot_message(*boot_message*)

Sends a SpiNNaker boot message using this connection

Parameters **boot_message** (*spinnman.messages.spinnaker_boot.spinnaker_boot_message.SpinnakerBootMessage*) – The message to be sent

Returns Nothing is returned

Return type None

Raises *spinnman.exceptions.SpinnmanIOException* – If there is an error sending the message

spinnman.connections.udp_packet_connections package

Submodules

spinnman.connections.udp_packet_connections.bmp_connection module

class spinnman.connections.udp_packet_connections.bmp_connection.**BMPConnection**(*cabinet*, *frame*, *boards*, *local_host=None*, *local_port=None*, *remote_host=None*, *remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.scp_receiver.SCPReceiver*, *spinnman.connections.abstract_classes.scp_sender.SCPSender*

A BMP connection which supports queries to the BMP of a SpiNNaker machine

Parameters

- **cabinet** (*int*) – The cabinet number of the connection
- **frame** (*int*) – The frame number of the connection
- **boards** (*iterable of int*) – The boards that the connection can control on the same backplane
- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

boards

The set of boards supported by the BMP

Return type iterable of int

cabinet

The cabinet id of the BMP

Return type int

chip_x

Defined to satisfy the SCPSender - always 0 for a BMP

chip_y
Defined to satisfy the SCPSender - always 0 for a BMP

frame
The frame id of the BMP

Return type int

get_scp_data(*scp_request*)

receive_scp_response(*timeout=1.0*)

send_scp_request(*scp_request*)

spinnman.connections.udp_packet_connections.boot_connection module

class spinnman.connections.udp_packet_connections.boot_connection.**BootConnection**(*local_host=None, local_port=None, remote_host=None, remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.spinnaker_boot_sender.SpinnakerBootSender*, *spinnman.connections.abstract_classes.spinnaker_boot_receiver.SpinnakerBootReceiver*

A connection to the spinnaker board that uses UDP to for booting

Parameters

- **local_host** (*str*) – The local host name or ip address to bind to. If not specified defaults to bind to all interfaces, unless remote_host is specified, in which case binding is _done to the ip address that will be used to send packets
- **local_port** (*int*) – The local port to bind to, between 1025 and 65535. If not specified, defaults to a random unused local port
- **remote_host** (*str*) – The remote host name or ip address to send packets to. If not specified, the socket will be available for listening only, and will throw and exception if used for sending
- **remote_port** – The remote port to send packets to. If remote_host is None, this is ignored.

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error setting up the communication channel

receive_boot_message(*timeout=None*)
See *spinnman.connections.abstract_classes.spinnaker_boot_receiver.SpinnakerBootReceiver.receive_boot_message()*

send_boot_message(*boot_message*)
See *spinnman.connections.abstract_classes.spinnaker_boot_sender.SpinnakerBootSender.send_boot_message()*

spinnman.connections.udp_packet_connections.eieio_connection module

class spinnman.connections.udp_packet_connections.eieio_connection.**EIEIOConnection**(*local_host=No*
lo-
cal_port=Non
re-
mote_host=No
re-
mote_port=No

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.eieio_receiver.EIEIOReceiver*, *spinnman.connections.abstract_classes.eieio_sender.EIEIOSender*, *spinnman.connections.abstract_classes.listenable.Listenable*

A UDP connection for sending and receiving raw EIEIO messages

Parameters

- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

get_receive_method()

receive_eieio_message(*timeout=None*)

send_eieio_message(*eieio_message*)

send_eieio_message_to(*eieio_message*, *ip_address*, *port*)

spinnman.connections.udp_packet_connections.ip_address_connection module

class spinnman.connections.udp_packet_connections.ip_address_connection.**IPAddressesConnectio**

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*

A connection that detects any UDP packet that is transmitted by spinnaker boards prior to boot

receive_ip_address(*timeout=None*)

supports_sends_message(*message*)

spinnman.connections.udp_packet_connections.scamp_connection module

class spinnman.connections.udp_packet_connections.scamp_connection.**SCAMPConnection**(*chip_x=255, chip_y=255, local_host=Non local_port=Non remote_host=No remote_port=No*

Bases: *spinnman.connections.udp_packet_connections.sdp_connection.SDPConnection*, *spinnman.connections.abstract_classes.scp_sender.SCPSender*, *spinnman.connections.abstract_classes.scp_receiver.SCPReceiver*

A UDP connection to SCAMP on the board

Parameters

- **chip_x** (*int*) – The x-coordinate of the chip on the board with this remote_host
- **chip_y** (*int*) – The y-coordinate of the chip on the board with this remote_host
- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** (*int*) – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

chip_x

chip_y

get_scp_data(*scp_request*)

receive_scp_response(*timeout=1.0*)

send_scp_request(*scp_request*)

update_chip_coordinates(*x*, *y*)

spinnman.connections.udp_packet_connections.sdp_connection module

class spinnman.connections.udp_packet_connections.sdp_connection.**SDPConnection**(*chip_x=None, chip_y=None, local_host=None, local_port=None, remote_host=None, remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.*

UDPConnection, *spinnman.connections.abstract_classes.sdp_receiver.SDPReceiver*, *spinnman.connections.abstract_classes.sdp_sender.SDPSender*, *spinnman.connections.abstract_classes.listenable.Listenable*

Parameters

- **chip_x** (*int*) – The optional x-coordinate of the chip at the remote end of the connection. If not specified, it will not be possible to send SDP messages that require a response with this connection.
- **chip_y** (*int*) – The optional y-coordinate of the chip at the remote end of the connection. If not specified, it will not be possible to send SDP messages that require a response with this connection.
- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

get_receive_method()

receive_sdp_message(*timeout=None*)

send_sdp_message(*sdp_message*)

spinnman.connections.udp_packet_connections.udp_connection module

class spinnman.connections.udp_packet_connections.udp_connection.**UDPConnection**(*local_host=None*, *local_port=None*, *remote_host=None*, *remote_port=None*)

Bases: *spinnman.connections.abstract_classes.connection.Connection*

Parameters

- **local_host** (*str or None*) – The local host name or ip address to bind to. If not specified defaults to bind to all interfaces, unless remote_host is specified, in which case binding is _done to the ip address that will be used to send packets
- **local_port** (*int*) – The local port to bind to, between 1025 and 65535. If not specified, defaults to a random unused local port
- **remote_host** (*str or None*) – The remote host name or ip address to send packets to. If not specified, the socket will be available for listening only, and will throw and exception if used for sending
- **remote_port** – The remote port to send packets to. If remote_host is None, this is ignored. If remote_host is specified, this must also be specified for the connection to allow sending

Raises *spinnman.exceptions.SpinnmanIOException* – If there is an error setting up the communication channel

close()
See *spinnman.connections.abstract_classes.connection.Connection.close()*

is_connected()
See *spinnman.connections.abstract_classes.connection.Connection.is_connected()*

is_ready_to_receive(*timeout=0*)

local_ip_address
The local IP address to which the connection is bound.

Returns The local ip address as a dotted string e.g. 0.0.0.0

Return type str

Raises **None** – No known exceptions are thrown

local_port
The local port to which the connection is bound.

Returns The local port number

Return type int

Raises **None** – No known exceptions are thrown

receive(*timeout=None*)
Receive data from the connection

Parameters **timeout** (*None or float*) – The timeout in seconds, or None to wait forever

Returns The data received as a bytestring

Return type str

Raises

- ***SpinnmanTimeoutException*** – If a timeout occurs before any data is received
- ***SpinnmanIOException*** – If an error occurs receiving the data

receive_with_address(*timeout=None*)
Receive data from the connection along with the address where the data was received from

Parameters **timeout** (*None*) – The timeout, or None to wait forever

Returns A tuple of the data received and a tuple of the (address, port) received from

Return type str, (str, int)

Raises

- ***SpinnmanTimeoutException*** – If a timeout occurs before any data is received
- ***SpinnmanIOException*** – If an error occurs receiving the data

remote_ip_address
The remote ip address to which the connection is connected.

Returns The remote ip address as a dotted string, or None if not connected remotely

Return type str

remote_port
The remote port to which the connection is connected.

Returns The remote port, or None if not connected remotely

Return type int

send(*data*)

Send data down this connection

Parameters **data** (*str*) – The data to be sent

Raises *SpinnmanIOException* – If there is an error sending the data

send_to(*data*, *address*)

Send data down this connection

Parameters

- **data** (*str*) – The data to be sent as a bytestring
- **address** ((*str, int*)) – A tuple of (address, port) to send the data to

Raises *SpinnmanIOException* – If there is an error sending the data

spinnman.connections.udp_packet_connections.udp_listenable_connection module

class spinnman.connections.udp_packet_connections.udp_listenable_connection.**UDPListenableConn**

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.listenable.Listenable*

get_receive_method()

Get the method that receives for this connection

spinnman.connections.udp_packet_connections.utils module

spinnman.connections.udp_packet_connections.utils.**update_sdp_header_for_udp_send**(*sdp_header*, *source_x*, *source_y*)

Apply defaults to the sdp header for sending over UDP

Parameters **sdp_header** (*spinnman.messages.sdp.sdp_header.SDPHeader*) – The SDP header values

Returns Nothing is returned

Module contents

class `spinnman.connections.udp_packet_connections.`**`BMPConnection`**(*cabinet, frame, boards, local_host=None, local_port=None, remote_host=None, remote_port=None*)

Bases: `spinnman.connections.udp_packet_connections.udp_connection.UDPConnection`, `spinnman.connections.abstract_classes.scp_receiver.SCPReceiver`, `spinnman.connections.abstract_classes.scp_sender.SCPSender`

A BMP connection which supports queries to the BMP of a SpiNNaker machine

Parameters

- **cabinet** (*int*) – The cabinet number of the connection
- **frame** (*int*) – The frame number of the connection
- **boards** (*iterable of int*) – The boards that the connection can control on the same backplane
- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

boards

The set of boards supported by the BMP

Return type iterable of int

cabinet

The cabinet id of the BMP

Return type int

chip_x

Defined to satisfy the SCPSender - always 0 for a BMP

chip_y

Defined to satisfy the SCPSender - always 0 for a BMP

frame

The frame id of the BMP

Return type int

get_scp_data(*scp_request*)

receive_scp_response(*timeout=1.0*)

send_scp_request(*scp_request*)

class spinnman.connections.udp_packet_connections.**BootConnection**(*local_host=None, local_port=None, remote_host=None, remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.spinnaker_boot_sender.SpinnakerBootSender*, *spinnman.connections.abstract_classes.spinnaker_boot_receiver.SpinnakerBootReceiver*

A connection to the spinnaker board that uses UDP to for booting

Parameters

- **local_host** (*str*) – The local host name or ip address to bind to. If not specified defaults to bind to all interfaces, unless remote_host is specified, in which case binding is _done to the ip address that will be used to send packets
- **local_port** (*int*) – The local port to bind to, between 1025 and 65535. If not specified, defaults to a random unused local port
- **remote_host** (*str*) – The remote host name or ip address to send packets to. If not specified, the socket will be available for listening only, and will throw and exception if used for sending
- **remote_port** – The remote port to send packets to. If remote_host is None, this is ignored.

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error setting up the communication channel

receive_boot_message(*timeout=None*)

See *spinnman.connections.abstract_classes.spinnaker_boot_receiver.SpinnakerBootReceiver.receive_boot_message()*

send_boot_message(*boot_message*)

See *spinnman.connections.abstract_classes.spinnaker_boot_sender.SpinnakerBootSender.send_boot_message()*

class spinnman.connections.udp_packet_connections.**UDPConnection**(*local_host=None, local_port=None, remote_host=None, remote_port=None*)

Bases: *spinnman.connections.abstract_classes.connection.Connection*

Parameters

- **local_host** (*str or None*) – The local host name or ip address to bind to. If not specified defaults to bind to all interfaces, unless remote_host is specified, in which case binding is _done to the ip address that will be used to send packets
- **local_port** (*int*) – The local port to bind to, between 1025 and 65535. If not specified, defaults to a random unused local port
- **remote_host** (*str or None*) – The remote host name or ip address to send packets to. If not specified, the socket will be available for listening only, and will throw and exception if used for sending

- **remote_port** – The remote port to send packets to. If remote_host is None, this is ignored. If remote_host is specified, this must also be specified for the connection to allow sending

Raises *spinnman.exceptions.SpinnmanIOException* – If there is an error setting up the communication channel

close()

See *spinnman.connections.abstract_classes.connection.Connection.close()*

is_connected()

See *spinnman.connections.abstract_classes.connection.Connection.is_connected()*

is_ready_to_receive(*timeout=0*)

local_ip_address

The local IP address to which the connection is bound.

Returns The local ip address as a dotted string e.g. 0.0.0.0

Return type str

Raises **None** – No known exceptions are thrown

local_port

The local port to which the connection is bound.

Returns The local port number

Return type int

Raises **None** – No known exceptions are thrown

receive(*timeout=None*)

Receive data from the connection

Parameters **timeout** (*None or float*) – The timeout in seconds, or None to wait forever

Returns The data received as a bytestring

Return type str

Raises

- *SpinnmanTimeoutException* – If a timeout occurs before any data is received
- *SpinnmanIOException* – If an error occurs receiving the data

receive_with_address(*timeout=None*)

Receive data from the connection along with the address where the data was received from

Parameters **timeout** (*None*) – The timeout, or None to wait forever

Returns A tuple of the data received and a tuple of the (address, port) received from

Return type str, (str, int)

Raises

- *SpinnmanTimeoutException* – If a timeout occurs before any data is received
- *SpinnmanIOException* – If an error occurs receiving the data

remote_ip_address

The remote ip address to which the connection is connected.

Returns The remote ip address as a dotted string, or None if not connected remotely

Return type str

remote_port

The remote port to which the connection is connected.

Returns The remote port, or None if not connected remotely

Return type int

send(*data*)

Send data down this connection

Parameters **data** (*str*) – The data to be sent

Raises *SpinnmanIOException* – If there is an error sending the data

send_to(*data*, *address*)

Send data down this connection

Parameters

- **data** (*str*) – The data to be sent as a bytestring
- **address** ((*str, int*)) – A tuple of (address, port) to send the data to

Raises *SpinnmanIOException* – If there is an error sending the data

class spinnman.connections.udp_packet_connections.**EIEIOConnection**(*local_host=None*, *local_port=None*, *remote_host=None*, *remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.eieio_receiver.EIEIOReceiver*, *spinnman.connections.abstract_classes.eieio_sender.EIEIOSender*, *spinnman.connections.abstract_classes.listenable.Listenable*

A UDP connection for sending and receiving raw EIEIO messages

Parameters

- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

get_receive_method()

receive_eieio_message(*timeout=None*)

send_eieio_message(*eieio_message*)

send_eieio_message_to(*eieio_message*, *ip_address*, *port*)

class spinnman.connections.udp_packet_connections.**IPAddressesConnection**(*local_host=None*, *local_port=54321*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*

A connection that detects any UDP packet that is transmitted by spinnaker boards prior to boot

receive_ip_address(*timeout=None*)

supports_sends_message(*message*)

class spinnman.connections.udp_packet_connections.**UDPListenableConnection**(*local_host=None*, *local_port=None*, *remote_host=None*, *remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.listenable.Listenable*

get_receive_method()

Get the method that receives for this connection

class spinnman.connections.udp_packet_connections.**SCAMPConnection**(*chip_x=255*, *chip_y=255*, *local_host=None*, *local_port=None*, *remote_host=None*, *remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.sdp_connection.SDPConnection*, *spinnman.connections.abstract_classes.scp_sender.SCPSender*, *spinnman.connections.abstract_classes.scp_receiver.SCPReceiver*

A UDP connection to SCAMP on the board

Parameters

- **chip_x** (*int*) – The x-coordinate of the chip on the board with this remote_host
- **chip_y** (*int*) – The y-coordinate of the chip on the board with this remote_host
- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** (*int*) – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

chip_x

chip_y

get_scp_data(*scp_request*)

receive_scp_response(*timeout=1.0*)

send_scp_request(*scp_request*)

update_chip_coordinates(*x*, *y*)

class spinnman.connections.udp_packet_connections.**SDPConnection**(*chip_x=None*, *chip_y=None*, *local_host=None*, *local_port=None*, *remote_host=None*, *remote_port=None*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, *spinnman.connections.abstract_classes.sdp_receiver.SDPReceiver*, *spinnman.connections.abstract_classes.sdp_sender.SDPSender*, *spinnman.connections.abstract_classes.listenable.Listenable*

Parameters

- **chip_x** (*int*) – The optional x-coordinate of the chip at the remote end of the connection. If not specified, it will not be possible to send SDP messages that require a response with this connection.
- **chip_y** (*int*) – The optional y-coordinate of the chip at the remote end of the connection. If not specified, it will not be possible to send SDP messages that require a response with this connection.
- **local_host** (*str*) – The optional ip address or host name of the local interface to listen on
- **local_port** (*int*) – The optional local port to listen on
- **remote_host** (*str*) – The optional remote host name or ip address to send messages to. If not specified, sending will not be possible using this connection
- **remote_port** – The optional remote port number to send messages to. If not specified, sending will not be possible using this connection

get_receive_method()

receive_sdp_message(*timeout=None*)

send_sdp_message(*sdp_message*)

spinnman.connections.udp_packet_connections.**update_sdp_header_for_udp_send**(*sdp_header*, *source_x*, *source_y*)

Apply defaults to the sdp header for sending over UDP

Parameters **sdp_header** (*spinnman.messages.sdp.sdp_header.SDPHeader*) – The SDP header values

Returns Nothing is returned

Submodules

spinnman.connections.connection_listener module

class `spinnman.connections.connection_listener.`**`ConnectionListener`**(*connection*, *n_processes=4*)

Bases: `threading.Thread`

Listens to a connection and calls callbacks with new messages when they arrive

Parameters

- **`connection`** – An AbstractListenable connection to listen to
- **`n_processes`** – The number of threads to use when calling callbacks

`add_callback`(*callback*)

Add a callback to be called when a message is received

Parameters **`callback`** – A callable which takes a single parameter, which is the message received

`close`()

Closes the listener. Note that this does not close the provider of the messages; this instead marks the listener as closed. The listener will not truly stop until the get message call returns.

`run`()

spinnman.connections.scp_request_pipeline module

class `spinnman.connections.scp_request_pipeline.`**`SCPRequestPipeLine`**(*connection*, *n_channels=1*, *intermediate_channel_waits=0*, *retry_codes=set([<SCPResult.RC_P2 142>, <SCPResult.RC_TIMEOUT: 134>, <SCPResult.RC_P2P_NOREPLY: 139>, <SCPResult.RC_LEN: 129>])*, *n_retries=3*, *packet_timeout=0.5*)

Bases: `object`

Allows a set of SCP requests to be grouped together in a communication across a number of channels for a given connection.

This class implements an SCP windowing, first suggested by Andrew Mundy. This extends the idea by having both send and receive windows. These are represented by the n_channels and the intermediate_channel_waits parameters respectively. This seems to help with the timeout issue; when a timeout is received, all requests for which a reply has not been received can also timeout.

Parameters

- **connection** – The connection over which the communication is to take place
- **n_channels** – The number of requests to send before checking for responses. If None, this will be determined automatically
- **intermediate_channel_waits** – The number of outstanding responses to wait for before continuing sending requests. If None, this will be determined automatically
- **retry_codes** – The set of response codes that will be retried
- **n_retries** – The number of times to resend any packet for any reason before an error is triggered
- **packet_timeout** – The number of elapsed seconds after sending a packet before it is considered a timeout.

finish()

Indicate the end of the packets to be sent. This must be called to ensure that all responses are received and handled.

n_channels

n_resent

n_retry_code_resent

n_timeouts

send_request(*request*, *callback*, *error_callback*)

Add an SCP request to the set to be sent

Parameters

- **request** – The SCP request to be sent
- **callback** – A callback function to call when the response has been received; takes SCPResponse as a parameter, or None if the response doesn't need to be processed
- **error_callback** – A callback function to call when an error is found when processing the message; takes original SCPRequest, exception caught and a list of tuples of (filename, line number, function name, text) as a traceback

spinnman.connections.socket_address_with_chip module

class spinnman.connections.socket_address_with_chip.**SocketAddressWithChip**(*hostname*, *chip_x*, *chip_y*, *port_num=17893*)

Bases: object

The address of a socket and an associated chip

chip_x

The x-coordinate of the chip

Returns the x-coordinate

chip_y

The y-coordinate of the chip

Returns the y-coordinate

hostname

The hostname of the socket

Returns the hostname

port_num
The port number of the socket

Returns the port

spinnman.connections.token_bucket module

class spinnman.connections.token_bucket.**TokenBucket**(*tokens*, *fill_rate*)
Bases: object

An implementation of the token bucket algorithm.

```
>>> bucket = TokenBucket(80, 0.5)
>>> print bucket.consume(10)
True
Not thread safe.
```

Parameters

- **tokens** – the total tokens in the bucket
- **fill_rate** – the rate in tokens/second that the bucket will be refilled.

consume(*tokens*, *block=True*)
Consume tokens from the bucket. Returns True if there were sufficient tokens.

If there are not enough tokens and block is True, sleeps until the bucket is replenished enough to satisfy the deficiency.

If there are not enough tokens and block is False, returns False.

It is an error to consume more tokens than the bucket _capacity.

tokens

Module contents

class spinnman.connections.**ConnectionListener**(*connection*, *n_processes=4*)
Bases: threading.Thread

Listens to a connection and calls callbacks with new messages when they arrive

Parameters

- **connection** – An AbstractListenable connection to listen to
- **n_processes** – The number of threads to use when calling callbacks

add_callback(*callback*)
Add a callback to be called when a message is received

Parameters **callback** – A callable which takes a single parameter, which is the message received

close()
Closes the listener. Note that this does not close the provider of the messages; this instead marks the listener as closed. The listener will not truly stop until the get message call returns.

run()

class spinnman.connections.**SCPRequestPipeLine**(*connection*, *n_channels=1*, *intermediate_channel_waits=0*, *retry_codes=set([<SCPResult.RC_P2P_TIMEOUT: 142>, <SCPResult.RC_TIMEOUT: 134>, <SCPResult.RC_P2P_NOREPLY: 139>, <SCPResult.RC_LEN: 129>])*, *n_retries=3*, *packet_timeout=0.5*)

Bases: object

Allows a set of SCP requests to be grouped together in a communication across a number of channels for a given connection.

This class implements an SCP windowing, first suggested by Andrew Mundy. This extends the idea by having both send and receive windows. These are represented by the n_channels and the intermediate_channel_waits parameters respectively. This seems to help with the timeout issue; when a timeout is received, all requests for which a reply has not been received can also timeout.

Parameters

- **connection** – The connection over which the communication is to take place
- **n_channels** – The number of requests to send before checking for responses. If None, this will be determined automatically
- **intermediate_channel_waits** – The number of outstanding responses to wait for before continuing sending requests. If None, this will be determined automatically
- **retry_codes** – The set of response codes that will be retried
- **n_retries** – The number of times to resend any packet for any reason before an error is triggered
- **packet_timeout** – The number of elapsed seconds after sending a packet before it is considered a timeout.

finish()

Indicate the end of the packets to be sent. This must be called to ensure that all responses are received and handled.

n_channels

n_resent

n_retry_code_resent

n_timeouts

send_request(*request*, *callback*, *error_callback*)

Add an SCP request to the set to be sent

Parameters

- **request** – The SCP request to be sent
- **callback** – A callback function to call when the response has been received; takes SCPResponse as a parameter, or None if the response doesn't need to be processed
- **error_callback** – A callback function to call when an error is found when processing the message; takes original SCPRequest, exception caught and a list of tuples of (filename, line number, function name, text) as a traceback

class spinnman.connections.**SocketAddressWithChip**(*hostname*, *chip_x*, *chip_y*, *port_num=17893*)

Bases: object

The address of a socket and an associated chip

chip_x

The x-coordinate of the chip

Returns the x-coordinate

chip_y

The y-coordinate of the chip

Returns the y-coordinate

hostname

The hostname of the socket

Returns the hostname

port_num

The port number of the socket

Returns the port

class spinnman.connections.**TokenBucket**(*tokens*, *fill_rate*)

Bases: object

An implementation of the token bucket algorithm.

```
>>> bucket = TokenBucket(80, 0.5)
>>> print bucket.consume(10)
True
Not thread safe.
```

Parameters

- **tokens** – the total tokens in the bucket
- **fill_rate** – the rate in tokens/second that the bucket will be refilled.

consume(*tokens*, *block=True*)

Consume tokens from the bucket. Returns True if there were sufficient tokens.

If there are not enough tokens and block is True, sleeps until the bucket is replenished enough to satisfy the deficiency.

If there are not enough tokens and block is False, returns False.

It is an error to consume more tokens than the bucket _capacity.

tokens

spinnman.messages package

Subpackages

spinnman.messages.eieio package

Subpackages

spinnman.messages.eieio.abstract_messages package

Submodules

spinnman.messages.eieio.abstract_messages.eieio_message module

class spinnman.messages.eieio.abstract_messages.eieio_message.**AbstractEIEIOMessage**
Bases: object

Marker interface for an EIEIOMessage

Module contents

class spinnman.messages.eieio.abstract_messages.**AbstractEIEIOMessage**
Bases: object

Marker interface for an EIEIOMessage

spinnman.messages.eieio.command_messages package

Submodules

spinnman.messages.eieio.command_messages.database_confirmation module

class spinnman.messages.eieio.command_messages.database_confirmation.**DatabaseConfirmation**(*data*
Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet which contains the path to the database created by the toolchain which is to be used by any software which interfaces with SpiNNaker

bytestring

database_path

static from_bytestring(*command_header*, *data*, *offset*)

spinnman.messages.eieio.command_messages.eieio_command_header module

class spinnman.messages.eieio.command_messages.eieio_command_header.**EIEIOCommandHeader**(*comman*
Bases: object

EIEIO header for command packets

bytestring
Get a bytestring of the header

Return type str

command

static **from_bytestring**(*data*, *offset*)
Read an eieio command header from a bytestring

Parameters

- **data** (*str*) – The bytestring to read the data from
- **offset** (*int*) – The offset where the valid data starts

Returns an EIEIO command header

Return type *spinnman.messages.eieio.command_messages.eieio_command_header.EIEIOCommandHeader*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error reading from the reader
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If there is an error setting any of the values

spinnman.messages.eieio.command_messages.eieio_command_message module

class spinnman.messages.eieio.command_messages.eieio_command_message.**EIEIOCommandMessage**(*eieio_data=, offset=0*)

Bases: *spinnman.messages.eieio.abstract_messages.eieio_message.AbstractEIEIOMessage*

An EIEIO command message

Parameters

- **eieio_command_header** (*spinnman.messages.eieio.command_messages.eieio_command_header.EIEIOCommandHeader*) – The header of the message
- **data** (*str*) – Optional incoming data
- **offset** (*int*) – Offset into the data where valid data begins

bytestring

data

eieio_header

static **from_bytestring**(*command_header*, *data*, *offset*)

static **get_min_packet_length**()

offset

spinnman.messages.eieio.command_messages.event_stop_request module

class spinnman.messages.eieio.command_messages.event_stop_request.**EventStopRequest**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used for the buffering input technique which causes the parser of the input packet to terminate its execution

spinnman.messages.eieio.command_messages.host_data_read module

class spinnman.messages.eieio.command_messages.host_data_read.**HostDataRead**(*n_requests*, *sequence_no*, *channel*, *region_id*, *space_read*)

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet sent by the host computer to the SpiNNaker system in the context of the buffering output technique to signal that the host has completed reading data from the output buffer, and that such space can be considered free to use again

bytestring

channel(*ack_id*)

static from_bytestring(*command_header*, *data*, *offset*)

static get_min_packet_length()

n_requests

region_id(*ack_id*)

sequence_no

space_read(*ack_id*)

spinnman.messages.eieio.command_messages.host_data_read_ack module

class spinnman.messages.eieio.command_messages.host_data_read_ack.**HostDataReadAck**(*sequence_no*)

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet sent by the host computer to the SpiNNaker system in the context of the buffering output technique to signal that the host has received a request to read data

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

sequence_no

spinnman.messages.eieio.command_messages.host_send_sequenced_data module

class spinnman.messages.eieio.command_messages.host_send_sequenced_data.**HostSendSequencedData**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet sent from the host to the SpiNNaker system in the context of buffering input mechanism to identify packet which needs to be stored in memory for future use

bytestring

eieio_data_message

static from_bytestring(*command_header*, *data*, *offset*)

static get_min_packet_length()

region_id

sequence_no

spinnman.messages.eieio.command_messages.notification_protocol_pause_stop module

class spinnman.messages.eieio.command_messages.notification_protocol_pause_stop.**NotificationP**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet which indicates that the toolchain has paused or stopped

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

spinnman.messages.eieio.command_messages.notification_protocol_start_resume module

class spinnman.messages.eieio.command_messages.notification_protocol_start_resume.**Notificatio**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet which indicates that the toolchain has started or resumed

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

spinnman.messages.eieio.command_messages.padding_request module

class spinnman.messages.eieio.command_messages.padding_request.**PaddingRequest**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used to pad space in the buffering area, if needed

static get_min_packet_length()

spinnman.messages.eieio.command_messages.spinnaker_request_buffers module

class spinnman.messages.eieio.command_messages.spinnaker_request_buffers.**SpinnakerRequestBuff**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Message used in the context of the buffering input mechanism which is sent by the SpiNNaker system to the host computer to ask for more data to inject during the simulation

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

static get_min_packet_length()

p

region_id

sequence_no

space_available

x

y

spinnman.messages.eieio.command_messages.spinnaker_request_read_data module

class spinnman.messages.eieio.command_messages.spinnaker_request_read_data.**SpinnakerRequestRe**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Message used in the context of the buffering output mechanism which is sent from the SpiNNaker system to the host computer to signal that some data is available to be read

bytestring

channel(*request_id*)

static from_bytestring(*command_header*, *data*, *offset*)

static **get_min_packet_length**()

n_requests

p

region_id(*request_id*)

sequence_no

space_to_be_read(*request_id*)

start_address(*request_id*)

x

y

spinnman.messages.eieio.command_messages.start_requests module

class spinnman.messages.eieio.command_messages.start_requests.**StartRequests**
Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used in the context of buffering input for the host computer to signal to the SpiNNaker system that, if needed, it is possible to send more "SpinnakerRequestBuffers" packet

spinnman.messages.eieio.command_messages.stop_requests module

class spinnman.messages.eieio.command_messages.stop_requests.**StopRequests**
Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used in the context of buffering input for the host computer to signal to the SpiNNaker system that to stop sending "SpinnakerRequestBuffers" packet

Module contents

class spinnman.messages.eieio.command_messages.**DatabaseConfirmation**(*database_path=None*)
Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet which contains the path to the database created by the toolchain which is to be used by any software which interfaces with SpiNNaker

bytestring

database_path

static **from_bytestring**(*command_header*, *data*, *offset*)

class spinnman.messages.eieio.command_messages.**EIEIOCommandHeader**(*command*)
Bases: object

EIEIO header for command packets

bytestring
Get a bytestring of the header

Return type str

command

static **from_bytestring**(*data*, *offset*)

Read an eieio command header from a bytestring

Parameters

- **data** (*str*) – The bytestring to read the data from
- **offset** (*int*) – The offset where the valid data starts

Returns an EIEIO command header

Return type *spinnman.messages.eieio.command_messages.eieio_command_header.EIEIOCommandHeader*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error reading from the reader
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If there is an error setting any of the values

class spinnman.messages.eieio.command_messages.**EIEIOCommandMessage**(*eieio_command_header*, *data=None*, *offset=0*)

Bases: *spinnman.messages.eieio.abstract_messages.eieio_message.AbstractEIEIOMessage*

An EIEIO command message

Parameters

- **eieio_command_header** (*spinnman.messages.eieio.command_messages.eieio_command_header.EIEIOCommandHeader*) – The header of the message
- **data** (*str*) – Optional incoming data
- **offset** (*int*) – Offset into the data where valid data begins

bytestring

data

eieio_header

static **from_bytestring**(*command_header*, *data*, *offset*)

static **get_min_packet_length**()

offset

class spinnman.messages.eieio.command_messages.**EventStopRequest**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used for the buffering input technique which causes the parser of the input packet to terminate its execution

class spinnman.messages.eieio.command_messages.**HostDataRead**(*n_requests*, *sequence_no*, *channel*, *region_id*, *space_read*)

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet sent by the host computer to the SpiNNaker system in the context of the buffering output technique to signal that the host has completed reading data from the output buffer, and that such space can be considered free to use again

bytestring

channel(*ack_id*)

static from_bytestring(*command_header*, *data*, *offset*)

static get_min_packet_length()

n_requests

region_id(*ack_id*)

sequence_no

space_read(*ack_id*)

class spinnman.messages.eieio.command_messages.**HostSendSequencedData**(*region_id*, *sequence_no*, *eieio_data_message*)

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet sent from the host to the SpiNNaker system in the context of buffering input mechanism to identify packet which needs to be stored in memory for future use

bytestring

eieio_data_message

static from_bytestring(*command_header*, *data*, *offset*)

static get_min_packet_length()

region_id

sequence_no

class spinnman.messages.eieio.command_messages.**NotificationProtocolPauseStop**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet which indicates that the toolchain has paused or stopped

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

class spinnman.messages.eieio.command_messages.**NotificationProtocolStartResume**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet which indicates that the toolchain has started or resumed

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

class spinnman.messages.eieio.command_messages.**PaddingRequest**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used to pad space in the buffering area, if needed

static get_min_packet_length()

class spinnman.messages.eieio.command_messages.**SpinnakerRequestBuffers**(*x*, *y*, *p*, *region_id*, *sequence_no*, *space_available*)

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Message used in the context of the buffering input mechanism which is sent by the SpiNNaker system to the host computer to ask for more data to inject during the simulation

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

static get_min_packet_length()

p

region_id

sequence_no

space_available

x

y

class spinnman.messages.eieio.command_messages.**HostDataReadAck**(*sequence_no*)

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet sent by the host computer to the SpiNNaker system in the context of the buffering output technique to signal that the host has received a request to read data

bytestring

static from_bytestring(*command_header*, *data*, *offset*)

sequence_no

class spinnman.messages.eieio.command_messages.**SpinnakerRequestReadData**(*x*, *y*, *p*, *region_id*, *sequence_no*, *n_requests*, *channel*, *start_address*, *space_to_be_read*)

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Message used in the context of the buffering output mechanism which is sent from the SpiNNaker system to the host computer to signal that some data is available to be read

bytestring

channel(*request_id*)

static **from_bytestring**(*command_header*, *data*, *offset*)

static **get_min_packet_length**()

n_requests

p

region_id(*request_id*)

sequence_no

space_to_be_read(*request_id*)

start_address(*request_id*)

x

y

class spinnman.messages.eieio.command_messages.**StartRequests**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used in the context of buffering input for the host computer to signal to the SpiNNaker system that, if needed, it is possible to send more "SpinnakerRequestBuffers" packet

class spinnman.messages.eieio.command_messages.**StopRequests**

Bases: *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

Packet used in the context of buffering input for the host computer to signal to the SpiNNaker system that to stop sending "SpinnakerRequestBuffers" packet

spinnman.messages.eieio.data_messages package

Submodules

spinnman.messages.eieio.data_messages.abstract_data_element module

class spinnman.messages.eieio.data_messages.abstract_data_element.**AbstractDataElement**

Bases: object

A marker interface for possible data elements in the EIEIO data packet

get_bytestring(*eieio_type*)

Get a bytestring for the given type

Parameters **eieio_type** (*spinnman.messages.eieio.eieio_type.EIEIOType*) – The type of the message being written

Returns A bytestring for the element

Return type str

Raises ***SpinnmanInvalidParameterException*** – If the type is incompatible with the element

spinnman.messages.eieio.data_messages.eieio_data_header module

class spinnman.messages.eieio.data_messages.eieio_data_header.**EIEIODataHeader**(*eieio_type*, *tag=0*, *prefix=None*, *prefix_type=<EIEIOPref 0>*, *payload_base=None*, *is_time=False*, *count=0*)

Bases: object

EIEIO header for data packets

Parameters

- **eieio_type** (spinnman.spinnman.messages.eieio.eieio_type.EIEIOType) – the type of message
- **tag** (*int*) – the tag of the message (0 by default)
- **prefix** (*int or None*) – the key prefix of the message or None if not prefixed
- **prefix_type** (*spinnman.messages.eieio.eieio_prefix.EIEIOPrefix*) – the position of the prefix (upper or lower)
- **payload_base** (*int or None*) – The base payload to be applied, or None if no base payload
- **is_time** (*bool*) – True if the payloads should be taken to be timestamps, or False otherwise
- **count** (*int*) – Count of the number of items in the packet

bytestring

Get a bytestring of the header

Returns The header as a bytestring

Return type str

count

eieio_type

static from_bytestring(*data*, *offset*)

Read an eieio data header from a bytestring

Parameters

- **data** (*str*) – The bytestring to be read
- **offset** (*int*) – The offset at which the data starts

Returns an EIEIO header

Return type *spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader*

static get_header_size(*eieio_type*, *is_prefix=False*, *is_payload_base=False*)

Get the size of a header with the given parameters

Parameters

- **eieio_type** (spinnman.spinnman.messages.eieio.eieio_type.EIEIOType) – the type of message
- **is_prefix** (*bool*) – True if there is a prefix, False otherwise
- **is_payload_base** (*bool*) – True if there is a payload base, False otherwise

Returns The size of the header in bytes

Return type int

increment_count()

is_time

payload_base

prefix

prefix_type

reset_count()

size

tag

spinnman.messages.eieio.data_messages.eieio_data_message module

class spinnman.messages.eieio.data_messages.eieio_data_message.**EIEIODataMessage**(*eieio_header*, *data=None*, *offset=0*)

Bases: *spinnman.messages.eieio.abstract_messages.eieio_message.AbstractEIEIOMessage*

An EIEIO Data message

Parameters

- **eieio_header** (*spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader*) – The header of the message
- **data** (*str*) – Optional data contained within the packet
- **offset** (*int*) – Optional offset where the valid data starts

add_element(*element*)

Add an element to the message. The correct type of element must be added, depending on the header values

Parameters **element** (spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement) – The element to be added

Raises

- ***SpinnmanInvalidParameterException*** – If the element is not compatible with the header
- ***SpinnmanInvalidPacketException*** – If the message was created to read data

add_key(*key*)
Add a key to the packet

Parameters **key** (*int*) – The key to add

Raises ***SpinnmanInvalidParameterException*** – If the key is too big for the format, or the format expects a payload

add_key_and_payload(*key*, *payload*)
Adds a key and payload to the packet

Parameters

- **key** (*int*) – The key to add
- **payload** (*int*) – The payload to add

Raises ***SpinnmanInvalidParameterException*** – If the key or payload is too big for the format, or the format doesn't expect a payload

bytestring

static create(*eieio_type*, *count=0*, *data=None*, *offset=0*, *key_prefix=None*, *payload_prefix=None*, *timestamp=None*, *prefix_type=<EIEIOPrefix.LOWER_HALF_WORD: 0>*)
Create a data message

Parameters

- **eieio_type** – The EIEIOType of the message
- **count** – The number of items in the message
- **data** – The data in the message
- **offset** – The offset in the data where the actual data starts
- **key_prefix** – The prefix of the keys
- **payload_prefix** – The prefix of the payload
- **timestamp** – The timestamp of the packet
- **prefix_type** – The type of the key prefix if 16-bits

eieio_header

get_min_packet_length()
Get the minimum length of a message instance in bytes

Return type int

is_next_element
Determine if there is another element to be read

Returns True if the message was created with data, and there are more elements to be read

Return type bool

max_n_elements
The maximum number of elements that can fit in the packet

Return type int

static min_packet_length(*eieio_type*, *is_prefix=False*, *is_payload_base=False*, *is_timestamp=False*)
The minimum length of a message with the given header, in bytes

Parameters

- **eieio_type** (spinnman.spinnman.messages.eieio.eieio_type.EIEIOType) – the type of message
- **is_prefix** (*bool*) – True if there is a prefix, False otherwise
- **is_payload_base** (*bool*) – True if there is a payload base, False otherwise
- **is_timestamp** – True if there is a timestamp, False otherwise

Returns The minimum size of the packet in bytes

Return type int

n_elements
The number of elements in the packet

next_element
The next element to be read, or None if no more elements. The exact type of element returned depends on the packet type

Return type spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement

size
The size of the packet with the current contents

spinnman.messages.eieio.data_messages.key_data_element module

class spinnman.messages.eieio.data_messages.key_data_element.**KeyDataElement**(*key*)

Bases: *spinnman.messages.eieio.data_messages.abstract_data_element.AbstractDataElement*

A data element that contains just a key

get_bytestring(*eieio_type*)

key

spinnman.messages.eieio.data_messages.key_payload_data_element module

class spinnman.messages.eieio.data_messages.key_payload_data_element.**KeyPayloadDataElement**(*key*, *pa*, *lo*, *pa*, *lo*

Bases: *spinnman.messages.eieio.data_messages.abstract_data_element.AbstractDataElement*

A data element that contains a key and a payload

get_bytestring(*eieio_type*)

key

payload

payload_is_timestamp

Module contents

class spinnman.messages.eieio.data_messages.**AbstractDataElement**

Bases: object

A marker interface for possible data elements in the EIEIO data packet

get_bytestring(*eieio_type*)

Get a bytestring for the given type

Parameters **eieio_type** (*spinnman.messages.eieio.eieio_type.EIEIOType*) – The type of the message being written

Returns A bytestring for the element

Return type str

Raises ***SpinnmanInvalidParameterException*** – If the type is incompatible with the element

class spinnman.messages.eieio.data_messages.**EIEIODataHeader**(*eieio_type, tag=0, prefix=None, prefix_type=<EIEIOPrefix.LOWER_HALF_WORD: 0>, payload_base=None, is_time=False, count=0*)

Bases: object

EIEIO header for data packets

Parameters

- **eieio_type** (spinnman.spinnman.messages.eieio.eieio_type.EIEIOType) – the type of message
- **tag** (*int*) – the tag of the message (0 by default)
- **prefix** (*int or None*) – the key prefix of the message or None if not prefixed
- **prefix_type** (*spinnman.messages.eieio.eieio_prefix.EIEIOPrefix*) – the position of the prefix (upper or lower)
- **payload_base** (*int or None*) – The base payload to be applied, or None if no base payload
- **is_time** (*bool*) – True if the payloads should be taken to be timestamps, or False otherwise
- **count** (*int*) – Count of the number of items in the packet

bytestring

Get a bytestring of the header

Returns The header as a bytestring

Return type str

count

eieio_type

static from_bytestring(*data, offset*)

Read an eieio data header from a bytestring

Parameters

- **data** (*str*) – The bytestring to be read
- **offset** (*int*) – The offset at which the data starts

Returns an EIEIO header

Return type *spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader*

static get_header_size(*eieio_type*, *is_prefix=False*, *is_payload_base=False*)

Get the size of a header with the given parameters

Parameters

- **eieio_type** (spinnman.spinnman.messages.eieio.eieio_type.EIEIOType) – the type of message
- **is_prefix** (*bool*) – True if there is a prefix, False otherwise
- **is_payload_base** (*bool*) – True if there is a payload base, False otherwise

Returns The size of the header in bytes

Return type int

increment_count()

is_time

payload_base

prefix

prefix_type

reset_count()

size

tag

class spinnman.messages.eieio.data_messages.**EIEIODataMessage**(*eieio_header*, *data=None*, *offset=0*)

Bases: *spinnman.messages.eieio.abstract_messages.eieio_message.AbstractEIEIOMessage*

An EIEIO Data message

Parameters

- **eieio_header** (*spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader*) – The header of the message
- **data** (*str*) – Optional data contained within the packet
- **offset** (*int*) – Optional offset where the valid data starts

add_element(*element*)

Add an element to the message. The correct type of element must be added, depending on the header values

Parameters **element** (spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement) – The element to be added

Raises

- **SpinnmanInvalidParameterException** – If the element is not compatible with the header
- **SpinnmanInvalidPacketException** – If the message was created to read data

add_key(*key*)

Add a key to the packet

Parameters **key** (*int*) – The key to add

Raises **SpinnmanInvalidParameterException** – If the key is too big for the format, or the format expects a payload

add_key_and_payload(*key*, *payload*)

Adds a key and payload to the packet

Parameters

- **key** (*int*) – The key to add
- **payload** (*int*) – The payload to add

Raises **SpinnmanInvalidParameterException** – If the key or payload is too big for the format, or the format doesn't expect a payload

bytestring

static create(*eieio_type*, *count=0*, *data=None*, *offset=0*, *key_prefix=None*, *payload_prefix=None*, *timestamp=None*, *prefix_type=<EIEIOPrefix.LOWER_HALF_WORD: 0>*)

Create a data message

Parameters

- **eieio_type** – The EIEIOType of the message
- **count** – The number of items in the message
- **data** – The data in the message
- **offset** – The offset in the data where the actual data starts
- **key_prefix** – The prefix of the keys
- **payload_prefix** – The prefix of the payload
- **timestamp** – The timestamp of the packet
- **prefix_type** – The type of the key prefix if 16-bits

eieio_header

get_min_packet_length()

Get the minimum length of a message instance in bytes

Return type int

is_next_element

Determine if there is another element to be read

Returns True if the message was created with data, and there are more elements to be read

Return type bool

max_n_elements

The maximum number of elements that can fit in the packet

Return type int

static **min_packet_length**(*eieio_type*, *is_prefix=False*, *is_payload_base=False*, *is_timestamp=False*)

The minimum length of a message with the given header, in bytes

Parameters

- **eieio_type** (spinnman.spinnman.messages.eieio.eieio_type.EIEIOType) – the type of message
- **is_prefix** (*bool*) – True if there is a prefix, False otherwise
- **is_payload_base** (*bool*) – True if there is a payload base, False otherwise
- **is_timestamp** – True if there is a timestamp, False otherwise

Returns The minimum size of the packet in bytes

Return type int

n_elements

The number of elements in the packet

next_element

The next element to be read, or None if no more elements. The exact type of element returned depends on the packet type

Return type spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement

size

The size of the packet with the current contents

class spinnman.messages.eieio.data_messages.**KeyDataElement**(*key*)

Bases: *spinnman.messages.eieio.data_messages.abstract_data_element.AbstractDataElement*

A data element that contains just a key

get_bytestring(*eieio_type*)

key

class spinnman.messages.eieio.data_messages.**KeyPayloadDataElement**(*key*, *payload*, *payload_is_timestamp=False*)

Bases: *spinnman.messages.eieio.data_messages.abstract_data_element.AbstractDataElement*

A data element that contains a key and a payload

get_bytestring(*eieio_type*)

key

payload

payload_is_timestamp

Submodules

spinnman.messages.eieio.create_eieio_command module

spinnman.messages.eieio.create_eieio_command.**read_eieio_command_message**(*data*, *offset*)

Reads the content of an EIEIO command message and returns an object identifying the command which was contained in the packet, including any parameter, if required by the command

Parameters

- **data** (*str*) – data received from the network as a bytestring
- **offset** (*int*) – offset at which the parsing operation should start

Returns an object which inherits from EIEIOCommandMessage which contains parsed data received from the network

Return type *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

spinnman.messages.eieio.create_eieio_data module

spinnman.messages.eieio.create_eieio_data.**read_eieio_data_message**(*data*, *offset*)

Reads the content of an EIEIO data message and returns an object identifying the data which was contained in the packet

Parameters

- **data** (*str*) – data received from the network as a bytestring
- **offset** (*int*) – offset at which the parsing operation should start

Returns an object which inherits from EIEIODataMessage which contains parsed data received from the network

Return type *spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage*

spinnman.messages.eieio.eieio_prefix module

class spinnman.messages.eieio.eieio_prefix.**EIEIOPrefix**(*value*, *doc=''*)

Bases: enum.Enum

Possible prefixing of keys in EIEIO packets

LOWER_HALF_WORD = 0

UPPER_HALF_WORD = 1

spinnman.messages.eieio.eieio_type module

class spinnman.messages.eieio.eieio_type.**EIEIOType**(*value*, *key_bytes*, *payload_bytes*, *doc=''*)

Bases: enum.Enum

Possible types of EIEIO packets

KEY_16_BIT = 0

KEY_32_BIT = 2

KEY_PAYLOAD_16_BIT = 1

KEY_PAYLOAD_32_BIT = 3

key_bytes
The number of bytes used by each key element

Return type int

max_value
The maximum value of the key or payload (if there is a payload)

Return type int

payload_bytes
The number of bytes used by each payload element

Return type int

Module contents

class `spinnman.messages.eieio.`**`EIEIOPrefix`**(*value*, *doc=''*)
Bases: `enum.Enum`

Possible prefixing of keys in EIEIO packets

LOWER_HALF_WORD = 0

UPPER_HALF_WORD = 1

class `spinnman.messages.eieio.`**`EIEIOType`**(*value*, *key_bytes*, *payload_bytes*, *doc=''*)
Bases: `enum.Enum`

Possible types of EIEIO packets

KEY_16_BIT = 0

KEY_32_BIT = 2

KEY_PAYLOAD_16_BIT = 1

KEY_PAYLOAD_32_BIT = 3

key_bytes
The number of bytes used by each key element

Return type int

max_value
The maximum value of the key or payload (if there is a payload)

Return type int

payload_bytes
The number of bytes used by each payload element

Return type int

spinnman.messages.eieio.**read_eieio_command_message**(*data*, *offset*)

Reads the content of an EIEIO command message and returns an object identifying the command which was contained in the packet, including any parameter, if required by the command

Parameters

- **data** (*str*) – data received from the network as a bytestring
- **offset** (*int*) – offset at which the parsing operation should start

Returns an object which inherits from EIEIOCommandMessage which contains parsed data received from the network

Return type *spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*

spinnman.messages.eieio.**read_eieio_data_message**(*data*, *offset*)

Reads the content of an EIEIO data message and returns an object identifying the data which was contained in the packet

Parameters

- **data** (*str*) – data received from the network as a bytestring
- **offset** (*int*) – offset at which the parsing operation should start

Returns an object which inherits from EIEIODataMessage which contains parsed data received from the network

Return type *spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage*

spinnman.messages.scp package

Subpackages

spinnman.messages.scp.abstract_messages package

Submodules

spinnman.messages.scp.abstract_messages.bmp_request module

BMPRequest

class spinnman.messages.scp.abstract_messages.bmp_request.**BMPRequest**(*boards*, *scp_request_header*, *argument_1=None*, *argument_2=None*, *argument_3=None*, *data=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request intended to be sent to a BMP

Parameters

- **boards** (*int or iterable of int*) – The board or boards to be addressed by this request
- **scp_request_header** – The SCP request header
- **argument_1** – The optional first argument
- **argument_2** – The optional second argument
- **argument_3** – The optional third argument
- **data** – The optional data to be sent

static get_board_mask(*boards*)
Get the board mask given an int or iterable of ints of board ids

static get_first_board(*boards*)
Get the first board id given an int or iterable of ints

spinnman.messages.scp.abstract_messages.bmp_response module

BMPResponse

class spinnman.messages.scp.abstract_messages.bmp_response.**BMPResponse**
Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

represents a scp request thats tialored for the bmp connection

spinnman.messages.scp.abstract_messages.scp_request module

class spinnman.messages.scp.abstract_messages.scp_request.**AbstractSCPRequest**(*sdp_header, scp_request_header, argument_1=None, argument_2=None, argument_3=None, data=None*)
Bases: object

Represents an Abstract SCP Request

Parameters

- **sdp_header** (*spinnman.messages.sdp.sdp_header.SDPHeader*) – The SDP header of the request
- **scp_request_header** (*spinnman.messages.scp.scp_request_header.SCPRequestHeader*) – The SCP header of the request
- **argument_1** (*int*) – The first argument, or None if no first argument
- **argument_2** (*int*) – The second argument, or None if no second argument
- **argument_3** (*int*) – The third argument, or None if no third argument

- **data** (*bytearray*) – The optional data, or None if no data

Raises **None** – No known exceptions are raised

DEFAULT_DEST_X_COORD = 255

DEFAULT_DEST_Y_COORD = 255

argument_1
The first argument, or None if no first argument

Return type int

argument_2
The second argument, or None if no second argument

Return type int

argument_3
The third argument, or None if no third argument

Return type int

bytestring
The request as a bytestring

Returns The request as a bytestring

Return type str

data
The data, or None if no data

Return type bytearray

get_scp_response()
Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type `spinnman.messages.scp_response.SCPResponse`

Raises **None** – No known exceptions are raised

scp_request_header
The SCP request header of the message

Return type *spinnman.messages.scp.scp_request_header.SCPRequestHeader*

sdp_header
The SDP header of the message

Return type `spinnman.message.sdp.sdp_header.SDPHeader`

spinnman.messages.scp.abstract_messages.scp_response module

class `spinnman.messages.scp.abstract_messages.scp_response.`**AbstractSCPResponse**
Bases: `object`

Represents an abstract SCP Response

read_bytestring(*data*, *offset*)
Reads a packet from a bytestring of data

Parameters

- **data** (*str*) – The bytestring to be read
- **offset** (*int*) – The offset in the data from which the response should be read

read_data_bytestring(*data*, *offset*)
Reads the remainder of the data following the header

Parameters

- **data** (*str*) – The bytestring to read from
- **offset** (*int*) – The offset into the data after the headers

scp_response_header
The SCP header from the response

Returns The SCP header

Return type *spinnman.messages.scp.scp_response_header.SCPResponseHeader*

Raises **None** – No known exceptions are raised

sdp_header
The SDP header from the response

Returns The SDP header

Return type *spinnman.messages.sdp.sdp_header.SDPHeader*

Raises **None** – No known exceptions are raised

Module contents

class spinnman.messages.scp.abstract_messages.**BMPRequest**(*boards*, *scp_request_header*, *argument_1=None*, *argument_2=None*, *argument_3=None*, *data=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request intended to be sent to a BMP

Parameters

- **boards** (*int or iterable of int*) – The board or boards to be addressed by this request
- **scp_request_header** – The SCP request header
- **argument_1** – The optional first argument
- **argument_2** – The optional second argument
- **argument_3** – The optional third argument
- **data** – The optional data to be sent

static get_board_mask(*boards*)
Get the board mask given an int or iterable of ints of board ids

static get_first_board(*boards*)
Get the first board id given an int or iterable of ints

class spinnman.messages.scp.abstract_messages.**BMPResponse**

Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

represents a scp request thats tialored for the bmp connection

class spinnman.messages.scp.abstract_messages.**AbstractSCPRequest**(*sdp_header*, *scp_request_header*, *argument_1=None*, *argument_2=None*, *argument_3=None*, *data=None*)

Bases: object

Represents an Abstract SCP Request

Parameters

- **sdp_header** (*spinnman.messages.sdp.sdp_header.SDPHeader*) – The SDP header of the request
- **scp_request_header** (*spinnman.messages.scp.scp_request_header.SCPRequestHeader*) – The SCP header of the request
- **argument_1** (*int*) – The first argument, or None if no first argument
- **argument_2** (*int*) – The second argument, or None if no second argument
- **argument_3** (*int*) – The third argument, or None if no third argument
- **data** (*bytearray*) – The optional data, or None if no data

Raises **None** – No known exceptions are raised

DEFAULT_DEST_X_COORD = 255

DEFAULT_DEST_Y_COORD = 255

argument_1

The first argument, or None if no first argument

Return type int

argument_2

The second argument, or None if no second argument

Return type int

argument_3

The third argument, or None if no third argument

Return type int

bytestring

The request as a bytestring

Returns The request as a bytestring

Return type str

data

The data, or None if no data

Return type bytearray

get_scp_response()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type `spinnman.messages.scp_response.SCPResponse`

Raises **None** – No known exceptions are raised

scp_request_header

The SCP request header of the message

Return type *spinnman.messages.scp.scp_request_header.SCPRequestHeader*

sdp_header

The SDP header of the message

Return type `spinnman.message.sdp.sdp_header.SDPHeader`

class `spinnman.messages.scp.abstract_messages.`**AbstractSCPResponse**

Bases: `object`

Represents an abstract SCP Response

read_bytestring(*data*, *offset*)

Reads a packet from a bytestring of data

Parameters

- **data** (*str*) – The bytestring to be read
- **offset** (*int*) – The offset in the data from which the response should be read

read_data_bytestring(*data*, *offset*)

Reads the remainder of the data following the header

Parameters

- **data** (*str*) – The bytestring to read from
- **offset** (*int*) – The offset into the data after the headers

scp_response_header

The SCP header from the response

Returns The SCP header

Return type *spinnman.messages.scp.scp_response_header.SCPResponseHeader*

Raises **None** – No known exceptions are raised

sdp_header

The SDP header from the response

Returns The SDP header

Return type *spinnman.messages.sdp.sdp_header.SDPHeader*

Raises **None** – No known exceptions are raised

spinnman.messages.scp.enums package

Submodules

spinnman.messages.scp.enums.alloc_free module

class spinnman.messages.scp.enums.alloc_free.**AllocFree**(*value*, *doc=''*)

Bases: enum.Enum

The SCP Allocation and Free codes

ALLOC_ROUTING = 3

ALLOC_SDRAM = 0

FREE_ROUTING_BY_APP_ID = 5

FREE_ROUTING_BY_POINTER = 4

FREE_SDRAM_BY_APP_ID = 2

FREE_SDRAM_BY_POINTER = 1

spinnman.messages.scp.enums.bmp_info module

class spinnman.messages.scp.enums.bmp_info.**BMPInfo**(*value*, *doc=''*)

Bases: enum.Enum

The SCP BMP Information Types

ADC = 3

CAN_STATUS = 2

IP_ADDR = 4

SERIAL = 0

spinnman.messages.scp.enums.iptag_command module

class spinnman.messages.scp.enums.iptag_command.**IPTagCommand**(*value*, *doc=''*)

Bases: enum.Enum

SCP IPTag Commands

CLR = 3

GET = 2

NEW = 0

SET = 1

TTO = 4

spinnman.messages.scp.enums.led_action module

class spinnman.messages.scp.enums.led_action.**LEDAction**(*value*, *doc=''*)

Bases: enum.Enum

The SCP LED actions

OFF = 2

ON = 3

TOGGLE = 1

spinnman.messages.scp.enums.power_command module

class spinnman.messages.scp.enums.power_command.**PowerCommand**(*value*, *doc=''*)

Bases: enum.Enum

The SCP Power Commands

POWER_OFF = 0

POWER_ON = 1

spinnman.messages.scp.enums.scp_command module

class spinnman.messages.scp.enums.scp_command.**SCPCommand**(*value*, *doc=''*)

Bases: enum.Enum

The SCP Commands

CMD_ALLOC = 28

CMD_APLX = 4

CMD_AR = 19

CMD_AS = 24

CMD_BMP_INFO = 48

CMD_BMP_POWER = 57

CMD_DPRI = 30

CMD_FFD = 23

CMD_FILL = 5

CMD_FLASH_COPY = 49

CMD_FLASH_ERASE = 50

CMD_FLASH_WRITE = 51

CMD_INFO = 31

CMD_IPTAG = 26

CMD_LED = 25

CMD_LINK_READ = 17

CMD_LINK_WRITE = 18

CMD_NNP = 20

CMD_P2PC = 21

CMD_READ = 2

CMD_REMAP = 16

CMD_RESET = 55

CMD_RTR = 29

CMD_RUN = 1

CMD_SIG = 22

CMD_SROM = 27

CMD_TUBE = 64

CMD_VER = 0

CMD_WRITE = 3

spinnman.messages.scp.enums.scp_result module

class spinnman.messages.scp.enums.scp_result.**SCPResult**(*value*, *doc=''*)

Bases: enum.Enum

The SCP Result codes

RC_ARG = 132

RC_BUF = 138

RC_CMD = 131

RC_CPU = 136

RC_DEAD = 137

RC_LEN = 129

RC_OK = 128

RC_P2P_BUSY = 141

RC_P2P_NOREPLY = 139

RC_P2P_REJECT = 140

RC_P2P_TIMEOUT = 142

RC_PKT_TX = 143

RC_PORT = 133

RC_ROUTE = 135

RC_SUM = 130

RC_TIMEOUT = 134

spinnman.messages.scp.enums.signal module

class spinnman.messages.scp.enums.signal.**Signal**(*value*, *signal_type*, *doc=''*)

Bases: enum.Enum

SCP Signals

Parameters

- **value** (*int*) – The value used for the signal
- **signal_type** (*SignalType*) – The "type" of the signal

CONTINUE = 7

EXIT = 8

INITIALISE = 0

PAUSE = 6

POWER_DOWN = 1

START = 3

STOP = 2

SYNC0 = 4

SYNC1 = 5

TIMER = 9

USER_0 = 10

USER_1 = 11

USER_2 = 12

USER_3 = 13

signal_type

class spinnman.messages.scp.enums.signal.**SignalType**

Bases: enum.Enum

The type of signal, determined by how it is transmitted

MULTICAST = 0

NEAREST_NEIGHBOUR = 2

POINT_TO_POINT = 1

Module contents

class spinnman.messages.scp.enums.**AllocFree**(*value*, *doc=''*)

Bases: enum.Enum

The SCP Allocation and Free codes

ALLOC_ROUTING = 3

ALLOC_SDRAM = 0

FREE_ROUTING_BY_APP_ID = 5

FREE_ROUTING_BY_POINTER = 4

FREE_SDRAM_BY_APP_ID = 2

FREE_SDRAM_BY_POINTER = 1

class spinnman.messages.scp.enums.**BMPInfo**(*value*, *doc=''*)
Bases: enum.Enum

The SCP BMP Information Types

ADC = 3

CAN_STATUS = 2

IP_ADDR = 4

SERIAL = 0

class spinnman.messages.scp.enums.**SCPCommand**(*value*, *doc=''*)
Bases: enum.Enum

The SCP Commands

CMD_ALLOC = 28

CMD_APLX = 4

CMD_AR = 19

CMD_AS = 24

CMD_BMP_INFO = 48

CMD_BMP_POWER = 57

CMD_DPRI = 30

CMD_FFD = 23

CMD_FILL = 5

CMD_FLASH_COPY = 49

CMD_FLASH_ERASE = 50

CMD_FLASH_WRITE = 51

CMD_INFO = 31

CMD_IPTAG = 26

CMD_LED = 25

CMD_LINK_READ = 17

CMD_LINK_WRITE = 18

CMD_NNP = 20

CMD_P2PC = 21

CMD_READ = 2

CMD_REMAP = 16

CMD_RESET = 55

CMD_RTR = 29

CMD_RUN = 1

CMD_SIG = 22

CMD_SROM = 27

CMD_TUBE = 64

CMD_VER = 0

CMD_WRITE = 3

class spinnman.messages.scp.enums.**IPTagCommand**(*value*, *doc=''*)
Bases: enum.Enum

SCP IPTag Commands

CLR = 3

GET = 2

NEW = 0

SET = 1

TTO = 4

class spinnman.messages.scp.enums.**LEDAction**(*value*, *doc=''*)
Bases: enum.Enum

The SCP LED actions

OFF = 2

ON = 3

TOGGLE = 1

class spinnman.messages.scp.enums.**PowerCommand**(*value*, *doc=''*)
Bases: enum.Enum

The SCP Power Commands

POWER_OFF = 0

POWER_ON = 1

class spinnman.messages.scp.enums.**SCPResult**(*value*, *doc=''*)
Bases: enum.Enum

The SCP Result codes

RC_ARG = 132

RC_BUF = 138

RC_CMD = 131

RC_CPU = 136

RC_DEAD = 137

RC_LEN = 129

RC_OK = 128

RC_P2P_BUSY = 141

RC_P2P_NOREPLY = 139

RC_P2P_REJECT = 140

RC_P2P_TIMEOUT = 142

RC_PKT_TX = 143

RC_PORT = 133

RC_ROUTE = 135

RC_SUM = 130

RC_TIMEOUT = 134

class spinnman.messages.scp.enums.**Signal**(*value*, *signal_type*, *doc=''*)

Bases: enum.Enum

SCP Signals

Parameters

- **value** (*int*) – The value used for the signal
- **signal_type** (*SignalType*) – The "type" of the signal

CONTINUE = 7

EXIT = 8

INITIALISE = 0

PAUSE = 6

POWER_DOWN = 1

START = 3

STOP = 2

SYNC0 = 4

SYNC1 = 5

TIMER = 9

USER_0 = 10

USER_1 = 11

USER_2 = 12

USER_3 = 13

signal_type

spinnman.messages.scp.impl package

Submodules

spinnman.messages.scp.impl.app_stop module

class spinnman.messages.scp.impl.app_stop.**AppStop**(*app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to stop an application

Parameters **app_id** (*int*) – The id of the application, between 0 and 255

get_scp_response()

spinnman.messages.scp.impl.application_run module

class spinnman.messages.scp.impl.application_run.**ApplicationRun**(*app_id*, *x*, *y*, *processors*, *wait=False*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to run an application loaded on a chip

Parameters

- **app_id** (*int*) – The id of the application to run, between 16 and 255
- **x** (*int*) – The x-coordinate of the chip to run on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to run on, between 0 and 255
- **processors** (*list of int*) – The processors on the chip where the executable should be started, between 1 and 17
- **wait** (*bool*) – True if the processors should enter a "wait" state on starting

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

spinnman.messages.scp.impl.bmp_get_version module

BMPGetVersion

class spinnman.messages.scp.impl.bmp_get_version.**BMPGetVersion**(*board*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

An SCP request to read the version of software running on a core

Parameters **board** (*int*) – The board to get the version from

Raises ***spinnman.exceptions.SpinnmanInvalidParameterException*** –

- If the chip coordinates are out of range
- If the processor is out of range

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

spinnman.messages.scp.impl.bmp_set_led module

BMPSetLed

class spinnman.messages.scp.impl.bmp_set_led.**BMPSetLed**(*led*, *action*, *boards*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

Set the led(s) of a board to either on, off or toggling

Parameters

- **led** (*int or iterable of int*) – Number of the LED or an iterable of LEDs to set the state of (0-7)
- **action** (spinnman.messages.scp.scp_led_action.SCPLEDAction) – State to set the LED to, either on, off or toggle
- **boards** (*int or iterable of int*) – Specifies the board to control the LEDs of. This may also be an iterable of multiple boards (in the same frame).

Return type None

get_scp_response()
Get the response from the write fpga register request

spinnman.messages.scp.impl.check_ok_response module

class spinnman.messages.scp.impl.check_ok_response.**CheckOKResponse**(*operation*, *command*)
Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

An SCP response to a request which returns nothing other than OK

Parameters

- **operation** (*str*) – The operation being performed
- **command** (*str*) – The command that was sent

read_data_bytestring(*data*, *offset*)
See spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()

spinnman.messages.scp.impl.count_state module

class spinnman.messages.scp.impl.count_state.**CountState**(*app_id*, *state*)
Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to get a count of the cores in a particular state

Parameters

- **app_id** (*int*) – The id of the application, between 0 and 255
- **state** (spinnman.model.cpu_state.CPUState) – The state to count

get_scp_response()

spinnman.messages.scp.impl.count_state_response module

class spinnman.messages.scp.impl.count_state_response.**CountStateResponse**
Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

An SCP response to a request for the number of cores in a given state

count
The count of the number of cores with the requested state

Return type int

read_data_bytestring(*data*, *offset*)

See `spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()`

spinnman.messages.scp.impl.fill_request module

class spinnman.messages.scp.impl.fill_request.**FillRequest**(*x*, *y*, *base_address*, *data*, *size*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to fill a region of memory on a chip with repeated data

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **base_address** (*int*) – The positive base address to start the fill from
- **data** (*int*) – The data to fill in the space with
- **size** (*int*) – The number of bytes to fill in

get_scp_response()

See `spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()`

spinnman.messages.scp.impl.fixed_route_init module

class spinnman.messages.scp.impl.fixed_route_init.**FixedRouteInit**(*x*, *y*, *entry*, *app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

sets a fixed route entry

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **entry** – the fixed route entry converted for writing
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If x is out of range
- If y is out of range

get_scp_response()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type `spinnman.messages.scp_response.SCPResponse`

Raises **`None`** – No known exceptions are raised

spinnman.messages.scp.impl.fixed_route_read module

class `spinnman.messages.scp.impl.fixed_route_read.`**`FixedRouteRead`**(*x*, *y*, *app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

sets a fixed route entry

Parameters

- **`x`** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **`y`** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **`app_id`** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Raises ***spinnman.exceptions.SpinnmanInvalidParameterException*** –

- If x is out of range
- If y is out of range

`get_scp_response`()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type `spinnman.messages.scp_response.SCPResponse`

Raises **`None`** – No known exceptions are raised

spinnman.messages.scp.impl.flood_fill_data module

class `spinnman.messages.scp.impl.flood_fill_data.`**`FloodFillData`**(*nearest_neighbour_id*, *block_no*, *base_address*, *data*, *offset=0*, *length=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to start a flood fill of data

Parameters

- **`nearest_neighbour_id`** (*int*) – The id of the packet, between 0 and 127
- **`block_no`** (*int*) – Which block this block is, between 0 and 255
- **`base_address`** (*int*) – The base address where the data is to be loaded
- **`data`** (*bytearray*) – The data to load, between 4 and 256 bytes and the size must be divisible by 4

bytestring

get_scp_response()

spinnman.messages.scp.impl.flood_fill_end module

class spinnman.messages.scp.impl.flood_fill_end.**FloodFillEnd**(*nearest_neighbour_id*, *app_id=0*, *processors=None*, *wait=False*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to start a flood fill of data

Parameters

- **nearest_neighbour_id** (*int*) – The id of the packet, between 0 and 127
- **app_id** (*int*) – The application id to start using the data, between 16 and 255. If not specified, no application is started
- **processors** (*iterable of int*) – A list of processors on which to start the application, each between 1 and 17. If not specified, no application is started.
- **wait** (*bool*) – True if the binary should go into a "wait" state before executing

get_scp_response()

spinnman.messages.scp.impl.flood_fill_start module

class spinnman.messages.scp.impl.flood_fill_start.**FloodFillStart**(*nearest_neighbour_id*, *n_blocks*, *x=None*, *y=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to start a flood fill of data

Parameters

- **nearest_neighbour_id** (*int*) – The id of the packet, between 0 and 127
- **n_blocks** (*int*) – The number of blocks of data that will be sent, between 0 and 255
- **x** (*int*) – The x-coordindate of the chip to load the data on to. If not specified, the data will be loaded on to all chips
- **y** (*int*) – The y-coordinate of the chip to load the data on to. If not specified, the data will be loaded on to all chips

get_scp_response()

spinnman.messages.scp.impl.get_chip_info module

class spinnman.messages.scp.impl.get_chip_info.**GetChipInfo**(*x*, *y*, *with_size=False*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read the chip information from a core

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **with_size** (*bool*) – True if the size should be included in the response, False if not

get_scp_response()

See `spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()`

spinnman.messages.scp.impl.get_chip_info_response module

class spinnman.messages.scp.impl.get_chip_info_response.**GetChipInfoResponse**

Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

An SCP response to a request for the version of software running

chip_info

The chip information received

Return type *spinnman.model.chip_summary_info.ChipSummaryInfo*

read_data_bytestring(*data*, *offset*)

See `spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()`

spinnman.messages.scp.impl.get_version module

class spinnman.messages.scp.impl.get_version.**GetVersion**(*x*, *y*, *p*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read the version of software running on a core

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **p** (*int*) – The id of the processor to read the version from, between 0 and 31

Raises ***spinnman.exceptions.SpinnmanInvalidParameterException*** –

- If the chip coordinates are out of range
- If the processor is out of range

get_scp_response()

See `spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()`

spinnman.messages.scp.impl.get_version_response module

class spinnman.messages.scp.impl.get_version_response.**GetVersionResponse**

Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

An SCP response to a request for the version of software running

read_data_bytestring(*data*, *offset*)

See spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()

version_info

The version information received

Return type *spinnman.model.version_info.VersionInfo*

spinnman.messages.scp.impl.iptag_clear module

class spinnman.messages.scp.impl.iptag_clear.**IPTagClear**(*x*, *y*, *tag*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to clear an IP Tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **tag** (*int*) – The tag, between 0 and 7

get_scp_response()

spinnman.messages.scp.impl.iptag_get module

class spinnman.messages.scp.impl.iptag_get.**IPTagGet**(*x*, *y*, *tag*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to get an IP tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **tag** (*int*) – The tag to get details of, between 0 and 7
- **tag** – The tag, between 0 and 7

get_scp_response()

spinnman.messages.scp.impl.iptag_get_info module

class spinnman.messages.scp.impl.iptag_get_info.**IPTagGetInfo**(*x*, *y*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request information about IP tags

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255

get_scp_response()

spinnman.messages.scp.impl.iptag_get_info_response module

class spinnman.messages.scp.impl.iptag_get_info_response.**IPTagGetInfoResponse**

Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

An SCP response to a request for information about IP tags

fixed_size

The count of the number of fixed IP tag entries

Return type int

pool_size

The count of the IP tag pool size

Return type int

read_data_bytestring(*data*, *offset*)

See spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()

transient_timeout

The timeout for transient IP tags (i.e. responses to SCP commands)

Return type int

spinnman.messages.scp.impl.iptag_set module

class spinnman.messages.scp.impl.iptag_set.**IPTagSet**(*x*, *y*, *host*, *port*, *tag*, *strip*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to set an IP Tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **host** (*bytearray*) – The host address, as an array of 4 bytes
- **port** (*int*) – The port, between 0 and 65535
- **tag** (*int*) – The tag, between 0 and 7

- **strip** (*bool*) – if the SDP header should be striped from the packet.

get_scp_response()

spinnman.messages.scp.impl.iptag_set_tto module

class spinnman.messages.scp.impl.iptag_set_tto.**IPTagSetTTO**(*x*, *y*, *tag_timeout*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to set the transient timeout for future SCP requests

Parameters

- **x** (*int*) – The x-coordinate of the chip to run on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to run on, between 0 and 255
- **tag_timeout** – The timeout value, via the IPTAG_TIME_OUT_WAIT_TIMES enum located in spinnman.constants

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

spinnman.messages.scp.impl.read_adc module

ScpReadAdcRequest

class spinnman.messages.scp.impl.read_adc.**ReadADC**(*board*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

SCP Request for the data from the BMP including voltages and temperature.

Parameters **board** – which board to request the adc register from

Return type None

get_scp_response()

spinnman.messages.scp.impl.read_fpga_register module

ReadFPGARegister

class spinnman.messages.scp.impl.read_fpga_register.**ReadFPGARegister**(*fpga_num*, *register*, *board*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

Requests the data from a fpga's register

sets up a read fpga register request

Parameters

- **fpga_num** – FPGA number (0, 1 or 2) to communicate with.
- **register** – Register address to read to (will be rounded down to the nearest 32-bit word boundary).
- **board** – which board to request the fpga register from

Return type None

get_scp_response()

spinnman.messages.scp.impl.read_link module

class spinnman.messages.scp.impl.read_link.**ReadLink**(*x*, *y*, *link*, *base_address*, *size*, *cpu=0*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read a region of memory via a link on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **cpu** (*int*) – The CPU core to use, normally 0 (or if a BMP, the board slot number)
- **link** (*int*) – The id of the link down which to send the query
- **base_address** (*int*) – The positive base address to start the read from
- **size** (*int*) – The number of bytes to read, between 1 and 256

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

spinnman.messages.scp.impl.read_memory module

class spinnman.messages.scp.impl.read_memory.**ReadMemory**(*x*, *y*, *base_address*, *size*, *cpu=0*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read a region of memory on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **base_address** (*int*) – The positive base address to start the read from
- **size** (*int*) – The number of bytes to read, between 1 and 256

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If the chip coordinates are out of range
- If the base address is not a positive number
- If the size is out of range

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

spinnman.messages.scp.impl.reverse_iptag_set module

class spinnman.messages.scp.impl.reverse_iptag_set.**ReverseIPTagSet**(*x*, *y*, *destination_x*, *destination_y*, *destination_p*, *port*, *tag*, *sdp_port*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to set an IP Tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **destination_x** (*int*) – The x-coordinate of the destination chip, between 0 and 255
- **destination_y** (*int*) – The y-coordinate of the destination chip, between 0 and 255
- **destination_p** (*int*) – the id of the destination processor, between 0 and 17
- **port** (*int*) – The port, between 0 and 65535
- **tag** (*int*) – The tag, between 0 and 7

get_scp_response()

spinnman.messages.scp.impl.router_alloc module

class spinnman.messages.scp.impl.router_alloc.**RouterAlloc**(*x*, *y*, *app_id*, *n_entries*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to allocate space for routing entries

Parameters

- **x** (*int*) – The x-coordinate of the chip to allocate on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to allocate on, between 0 and 255
- **app_id** (*int*) – The id of the application, between 0 and 255
- **n_entries** (*int*) – The number of entries to allocate

get_scp_response()

spinnman.messages.scp.impl.router_clear module

class spinnman.messages.scp.impl.router_clear.**RouterClear**(*x*, *y*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to clear the router on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If x is out of range
- If y is out of range

get_scp_response()

spinnman.messages.scp.impl.router_init module

class spinnman.messages.scp.impl.router_init.**RouterInit**(*x*, *y*, *n_entries*, *table_address*, *base_address*, *app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to initialize the router on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **n_entries** (*int*) – The number of entries in the table, more than 0
- **table_address** (*int*) – The allocated table address
- **base_address** (*int*) – The base_address containing the entries
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If x is out of range
- If y is out of range
- If n_entries is 0 or less
- If table_address is not positive
- If base_address is not positive

get_scp_response()

spinnman.messages.scp.impl.sdram_alloc module

class spinnman.messages.scp.impl.sdram_alloc.**SDRAMAlloc**(*x*, *y*, *app_id*, *size*, *tag=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to allocate space in the SDRAM space

Parameters

- **x** (*int*) – The x-coordinate of the chip to allocate on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to allocate on, between 0 and 255
- **app_id** (*int*) – The id of the application, between 0 and 255
- **size** (*int*) – The size in bytes of memory to be allocated
- **tag** (*int*) – the tag for the SDRAM, a 8-bit (chip-wide) tag that can be looked up by a SpiNNaker application to discover the address of the allocated block. If *0* then no tag is applied.

get_scp_response()

spinnman.messages.scp.impl.sdram_de_alloc module

class spinnman.messages.scp.impl.sdram_de_alloc.**SDRAMDeAlloc**(*x*, *y*, *app_id*, *base_address=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to free space in the SDRAM

Parameters

- **x** (*int*) – The x-coordinate of the chip to allocate on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to allocate on, between 0 and 255
- **app_id** (*int*) – The id of the application, between 0 and 255
- **base_address** (*int or None*) – The start address in SDRAM to which the block needs to be deallocated, or none if deallocating via app_id

get_scp_response()

spinnman.messages.scp.impl.send_signal module

class spinnman.messages.scp.impl.send_signal.**SendSignal**(*app_id*, *signal*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to send a signal to cores

Parameters

- **app_id** (*int*) – The id of the application, between 0 and 255
- **signal** (spinnman.messages.scp.scp_signal.SCPSignal) – The signal to send

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If app_id is out of range

get_scp_response()

spinnman.messages.scp.impl.set_led module

class spinnman.messages.scp.impl.set_led.**SetLED**(*x*, *y*, *cpu*, *led_states*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to change the state of an SetLED

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255
- **cpu** (*int*) – The CPU-number to use to set the SetLED.
- **led_states** (*dict*) – A dictionary mapping SetLED index to state with 0 being off, 1 on and 2 inverted.

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

spinnman.messages.scp.impl.set_power module

SetPower

class spinnman.messages.scp.impl.set_power.**SetPower**(*power_command*, *boards*, *delay=0.0*, *board_to_send_to=0*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

An SCP request for the BMP to power on or power off a rack of boards

Parameters

- **power_command** (spinnman.messages.scp.scp_power_command.SCPPowerCommand) – The power command being sent
- **boards** (*int or iterable of int*) – The boards on the same backplane to power on or off
- **delay** (*int*) – Number of seconds delay between power state changes of the different boards.
- **board_to_send_to** – The optional board to send the command to if this is to be sent to a frame of boards. NOTE: There is currently a bug in the BMP that means some boards don't respond to power commands not sent to BMP 0. Thus changing this parameter is not recommended!

Type board_to_send_to: 0

Return type None

get_scp_response()

Get the response from the powering message

spinnman.messages.scp.impl.write_fpga_register module

ScpWriteFPGARegisterRequest

class spinnman.messages.scp.impl.write_fpga_register.**WriteFPGARegister**(*fpga_num*, *addr*, *value*, *board*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

A request for writing data to a FPGA register

Write the value of an FPGA (SPI) register.

See the SpI/O project's spinnaker_fpga design's README for a listing of FPGA registers. The SpI/O project can be found on GitHub at: https://github.com/SpiNNakerManchester/spio/

Parameters

- **fpga_num** (*int*) – FPGA number (0, 1 or 2) to communicate with.
- **addr** (*int*) – Register address to read or write to (will be rounded down to the nearest 32-bit word boundary).
- **value** (*int*) – A 32-bit int value to write to the register

get_scp_response()

Return type *spinnman.messages.scp.impl.CheckOKResponse*

spinnman.messages.scp.impl.write_link module

class spinnman.messages.scp.impl.write_link.**WriteLink**(*x*, *y*, *link*, *base_address*, *data*, *cpu=0*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to write memory on a neighbouring chip

Parameters

- **x** (*int*) – The x-coordinate of the chip whose neighbour will be written to, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip whose neighbour will be written to, between 0 and 255
- **cpu** (*int*) – The CPU core to use, normally 0 (or if a BMP, the board slot number)
- **link** (*int*) – The link number to write to between 0 and 5 (or if a BMP, the FPGA between 0 and 2)
- **base_address** (*int*) – The base_address to start writing to
- **data** (*bytearray*) – Up to 256 bytes of data to write

bytestring

get_scp_response()

spinnman.messages.scp.impl.write_memory module

class spinnman.messages.scp.impl.write_memory.**WriteMemory**(*x*, *y*, *base_address*, *data*, *cpu=0*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to write memory on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **base_address** (*int*) – The base_address to start writing to the base address is not checked to see if its not valid
- **data** (*bytearray or string*) – between 1 and 256 bytes of data to write this is not checked due to speed restrictions

bytestring

get_scp_response()

Module contents

class spinnman.messages.scp.impl.**AppStop**(*app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to stop an application

Parameters **app_id** (*int*) – The id of the application, between 0 and 255

get_scp_response()

class spinnman.messages.scp.impl.**ApplicationRun**(*app_id*, *x*, *y*, *processors*, *wait=False*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to run an application loaded on a chip

Parameters

- **app_id** (*int*) – The id of the application to run, between 16 and 255
- **x** (*int*) – The x-coordinate of the chip to run on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to run on, between 0 and 255
- **processors** (*list of int*) – The processors on the chip where the executable should be started, between 1 and 17
- **wait** (*bool*) – True if the processors should enter a "wait" state on starting

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

class spinnman.messages.scp.impl.**BMPSetLed**(*led*, *action*, *boards*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

Set the led(s) of a board to either on, off or toggling

Parameters

- **led** (*int or iterable of int*) – Number of the LED or an iterable of LEDs to set the state of (0-7)

- **action** (spinnman.messages.scp.scp_led_action.SCPLEDAction) – State to set the LED to, either on, off or toggle
- **boards** (*int or iterable of int*) – Specifies the board to control the LEDs of. This may also be an iterable of multiple boards (in the same frame).

Return type None

get_scp_response()
Get the response from the write fpga register request

class spinnman.messages.scp.impl.**BMPGetVersion**(*board*)
Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

An SCP request to read the version of software running on a core

Parameters **board** (*int*) – The board to get the version from

Raises ***spinnman.exceptions.SpinnmanInvalidParameterException*** –

- If the chip coordinates are out of range
- If the processor is out of range

get_scp_response()
See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

class spinnman.messages.scp.impl.**CheckOKResponse**(*operation*, *command*)
Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

An SCP response to a request which returns nothing other than OK

Parameters

- **operation** (*str*) – The operation being performed
- **command** (*str*) – The command that was sent

read_data_bytestring(*data*, *offset*)
See spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()

class spinnman.messages.scp.impl.**GetChipInfo**(*x*, *y*, *with_size=False*)
Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read the chip information from a core

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **with_size** (*bool*) – True if the size should be included in the response, False if not

get_scp_response()
See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

class spinnman.messages.scp.impl.**CountState**(*app_id*, *state*)
Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to get a count of the cores in a particular state

Parameters

- **app_id** (*int*) – The id of the application, between 0 and 255
- **state** (spinnman.model.cpu_state.CPUState) – The state to count

get_scp_response()

class spinnman.messages.scp.impl.**FloodFillData**(*nearest_neighbour_id*, *block_no*, *base_address*, *data*, *offset=0*, *length=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to start a flood fill of data

Parameters

- **nearest_neighbour_id** (*int*) – The id of the packet, between 0 and 127
- **block_no** (*int*) – Which block this block is, between 0 and 255
- **base_address** (*int*) – The base address where the data is to be loaded
- **data** (*bytearray*) – The data to load, between 4 and 256 bytes and the size must be divisible by 4

bytestring

get_scp_response()

class spinnman.messages.scp.impl.**FillRequest**(*x*, *y*, *base_address*, *data*, *size*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to fill a region of memory on a chip with repeated data

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **base_address** (*int*) – The positive base address to start the fill from
- **data** (*int*) – The data to fill in the space with
- **size** (*int*) – The number of bytes to fill in

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

class spinnman.messages.scp.impl.**FloodFillEnd**(*nearest_neighbour_id*, *app_id=0*, *processors=None*, *wait=False*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to start a flood fill of data

Parameters

- **nearest_neighbour_id** (*int*) – The id of the packet, between 0 and 127
- **app_id** (*int*) – The application id to start using the data, between 16 and 255. If not specified, no application is started

- **processors** (*iterable of int*) – A list of processors on which to start the application, each between 1 and 17. If not specified, no application is started.
- **wait** (*bool*) – True if the binary should go into a "wait" state before executing

get_scp_response()

class spinnman.messages.scp.impl.**FloodFillStart**(*nearest_neighbour_id*, *n_blocks*, *x=None*, *y=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to start a flood fill of data

Parameters

- **nearest_neighbour_id** (*int*) – The id of the packet, between 0 and 127
- **n_blocks** (*int*) – The number of blocks of data that will be sent, between 0 and 255
- **x** (*int*) – The x-coordindate of the chip to load the data on to. If not specified, the data will be loaded on to all chips
- **y** (*int*) – The y-coordinate of the chip to load the data on to. If not specified, the data will be loaded on to all chips

get_scp_response()

class spinnman.messages.scp.impl.**IPTagClear**(*x*, *y*, *tag*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to clear an IP Tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **tag** (*int*) – The tag, between 0 and 7

get_scp_response()

class spinnman.messages.scp.impl.**IPTagGet**(*x*, *y*, *tag*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to get an IP tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **tag** (*int*) – The tag to get details of, between 0 and 7
- **tag** – The tag, between 0 and 7

get_scp_response()

class spinnman.messages.scp.impl.**IPTagGetInfo**(*x*, *y*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request information about IP tags

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255

get_scp_response()

class spinnman.messages.scp.impl.**IPTagSet**(*x*, *y*, *host*, *port*, *tag*, *strip*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to set an IP Tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **host** (*bytearray*) – The host address, as an array of 4 bytes
- **port** (*int*) – The port, between 0 and 65535
- **tag** (*int*) – The tag, between 0 and 7
- **strip** (*bool*) – if the SDP header should be striped from the packet.

get_scp_response()

class spinnman.messages.scp.impl.**IPTagSetTTO**(*x*, *y*, *tag_timeout*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to set the transient timeout for future SCP requests

Parameters

- **x** (*int*) – The x-coordinate of the chip to run on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to run on, between 0 and 255
- **tag_timeout** – The timeout value, via the IPTAG_TIME_OUT_WAIT_TIMES enum located in spinnman.constants

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

class spinnman.messages.scp.impl.**SetLED**(*x*, *y*, *cpu*, *led_states*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to change the state of an SetLED

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255
- **cpu** (*int*) – The CPU-number to use to set the SetLED.
- **led_states** (*dict*) – A dictionary mapping SetLED index to state with 0 being off, 1 on and 2 inverted.

get_scp_response()
See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

class spinnman.messages.scp.impl.**SetPower**(*power_command*, *boards*, *delay=0.0*, *board_to_send_to=0*)
Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

An SCP request for the BMP to power on or power off a rack of boards

Parameters

- **power_command** (spinnman.messages.scp.scp_power_command.SCPPowerCommand) – The power command being sent
- **boards** (*int or iterable of int*) – The boards on the same backplane to power on or off
- **delay** (*int*) – Number of seconds delay between power state changes of the different boards.
- **board_to_send_to** – The optional board to send the command to if this is to be sent to a frame of boards. NOTE: There is currently a bug in the BMP that means some boards don't respond to power commands not sent to BMP 0. Thus changing this parameter is not recommended!

Type board_to_send_to: 0

Return type None

get_scp_response()
Get the response from the powering message

class spinnman.messages.scp.impl.**ReadADC**(*board*)
Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

SCP Request for the data from the BMP including voltages and temperature.

Parameters **board** – which board to request the adc register from

Return type None

get_scp_response()

class spinnman.messages.scp.impl.**ReadFPGARegister**(*fpga_num*, *register*, *board*)
Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

Requests the data from a fpga's register

sets up a read fpga register request

Parameters

- **fpga_num** – FPGA number (0, 1 or 2) to communicate with.
- **register** – Register address to read to (will be rounded down to the nearest 32-bit word boundary).
- **board** – which board to request the fpga register from

Return type None

get_scp_response()

class spinnman.messages.scp.impl.**ReadLink**(*x*, *y*, *link*, *base_address*, *size*, *cpu=0*)
Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read a region of memory via a link on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **cpu** (*int*) – The CPU core to use, normally 0 (or if a BMP, the board slot number)
- **link** (*int*) – The id of the link down which to send the query
- **base_address** (*int*) – The positive base address to start the read from
- **size** (*int*) – The number of bytes to read, between 1 and 256

get_scp_response()

See `spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()`

class spinnman.messages.scp.impl.**ReadMemory**(*x*, *y*, *base_address*, *size*, *cpu=0*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read a region of memory on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **base_address** (*int*) – The positive base address to start the read from
- **size** (*int*) – The number of bytes to read, between 1 and 256

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If the chip coordinates are out of range
- If the base address is not a positive number
- If the size is out of range

get_scp_response()

See `spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()`

class spinnman.messages.scp.impl.**ReverseIPTagSet**(*x*, *y*, *destination_x*, *destination_y*, *destination_p*, *port*, *tag*, *sdp_port*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to set an IP Tag

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **destination_x** (*int*) – The x-coordinate of the destination chip, between 0 and 255
- **destination_y** (*int*) – The y-coordinate of the destination chip, between 0 and 255
- **destination_p** (*int*) – the id of the destination processor, between 0 and 17
- **port** (*int*) – The port, between 0 and 65535

- **tag** (*int*) – The tag, between 0 and 7

get_scp_response()

class spinnman.messages.scp.impl.**RouterAlloc**(*x*, *y*, *app_id*, *n_entries*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to allocate space for routing entries

Parameters

- **x** (*int*) – The x-coordinate of the chip to allocate on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to allocate on, between 0 and 255
- **app_id** (*int*) – The id of the application, between 0 and 255
- **n_entries** (*int*) – The number of entries to allocate

get_scp_response()

class spinnman.messages.scp.impl.**RouterClear**(*x*, *y*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to clear the router on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If x is out of range
- If y is out of range

get_scp_response()

class spinnman.messages.scp.impl.**RouterInit**(*x*, *y*, *n_entries*, *table_address*, *base_address*, *app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to initialize the router on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **n_entries** (*int*) – The number of entries in the table, more than 0
- **table_address** (*int*) – The allocated table address
- **base_address** (*int*) – The base_address containing the entries
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If x is out of range
- If y is out of range
- If n_entries is 0 or less
- If table_address is not positive
- If base_address is not positive

get_scp_response()

class spinnman.messages.scp.impl.**SDRAMAlloc**(*x*, *y*, *app_id*, *size*, *tag=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to allocate space in the SDRAM space

Parameters

- **x** (*int*) – The x-coordinate of the chip to allocate on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to allocate on, between 0 and 255
- **app_id** (*int*) – The id of the application, between 0 and 255
- **size** (*int*) – The size in bytes of memory to be allocated
- **tag** (*int*) – the tag for the SDRAM, a 8-bit (chip-wide) tag that can be looked up by a SpiNNaker application to discover the address of the allocated block. If *0* then no tag is applied.

get_scp_response()

class spinnman.messages.scp.impl.**SDRAMDeAlloc**(*x*, *y*, *app_id*, *base_address=None*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to free space in the SDRAM

Parameters

- **x** (*int*) – The x-coordinate of the chip to allocate on, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to allocate on, between 0 and 255
- **app_id** (*int*) – The id of the application, between 0 and 255
- **base_address** (*int or None*) – The start address in SDRAM to which the block needs to be deallocated, or none if deallocating via app_id

get_scp_response()

class spinnman.messages.scp.impl.**SendSignal**(*app_id*, *signal*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to send a signal to cores

Parameters

- **app_id** (*int*) – The id of the application, between 0 and 255
- **signal** (spinnman.messages.scp.scp_signal.SCPSignal) – The signal to send

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If app_id is out of range

get_scp_response()

class spinnman.messages.scp.impl.**GetVersion**(*x*, *y*, *p*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP request to read the version of software running on a core

Parameters

- **x** (*int*) – The x-coordinate of the chip to read from, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip to read from, between 0 and 255
- **p** (*int*) – The id of the processor to read the version from, between 0 and 31

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If the chip coordinates are out of range
- If the processor is out of range

get_scp_response()

See spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest.get_scp_response()

class spinnman.messages.scp.impl.**WriteFPGARegister**(*fpga_num*, *addr*, *value*, *board*)

Bases: *spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest*

A request for writing data to a FPGA register

Write the value of an FPGA (SPI) register.

See the SpI/O project's spinnaker_fpga design's README for a listing of FPGA registers. The SpI/O project can be found on GitHub at: https://github.com/SpiNNakerManchester/spio/

Parameters

- **fpga_num** (*int*) – FPGA number (0, 1 or 2) to communicate with.
- **addr** (*int*) – Register address to read or write to (will be rounded down to the nearest 32-bit word boundary).
- **value** (*int*) – A 32-bit int value to write to the register

get_scp_response()

Return type *spinnman.messages.scp.impl.CheckOKResponse*

class spinnman.messages.scp.impl.**FixedRouteRead**(*x*, *y*, *app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

sets a fixed route entry

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions

- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If x is out of range
- If y is out of range

get_scp_response()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type spinnman.messages.scp_response.SCPResponse

Raises **None** – No known exceptions are raised

class spinnman.messages.scp.impl.**WriteLink**(*x*, *y*, *link*, *base_address*, *data*, *cpu=0*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to write memory on a neighbouring chip

Parameters

- **x** (*int*) – The x-coordinate of the chip whose neighbour will be written to, between 0 and 255
- **y** (*int*) – The y-coordinate of the chip whose neighbour will be written to, between 0 and 255
- **cpu** (*int*) – The CPU core to use, normally 0 (or if a BMP, the board slot number)
- **link** (*int*) – The link number to write to between 0 and 5 (or if a BMP, the FPGA between 0 and 2)
- **base_address** (*int*) – The base_address to start writing to
- **data** (*bytearray*) – Up to 256 bytes of data to write

bytestring

get_scp_response()

class spinnman.messages.scp.impl.**WriteMemory**(*x*, *y*, *base_address*, *data*, *cpu=0*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

A request to write memory on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **base_address** (*int*) – The base_address to start writing to the base address is not checked to see if its not valid
- **data** (*bytearray or string*) – between 1 and 256 bytes of data to write this is not checked due to speed restrictions

bytestring

get_scp_response()

class spinnman.messages.scp.impl.**FixedRouteInit**(*x*, *y*, *entry*, *app_id*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

sets a fixed route entry

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **entry** – the fixed route entry converted for writing
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* –

- If x is out of range
- If y is out of range

get_scp_response()

Get an SCP response message to be used to process any response received

Returns An SCP response, or None if no response is required

Return type spinnman.messages.scp_response.SCPResponse

Raises **None** – No known exceptions are raised

Submodules

spinnman.messages.scp.scp_request_header module

class spinnman.messages.scp.scp_request_header.**SCPRequestHeader**(*command*, *sequence=0*)

Bases: object

Represents the header of an SCP Request Each optional parameter in the constructor can be set to a value other than None once, after which it is immutable. It is an error to set a parameter that is not currently None.

Parameters

- **command** (spinnman.messages.scp.scp_command.SCPCommand) – The SCP command
- **sequence** (*int*) – The number of the SCP packet in order of all packets sent or received, between 0 and 65535

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If one of the parameters is incorrect

bytestring

The header as a bytestring

Returns The header as a bytestring

Return type str

command
The command of the SCP packet

Returns The command

Return type `spinnman.messages.scp.scp_command.SCPCommand`

sequence
The sequence number of the SCP packet

Returns The sequence number of the packet, between 0 and 65535

Return type int

spinnman.messages.scp.scp_response_header module

class `spinnman.messages.scp.scp_response_header.`**`SCPResponseHeader`**(*result=None*, *sequence=None*)

Bases: `object`

Represents the header of an SCP Response

static from_bytestring(*data*, *offset*)
Read a header from a bytestring

Parameters

- **data** (*str*) – The bytestring to read from
- **offset** –

result
The result of the SCP response

Returns The result

Return type `spinnman.messages.scp.scp_result.SCPResult`

sequence
The sequence number of the SCP response

Returns The sequence number of the packet, between 0 and 65535

Return type int

Module contents

class `spinnman.messages.scp.`**`SCPRequestHeader`**(*command*, *sequence=0*)
Bases: `object`

Represents the header of an SCP Request Each optional parameter in the constructor can be set to a value other than None once, after which it is immutable. It is an error to set a parameter that is not currently None.

Parameters

- **command** (`spinnman.messages.scp.scp_command.SCPCommand`) – The SCP command
- **sequence** (*int*) – The number of the SCP packet in order of all packets sent or received, between 0 and 65535

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If one of the parameters is incorrect

bytestring
The header as a bytestring

Returns The header as a bytestring

Return type str

command
The command of the SCP packet

Returns The command

Return type `spinnman.messages.scp.scp_command.SCPCommand`

sequence
The sequence number of the SCP packet

Returns The sequence number of the packet, between 0 and 65535

Return type int

class `spinnman.messages.scp.`**SCPResponseHeader**(*result=None*, *sequence=None*)
Bases: `object`

Represents the header of an SCP Response

static from_bytestring(*data*, *offset*)
Read a header from a bytestring

Parameters

- **data** (*str*) – The bytestring to read from
- **offset** –

result
The result of the SCP response

Returns The result

Return type `spinnman.messages.scp.scp_result.SCPResult`

sequence
The sequence number of the SCP response

Returns The sequence number of the packet, between 0 and 65535

Return type int

spinnman.messages.sdp package

Submodules

spinnman.messages.sdp.sdp_flag module

class `spinnman.messages.sdp.sdp_flag.`**SDPFlag**(*value*, *doc=''*)
Bases: `enum.Enum`

SDPFlag for the message

REPLY_EXPECTED = 135

REPLY_NOT_EXPECTED = 7

spinnman.messages.sdp.sdp_header module

class spinnman.messages.sdp.sdp_header.**SDPHeader**(*flags=None*, *tag=None*, *destination_port=None*, *destination_cpu=None*, *destination_chip_x=None*, *destination_chip_y=None*, *source_port=None*, *source_cpu=None*, *source_chip_x=None*, *source_chip_y=None*)

Bases: object

Represents the header of an SDP message. Each optional parameter in the constructor can be set to a value other than None once, after which it is immutable. It is an error to set a parameter that is not currently None.

Parameters

- **flags** (*spinnman.messages.sdp.sdp_flag.SDPFlag*) – Any flags for the packet
- **tag** (*int*) – The ip tag of the packet between 0 and 255, or None if it is to be set later
- **destination_port** (*int*) – The destination port of the packet between 0 and 7
- **destination_cpu** (*int*) – The destination processor id within the chip between 0 and 31
- **destination_chip_x** (*int*) – The x-coordinate of the destination chip between 0 and 255
- **destination_chip_y** (*int*) – The y-coordinate of the destination chip between 0 and 255
- **source_port** (*int*) – The source port of the packet between 0 and 7, or None if it is to be set later
- **source_cpu** (*int*) – The source processor id within the chip between 0 and 31, or None if it is to be set later
- **source_chip_x** (*int*) – The x-coordinate of the source chip between 0 and 255, or None if it is to be set later
- **source_chip_y** – The y-coordinate of the source chip between 0 and 255, or None if it is to be set later

bytestring

The header as a bytestring

Returns The header bytestring

Return type str

destination_chip_x

The x-coordinate of the destination chip of the packet

Returns The x-coordinate of the chip, between 0 and 255

Return type int

destination_chip_y

The y-coordinate of the destination chip of the packet

Returns The y-coordinate of the chip, between 0 and 255

Return type int

destination_cpu
The core on the destination chip

Returns The core on the destination chip, between 0 and 31

Return type int

destination_port
The destination port of the packet

Returns The destination port of the packet between 0 and 7

Return type int

flags
The flags of the packet

Returns The flags of the packet

Return type *spinnman.messages.sdp.sdp_flag.SDPFlag*

static **from_bytestring**(*data*, *offset*)
Read the header from a bytestring

Parameters

- **data** (*str*) – The bytestring to read the header from
- **offset** (*int*) – The offset into the data from which to start reading

source_chip_x
The x-coordinate of the source chip of the packet

Returns The x-coordinate of the chip, between 0 and 255

Return type int

source_chip_y
The y-coordinate of the source chip of the packet

Returns The y-coordinate of the chip, between 0 and 255

Return type int

source_cpu
The core on the source chip

Returns The core on the source chip, between 0 and 31

Return type int

source_port
The source port of the packet

Returns The source port of the packet between 0 and 7

Return type int

tag
The tag of the packet

Returns The tag of the packet between 0 and 255

Return type int

spinnman.messages.sdp.sdp_message module

class spinnman.messages.sdp.sdp_message.**SDPMessage**(*sdp_header*, *data=None*, *offset=0*)

Bases: object

Wraps up an SDP message with a header and optional data.

Parameters

- **sdp_header** (*spinnman.messages.sdp.sdp_header.SDPHeader*) – The header of the message
- **data** (*str*) – The data of the SDP packet, or None if no data
- **offset** (*int*) – The offset where the valid data starts

Raises **None** – No known exceptions are thrown

bytestring

The bytestring of the message

Returns The bytestring of the message

Return type str

data

The data in the packet

Return type str

static from_bytestring(*data*, *offset*)

offset

The offset where the valid data starts

Return type int

sdp_header

The header of the packet

Returns An SDP header

Return type *spinnman.messages.sdp.sdp_header.SDPHeader*

Module contents

class spinnman.messages.sdp.**SDPFlag**(*value*, *doc=''*)

Bases: enum.Enum

SDPFlag for the message

REPLY_EXPECTED = 135

REPLY_NOT_EXPECTED = 7

class spinnman.messages.sdp.**SDPHeader**(*flags=None*, *tag=None*, *destination_port=None*, *destination_cpu=None*, *destination_chip_x=None*, *destination_chip_y=None*, *source_port=None*, *source_cpu=None*, *source_chip_x=None*, *source_chip_y=None*)

Bases: object

Represents the header of an SDP message. Each optional parameter in the constructor can be set to a value other than None once, after which it is immutable. It is an error to set a parameter that is not currently None.

Parameters

- **flags** (*spinnman.messages.sdp.sdp_flag.SDPFlag*) – Any flags for the packet
- **tag** (*int*) – The ip tag of the packet between 0 and 255, or None if it is to be set later
- **destination_port** (*int*) – The destination port of the packet between 0 and 7
- **destination_cpu** (*int*) – The destination processor id within the chip between 0 and 31
- **destination_chip_x** (*int*) – The x-coordinate of the destination chip between 0 and 255
- **destination_chip_y** (*int*) – The y-coordinate of the destination chip between 0 and 255
- **source_port** (*int*) – The source port of the packet between 0 and 7, or None if it is to be set later
- **source_cpu** (*int*) – The source processor id within the chip between 0 and 31, or None if it is to be set later
- **source_chip_x** (*int*) – The x-coordinate of the source chip between 0 and 255, or None if it is to be set later
- **source_chip_y** – The y-coordinate of the source chip between 0 and 255, or None if it is to be set later

bytestring

The header as a bytestring

Returns The header bytestring

Return type str

destination_chip_x

The x-coordinate of the destination chip of the packet

Returns The x-coordinate of the chip, between 0 and 255

Return type int

destination_chip_y

The y-coordinate of the destination chip of the packet

Returns The y-coordinate of the chip, between 0 and 255

Return type int

destination_cpu

The core on the destination chip

Returns The core on the destination chip, between 0 and 31

Return type int

destination_port

The destination port of the packet

Returns The destination port of the packet between 0 and 7

Return type int

flags

The flags of the packet

Returns The flags of the packet

Return type *spinnman.messages.sdp.sdp_flag.SDPFlag*

static from_bytestring(*data*, *offset*)
Read the header from a bytestring

Parameters

- **data** (*str*) – The bytestring to read the header from
- **offset** (*int*) – The offset into the data from which to start reading

source_chip_x
The x-coordinate of the source chip of the packet

Returns The x-coordinate of the chip, between 0 and 255

Return type int

source_chip_y
The y-coordinate of the source chip of the packet

Returns The y-coordinate of the chip, between 0 and 255

Return type int

source_cpu
The core on the source chip

Returns The core on the source chip, between 0 and 31

Return type int

source_port
The source port of the packet

Returns The source port of the packet between 0 and 7

Return type int

tag
The tag of the packet

Returns The tag of the packet between 0 and 255

Return type int

class spinnman.messages.sdp.**SDPMessage**(*sdp_header*, *data=None*, *offset=0*)
Bases: object

Wraps up an SDP message with a header and optional data.

Parameters

- **sdp_header** (*spinnman.messages.sdp.sdp_header.SDPHeader*) – The header of the message
- **data** (*str*) – The data of the SDP packet, or None if no data
- **offset** (*int*) – The offset where the valid data starts

Raises **None** – No known exceptions are thrown

bytestring
The bytestring of the message

Returns The bytestring of the message

Return type str

data
The data in the packet

Return type str

static from_bytestring(*data*, *offset*)

offset
The offset where the valid data starts

Return type int

sdp_header
The header of the packet

Returns An SDP header

Return type *spinnman.messages.sdp.sdp_header.SDPHeader*

spinnman.messages.spinnaker_boot package

Subpackages

Submodules

spinnman.messages.spinnaker_boot.spinnaker_boot_message module

class spinnman.messages.spinnaker_boot.spinnaker_boot_message.**SpinnakerBootMessage**(*opcode*, *operand_1*, *operand_2*, *operand_3*, *data=None*, *offset=0*)

Bases: object

A message used for booting the board

Parameters

- **opcode** (*spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.SpinnakerBootOpCode*) – The operation of this packet
- **operand_1** (*int*) – The first operand
- **operand_2** (*int*) – The second operand
- **operand_3** (*int*) – The third operand
- **data** (*str*) – The optional data, up to 256 words
- **offset** (*int*) – The offset of the valid data

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the opcode is not a valid value

bytestring
The message as a bytestring

data
The data

Returns The data or None if no data

Return type bytearray

static from_bytestring(*data*, *offset*)

opcode
The operation of this packet

Returns The operation code

Return type *spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.SpinnakerBootOpCode*

operand_1
The first operand

Returns The operand

Return type int

operand_2
The second operand

Returns The second operand

Return type int

operand_3
The third operand

Returns The third operand

Return type int

spinnman.messages.spinnaker_boot.spinnaker_boot_messages module

SpinnakerBootMessages

class spinnman.messages.spinnaker_boot.spinnaker_boot_messages.**SpinnakerBootMessages**(*board_vers ex- tra_boot_v*

Bases: object

Represents a set of boot messages to be sent to boot the board

builds the boot messages needed to boot the spinnaker machine

Parameters

- **board_version** (*int*) – The version of the board to be booted
- **extra_boot_values** (*dict of SystemVariableDefinition to value*) – Any additional values to be set during boot

Raises

- *spinnman.exceptions.SpinnmanInvalidParameterException* – If the board version is not supported
- *spinnman.exceptions.SpinnmanIOException* – If there is an error assembling the packets

messages
Get an iterable of message to be sent.

spinnman.messages.spinnaker_boot.spinnaker_boot_op_code module

class spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.**SpinnakerBootOpCode**(*value*, *doc=''*)

Bases: enum.Enum

Boot message Operation Codes

FLOOD_FILL_BLOCK = 3

FLOOD_FILL_CONTROL = 5

FLOOD_FILL_START = 1

HELLO = 65

spinnman.messages.spinnaker_boot.system_variable_boot_values module

class spinnman.messages.spinnaker_boot.system_variable_boot_values.**SystemVariableBootValues**(*h*
l

Bases: object

Default values of the system variables that get passed to SpiNNaker during boot

bytestring

set_value(*system_variable_definition*, *value*)

class spinnman.messages.spinnaker_boot.system_variable_boot_values.**SystemVariableDefinition**(*d*
a
a
f
a
r
a

Bases: enum.Enum

Defines the system variables available

Parameters

- **data_type** (_DataType) – The data type of the variable
- **offset** (*int*) – The offset from the start of the system variable structure where the variable is found
- **default** (*int*) – The default value assigned to the variable if not overridden
- **array_size** (*int*) – The length of the array, or None if not an array

allocated_tag_table_address = _Definition(offset=220, data_type=<_DataType.INT: 4>, default=0, array_size=N

app_data_table_address = _Definition(offset=228, data_type=<_DataType.INT: 4>, default=0, array_size=None, do

array_size

board_info = _Definition(offset=248, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A pointer to t

board_test_flags = _Definition(offset=103, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Bo

boot_signature = _Definition(offset=92, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The boot

clock_divisor = _Definition(offset=14, data_type=<_DataType.BYTE: 1>, default=51, array_size=None, doc='The clo

clock_milliseconds = _Definition(offset=16, data_type=<_DataType.LONG: 8>, default=0, array_size=None, doc='T

cpu_clock_mhz = _Definition(offset=36, data_type=<_DataType.SHORT: 2>, default=200, array_size=None, doc='The

cpu_information_base_address = _Definition(offset=204, data_type=<_DataType.INT: 4>, default=0, array_size=

data_type

debug_x = _Definition(offset=5, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The x-coordinate

debug_y = _Definition(offset=4, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The y-coordinate

default

ethernet_ip_address = _Definition(offset=240, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x00\x00\x00\

first_free_router_entry = _Definition(offset=224, data_type=<_DataType.SHORT: 2>, default=0, array_size=Nor

fixed_route_copy = _Definition(offset=244, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A (vir

hardware_version = _Definition(offset=10, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The

iobuf_size = _Definition(offset=80, data_type=<_DataType.INT: 4>, default=16384, array_size=None, doc='The size of

is_ethernet_available = _Definition(offset=11, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc

is_peer_to_peer_available = _Definition(offset=6, data_type=<_DataType.BYTE: 1>, default=0, array_size=None

is_root_chip = _Definition(offset=64, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Indicates

last_biff_id = _Definition(offset=102, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Last ID u

led_0 = _Definition(offset=48, data_type=<_DataType.INT: 4>, default=1, array_size=None, doc='The first part of the LE

led_1 = _Definition(offset=52, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The last part of the LE

led_half_period_10_ms = _Definition(offset=43, data_type=<_DataType.BYTE: 1>, default=1, array_size=None, doc

link_peek_timeout_microseconds = _Definition(offset=42, data_type=<_DataType.BYTE: 1>, default=100, array_

links_available = _Definition(offset=101, data_type=<_DataType.BYTE: 1>, default=63, array_size=None, doc='Bit

lock = _Definition(offset=100, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The lock')

log_peer_to_peer_sequence_length = _Definition(offset=13, data_type=<_DataType.BYTE: 1>, default=4, array

ltpc_period = _Definition(offset=26, data_type=<_DataType.SHORT: 2>, default=0, array_size=None, doc='‘)

monitor_mailbox_flags = _Definition(offset=236, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc=

n_active_peer_to_peer_addresses = _Definition(offset=226, data_type=<_DataType.SHORT: 2>, default=0, arra

n_scamp_working_cores = _Definition(offset=189, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, d

n_shared_message_buffers = _Definition(offset=65, data_type=<_DataType.BYTE: 1>, default=7, array_size=None

n_working_cores = _Definition(offset=188, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The

nearest_ethernet_x = _Definition(offset=9, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Th

nearest_ethernet_y = _Definition(offset=8, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Th

nearest_neighbour_delay_us = _Definition(offset=66, data_type=<_DataType.BYTE: 1>, default=10, array_size=N

nearest_neighbour_forward = _Definition(offset=40, data_type=<_DataType.BYTE: 1>, default=63, array_size=No

nearest_neighbour_last_id = _Definition(offset=7, data_type=<_DataType.BYTE: 1>, default=0, array_size=None

nearest_neighbour_memory_pointer = _Definition(offset=96, data_type=<_DataType.INT: 4>, default=0, array_s

nearest_neighbour_retry = _Definition(offset=41, data_type=<_DataType.BYTE: 1>, default=0, array_size=None,

netinit_bc_wait_time = _Definition(offset=44, data_type=<_DataType.BYTE: 1>, default=50, array_size=None, doc

netinit_phase = _Definition(offset=45, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The pha

offset

p2p_b_repeats = _Definition(offset=12, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Number

p2p_root_x = _Definition(offset=47, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The x-coordi

p2p_root_y = _Definition(offset=46, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The y-coordi

padding_1 = _Definition(offset=56, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A word of paddi

padding_2 = _Definition(offset=68, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A word of paddi

padding_3 = _Definition(offset=190, data_type=<_DataType.SHORT: 2>, default=0, array_size=None, doc='A short of p

padding_4 = _Definition(offset=252, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A word of padd

peer_to_peer_hop_table_address = _Definition(offset=216, data_type=<_DataType.INT: 4>, default=0, array_siz

physical_to_virtual_core_map = _Definition(offset=148, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x

random_seed = _Definition(offset=60, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The random s

router_table_copy_address = _Definition(offset=212, data_type=<_DataType.INT: 4>, default=0, array_size=None

router_time_phase_timer = _Definition(offset=32, data_type=<_DataType.INT: 4>, default=0, array_size=None, do

sdram_base_address = _Definition(offset=192, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='Th

sdram_clock_frequency_mhz = _Definition(offset=38, data_type=<_DataType.SHORT: 2>, default=130, array_size=

sdram_heap_address = _Definition(offset=76, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The

shared_message_buffer_address = _Definition(offset=232, data_type=<_DataType.INT: 4>, default=0, array_size=

shared_message_count_in_use = _Definition(offset=108, data_type=<_DataType.SHORT: 2>, default=0, array_siz

shared_message_first_free_address = _Definition(offset=104, data_type=<_DataType.INT: 4>, default=0, array

shared_message_maximum_used = _Definition(offset=110, data_type=<_DataType.SHORT: 2>, default=0, array_siz

software_watchdog_count = _Definition(offset=67, data_type=<_DataType.BYTE: 1>, default=3, array_size=None,

status_map = _Definition(offset=128, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x00\x00\x00\x00\x00\x00\x

system_buffer_words = _Definition(offset=88, data_type=<_DataType.INT: 4>, default=32768, array_size=None, doc

system_ram_base_address = _Definition(offset=196, data_type=<_DataType.INT: 4>, default=0, array_size=None, d

system_ram_heap_address = _Definition(offset=72, data_type=<_DataType.INT: 4>, default=1024, array_size=None

system_sdram_base_address = _Definition(offset=200, data_type=<_DataType.INT: 4>, default=0, array_size=None

system_sdram_bytes = _Definition(offset=84, data_type=<_DataType.INT: 4>, default=8388608, array_size=None, do

system_sdram_heap_address = _Definition(offset=208, data_type=<_DataType.INT: 4>, default=0, array_size=None

time_milliseconds = _Definition(offset=24, data_type=<_DataType.SHORT: 2>, default=0, array_size=None, doc='T

time_phase_scale = _Definition(offset=15, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The

unix_timestamp = _Definition(offset=28, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The time

user_temp_0 = _Definition(offset=112, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The first use

user_temp_1 = _Definition(offset=116, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The second u

user_temp_2 = _Definition(offset=120, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The third us

user_temp_4 = _Definition(offset=124, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The fourth u

virtual_to_physical_core_map = _Definition(offset=168, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x

x = _Definition(offset=1, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The x-coordinate of the ch

x_size = _Definition(offset=3, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The number of chip

y = _Definition(offset=0, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The y-coordinate of the ch

y_size = _Definition(offset=2, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The number of chip

Module contents

class spinnman.messages.spinnaker_boot.**SpinnakerBootMessage**(*opcode*, *operand_1*, *operand_2*, *operand_3*, *data=None*, *offset=0*)

Bases: object

A message used for booting the board

Parameters

- **opcode** (*spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.SpinnakerBootOpCode*) – The operation of this packet
- **operand_1** (*int*) – The first operand
- **operand_2** (*int*) – The second operand
- **operand_3** (*int*) – The third operand
- **data** (*str*) – The optional data, up to 256 words
- **offset** (*int*) – The offset of the valid data

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the opcode is not a valid value

bytestring

The message as a bytestring

data

The data

Returns The data or None if no data

Return type bytearray

static from_bytestring(*data*, *offset*)

opcode

The operation of this packet

Returns The operation code

Return type *spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.SpinnakerBootOpCode*

operand_1

The first operand

Returns The operand

Return type int

operand_2
The second operand

Returns The second operand

Return type int

operand_3
The third operand

Returns The third operand

Return type int

class spinnman.messages.spinnaker_boot.**SpinnakerBootMessages**(*board_version=None, extra_boot_values=None*)

Bases: object

Represents a set of boot messages to be sent to boot the board

builds the boot messages needed to boot the spinnaker machine

Parameters

- **board_version** (*int*) – The version of the board to be booted
- **extra_boot_values** (*dict of SystemVariableDefinition to value*) – Any additional values to be set during boot

Raises

- *spinnman.exceptions.SpinnmanInvalidParameterException* – If the board version is not supported
- *spinnman.exceptions.SpinnmanIOException* – If there is an error assembling the packets

messages
Get an iterable of message to be sent.

class spinnman.messages.spinnaker_boot.**SpinnakerBootOpCode**(*value, doc=''*)

Bases: enum.Enum

Boot message Operation Codes

FLOOD_FILL_BLOCK = 3

FLOOD_FILL_CONTROL = 5

FLOOD_FILL_START = 1

HELLO = 65

class spinnman.messages.spinnaker_boot.**SystemVariableDefinition**(*offset, data_type, default, array_size, doc*)

Bases: enum.Enum

Defines the system variables available

Parameters

- **data_type** (_DataType) – The data type of the variable

- **offset** (*int*) – The offset from the start of the system variable structure where the variable is found
- **default** (*int*) – The default value assigned to the variable if not overridden
- **array_size** (*int*) – The length of the array, or None if not an array

allocated_tag_table_address = _Definition(offset=220, data_type=<_DataType.INT: 4>, default=0, array_size=N

app_data_table_address = _Definition(offset=228, data_type=<_DataType.INT: 4>, default=0, array_size=None, do

array_size

board_info = _Definition(offset=248, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A pointer to t

board_test_flags = _Definition(offset=103, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Bo

boot_signature = _Definition(offset=92, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The boot

clock_divisor = _Definition(offset=14, data_type=<_DataType.BYTE: 1>, default=51, array_size=None, doc='The clo

clock_milliseconds = _Definition(offset=16, data_type=<_DataType.LONG: 8>, default=0, array_size=None, doc='T

cpu_clock_mhz = _Definition(offset=36, data_type=<_DataType.SHORT: 2>, default=200, array_size=None, doc='The

cpu_information_base_address = _Definition(offset=204, data_type=<_DataType.INT: 4>, default=0, array_size=N

data_type

debug_x = _Definition(offset=5, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The x-coordinate

debug_y = _Definition(offset=4, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The y-coordinate

default

ethernet_ip_address = _Definition(offset=240, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x00\x00\x00\x

first_free_router_entry = _Definition(offset=224, data_type=<_DataType.SHORT: 2>, default=0, array_size=Non

fixed_route_copy = _Definition(offset=244, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A (vi

hardware_version = _Definition(offset=10, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The

iobuf_size = _Definition(offset=80, data_type=<_DataType.INT: 4>, default=16384, array_size=None, doc='The size o

is_ethernet_available = _Definition(offset=11, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, do

is_peer_to_peer_available = _Definition(offset=6, data_type=<_DataType.BYTE: 1>, default=0, array_size=Non

is_root_chip = _Definition(offset=64, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Indicates

last_biff_id = _Definition(offset=102, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Last ID

led_0 = _Definition(offset=48, data_type=<_DataType.INT: 4>, default=1, array_size=None, doc='The first part of the L

led_1 = _Definition(offset=52, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The last part of the L

led_half_period_10_ms = _Definition(offset=43, data_type=<_DataType.BYTE: 1>, default=1, array_size=None, do

link_peek_timeout_microseconds = _Definition(offset=42, data_type=<_DataType.BYTE: 1>, default=100, array

links_available = _Definition(offset=101, data_type=<_DataType.BYTE: 1>, default=63, array_size=None, doc='Bit

lock = _Definition(offset=100, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The lock')

log_peer_to_peer_sequence_length = _Definition(offset=13, data_type=<_DataType.BYTE: 1>, default=4, array

ltpc_period = _Definition(offset=26, data_type=<_DataType.SHORT: 2>, default=0, array_size=None, doc='‘')

monitor_mailbox_flags = _Definition(offset=236, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc=

n_active_peer_to_peer_addresses = _Definition(offset=226, data_type=<_DataType.SHORT: 2>, default=0, arra

n_scamp_working_cores = _Definition(offset=189, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, d

n_shared_message_buffers = _Definition(offset=65, data_type=<_DataType.BYTE: 1>, default=7, array_size=None

n_working_cores = _Definition(offset=188, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The

nearest_ethernet_x = _Definition(offset=9, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Th

nearest_ethernet_y = _Definition(offset=8, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Th

nearest_neighbour_delay_us = _Definition(offset=66, data_type=<_DataType.BYTE: 1>, default=10, array_size=N

nearest_neighbour_forward = _Definition(offset=40, data_type=<_DataType.BYTE: 1>, default=63, array_size=No

nearest_neighbour_last_id = _Definition(offset=7, data_type=<_DataType.BYTE: 1>, default=0, array_size=None

nearest_neighbour_memory_pointer = _Definition(offset=96, data_type=<_DataType.INT: 4>, default=0, array_s

nearest_neighbour_retry = _Definition(offset=41, data_type=<_DataType.BYTE: 1>, default=0, array_size=None,

netinit_bc_wait_time = _Definition(offset=44, data_type=<_DataType.BYTE: 1>, default=50, array_size=None, doc

netinit_phase = _Definition(offset=45, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The pha

offset

p2p_b_repeats = _Definition(offset=12, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='Number

p2p_root_x = _Definition(offset=47, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The x-coordi

p2p_root_y = _Definition(offset=46, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The y-coordi

padding_1 = _Definition(offset=56, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A word of paddi

padding_2 = _Definition(offset=68, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A word of paddi

padding_3 = _Definition(offset=190, data_type=<_DataType.SHORT: 2>, default=0, array_size=None, doc='A short of p

padding_4 = _Definition(offset=252, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='A word of padd

peer_to_peer_hop_table_address = _Definition(offset=216, data_type=<_DataType.INT: 4>, default=0, array_siz

physical_to_virtual_core_map = _Definition(offset=148, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x

random_seed = _Definition(offset=60, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The random s

router_table_copy_address = _Definition(offset=212, data_type=<_DataType.INT: 4>, default=0, array_size=None

router_time_phase_timer = _Definition(offset=32, data_type=<_DataType.INT: 4>, default=0, array_size=None, do

sdram_base_address = _Definition(offset=192, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='Th

sdram_clock_frequency_mhz = _Definition(offset=38, data_type=<_DataType.SHORT: 2>, default=130, array_size=

sdram_heap_address = _Definition(offset=76, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The

shared_message_buffer_address = _Definition(offset=232, data_type=<_DataType.INT: 4>, default=0, array_size=

shared_message_count_in_use = _Definition(offset=108, data_type=<_DataType.SHORT: 2>, default=0, array_size

shared_message_first_free_address = _Definition(offset=104, data_type=<_DataType.INT: 4>, default=0, array

shared_message_maximum_used = _Definition(offset=110, data_type=<_DataType.SHORT: 2>, default=0, array_size

software_watchdog_count = _Definition(offset=67, data_type=<_DataType.BYTE: 1>, default=3, array_size=None,

status_map = _Definition(offset=128, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x00\x00\x00\x00\x00\x00\x

system_buffer_words = _Definition(offset=88, data_type=<_DataType.INT: 4>, default=32768, array_size=None, doc

system_ram_base_address = _Definition(offset=196, data_type=<_DataType.INT: 4>, default=0, array_size=None, d

system_ram_heap_address = _Definition(offset=72, data_type=<_DataType.INT: 4>, default=1024, array_size=None

system_sdram_base_address = _Definition(offset=200, data_type=<_DataType.INT: 4>, default=0, array_size=None

system_sdram_bytes = _Definition(offset=84, data_type=<_DataType.INT: 4>, default=8388608, array_size=None, do

system_sdram_heap_address = _Definition(offset=208, data_type=<_DataType.INT: 4>, default=0, array_size=None

time_milliseconds = _Definition(offset=24, data_type=<_DataType.SHORT: 2>, default=0, array_size=None, doc='T

time_phase_scale = _Definition(offset=15, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The

unix_timestamp = _Definition(offset=28, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The time

user_temp_0 = _Definition(offset=112, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The first use

user_temp_1 = _Definition(offset=116, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The second u

user_temp_2 = _Definition(offset=120, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The third us

user_temp_4 = _Definition(offset=124, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The fourth u

virtual_to_physical_core_map = _Definition(offset=168, data_type=<_DataType.BYTE_ARRAY: 16>, default='\x

x = _Definition(offset=1, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The x-coordinate of the ch

x_size = _Definition(offset=3, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The number of chip

y = _Definition(offset=0, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The y-coordinate of the ch

y_size = _Definition(offset=2, data_type=<_DataType.BYTE: 1>, default=0, array_size=None, doc='The number of chip

Submodules

spinnman.messages.multicast_message module

class spinnman.messages.multicast_message.**MulticastMessage**(*key*, *payload=None*)

Bases: object

A SpiNNaker Multicast message

A multicast message has a key (determining the target locations) and an optional payload.

Parameters

- **key** (*int*) – The key of the packet
- **payload** (*int*) – The optional payload of the packet

Raises **None** – No known exceptions are raised

key

The key of the packet

Returns The key

Return type int

payload

The payload of the packet if there is one

Returns The payload, or None if there is no payload

Return type int

Module contents

spinnman.model package

Subpackages

spinnman.model.enums package

Submodules

spinnman.model.enums.cpu_state module

class `spinnman.model.enums.cpu_state.`**`CPUState`**(*value*, *doc*='')

Bases: `enum.Enum`

SARK CPU States

CPU_STATE_12 = 12

CPU_STATE_13 = 13

CPU_STATE_14 = 14

C_MAIN = 6

DEAD = 0

FINISHED = 11

IDLE = 15

INITIALISING = 4

PAUSED = 10

POWERED_DOWN = 1

READY = 5

RUNNING = 7

RUN_TIME_EXCEPTION = 2

SYNC0 = 8

SYNC1 = 9

WATCHDOG = 3

spinnman.model.enums.diagnostic_filter_default_routing_status module

class `spinnman.model.enums.diagnostic_filter_default_routing_status.`**`DiagnosticFilterDefaultRo`**

Bases: `enum.Enum`

Default routing flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

DEFAULT_ROUTED = 0

NON_DEFAULT_ROUTED = 1

spinnman.model.enums.diagnostic_filter_destination module

class spinnman.model.enums.diagnostic_filter_destination.**DiagnosticFilterDestination**(*value*, *doc=''*)

Bases: enum.Enum

Destination flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

DUMP = 0

LINK_0 = 3

LINK_1 = 4

LINK_2 = 5

LINK_3 = 6

LINK_4 = 7

LINK_5 = 8

LOCAL = 1

LOCAL_MONITOR = 2

spinnman.model.enums.diagnostic_filter_emergency_routing_status module

class spinnman.model.enums.diagnostic_filter_emergency_routing_status.**DiagnosticFilterEmerger**

Bases: enum.Enum

Emergency routing status flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

FIRST_STAGE = 2

FIRST_STAGE_COMBINED = 1

NORMAL = 0

SECOND_STAGE = 3

spinnman.model.enums.diagnostic_filter_packet_type module

class spinnman.model.enums.diagnostic_filter_packet_type.**DiagnosticFilterPacketType**(*value*, *doc=''*)

Bases: enum.Enum

Packet type flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

FIXED_ROUTE = 3

MULTICAST = 0

NEAREST_NEIGHBOUR = 2

POINT_TO_POINT = 1

spinnman.model.enums.diagnostic_filter_payload_status module

class spinnman.model.enums.diagnostic_filter_payload_status.**DiagnosticFilterPayloadStatus**(*valu*
doc

Bases: enum.Enum

Payload flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

WITHOUT_PAYLOAD = 1

WITH_PAYLOAD = 0

spinnman.model.enums.diagnostic_filter_source module

class spinnman.model.enums.diagnostic_filter_source.**DiagnosticFilterSource**(*value*, *doc=''*)

Bases: enum.Enum

Source flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

LOCAL = 0

NON_LOCAL = 1

spinnman.model.enums.mailbox_command module

class spinnman.model.enums.mailbox_command.**MailboxCommand**(*value*, *doc=''*)

Bases: enum.Enum

Commands sent between an application and the monitor processor

SHM_CMD = 4

SHM_IDLE = 0

SHM_MSG = 1

SHM_NOP = 2

SHM_SIGNAL = 3

spinnman.model.enums.p2p_table_route module

class spinnman.model.enums.p2p_table_route.**P2PTableRoute**(*value*, *doc=''*)

Bases: enum.Enum

P2P Routing table routes

EAST = 0

MONITOR = 7

NONE = 6

NORTH = 2

NORTH_EAST = 1

SOUTH = 5

SOUTH_WEST = 4

WEST = 3

spinnman.model.enums.run_time_error module

class spinnman.model.enums.run_time_error.**RunTimeError**(*value*, *doc=''*)

Bases: enum.Enum

SARK Run time errors

ABORT = 9

API = 19

DABT = 5

DIVBY0 = 11

ENABLE = 15

EVENT = 12

FIQ = 7

IOBUF = 14

IRQ = 6

MALLOC = 10

NONE = 0

NULL = 16

PABT = 4

PKT = 17

RESET = 1

SARK_VERSRION_INCORRECT = 20

SVC = 3

SWERR = 13

TIMER = 18

UNDEF = 2

VIC = 8

Module contents

class spinnman.model.enums.**CPUState**(*value*, *doc=''*)

Bases: enum.Enum

SARK CPU States

CPU_STATE_12 = 12

CPU_STATE_13 = 13

CPU_STATE_14 = 14

C_MAIN = 6

DEAD = 0

FINISHED = 11

IDLE = 15

INITIALISING = 4

PAUSED = 10

POWERED_DOWN = 1

READY = 5

RUNNING = 7

RUN_TIME_EXCEPTION = 2

SYNC0 = 8

SYNC1 = 9

WATCHDOG = 3

class spinnman.model.enums.**DiagnosticFilterDefaultRoutingStatus**(*value*, *doc=''*)

Bases: enum.Enum

Default routing flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

DEFAULT_ROUTED = 0

NON_DEFAULT_ROUTED = 1

class spinnman.model.enums.**DiagnosticFilterDestination**(*value*, *doc=''*)

Bases: enum.Enum

Destination flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

DUMP = 0

LINK_0 = 3

LINK_1 = 4

LINK_2 = 5

LINK_3 = 6

LINK_4 = 7

LINK_5 = 8

LOCAL = 1

LOCAL_MONITOR = 2

class spinnman.model.enums.**RunTimeError**(*value*, *doc=''*)

Bases: enum.Enum

SARK Run time errors

ABORT = 9

API = 19

DABT = 5

DIVBY0 = 11

ENABLE = 15

EVENT = 12

FIQ = 7

IOBUF = 14

IRQ = 6

MALLOC = 10

NONE = 0

NULL = 16

PABT = 4

PKT = 17

RESET = 1

SARK_VERSRION_INCORRECT = 20

SVC = 3

SWERR = 13

TIMER = 18

UNDEF = 2

VIC = 8

class spinnman.model.enums.**DiagnosticFilterEmergencyRoutingStatus**(*value*, *doc=''*)
Bases: enum.Enum

Emergency routing status flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

FIRST_STAGE = 2

FIRST_STAGE_COMBINED = 1

NORMAL = 0

SECOND_STAGE = 3

class spinnman.model.enums.**DiagnosticFilterPacketType**(*value*, *doc=''*)
Bases: enum.Enum

Packet type flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

FIXED_ROUTE = 3

MULTICAST = 0

NEAREST_NEIGHBOUR = 2

POINT_TO_POINT = 1

class spinnman.model.enums.**DiagnosticFilterPayloadStatus**(*value*, *doc=''*)
Bases: enum.Enum

Payload flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

WITHOUT_PAYLOAD = 1

WITH_PAYLOAD = 0

class spinnman.model.enums.**DiagnosticFilterSource**(*value*, *doc=''*)
Bases: enum.Enum

Source flags for the diagnostic filters. Note that only one has to match for the counter to be incremented

LOCAL = 0

NON_LOCAL = 1

class spinnman.model.enums.**MailboxCommand**(*value*, *doc=''*)
Bases: enum.Enum

Commands sent between an application and the monitor processor

SHM_CMD = 4

SHM_IDLE = 0

SHM_MSG = 1

SHM_NOP = 2

SHM_SIGNAL = 3

class spinnman.model.enums.**P2PTableRoute**(*value*, *doc=''*)
Bases: enum.Enum

P2P Routing table routes

EAST = 0

MONITOR = 7

NONE = 6

NORTH = 2

NORTH_EAST = 1

SOUTH = 5

SOUTH_WEST = 4

WEST = 3

Submodules

spinnman.model.adc_info module

ADCInfo

class spinnman.model.adc_info.**ADCInfo**(*adc_data*, *offset*)
Bases: object

container for the ADC data thats been retrieved from a fpga

Parameters **adc_data** (*str*) – bytes from an SCP packet containing adc information

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the message does not contain valid adc information

fan_0
property for fan 0

fan_1
property for fan 1

temp_btm
property for temp btm

temp_ext_0
property for temp ext 0

temp_ext_1
property for temp_ext_1

temp_top
property for temp_top

voltage_1_2a
property for voltage 1 2a

voltage_1_2b
property for voltage 1 2b

voltage_1_2c
property for voltage 1 2c

voltage_1_8
property for voltage 1 8

voltage_3_3
property for voltage 3 3

voltage_supply
property for voltage supply

spinnman.model.bmp_connection_data module

class `spinnman.model.bmp_connection_data.`**`BMPConnectionData`**(*cabinet*, *frame*, *ip_address*, *boards*, *port_num*)

Bases: `object`

Contains the details of a BMP connection

boards
Iterable of integers for the boards to be addressed

Return type iterable of int

cabinet
Get the cabinet number

Return type int

frame
Get the frame number

Return type int

ip_address
Get the IP address of the BMP

Return type str

port_num
The port number associated with this BMP connection

Returns The port number

spinnman.model.chip_info module

class `spinnman.model.chip_info.`**`ChipInfo`**(*system_data*, *offset*)

Bases: `object`

Represents the system variables for a chip, received from the chip SDRAM

Parameters

- **`system_data`** (*`str`*) – An bytestring retrieved from SDRAM on the board
- **`offset`** – The offset into the bytestring where the actual data starts

Raises *`spinnman.exceptions.SpinnmanInvalidParameterException`* – If the data doesn't contain valid system data information

`cpu_clock_mhz`

The speed of the CPU clock in MHz

Return type int

`cpu_information_base_address`

The base address of the cpu information structure

Return type int

`first_free_router_entry`

The id of the first free routing entry on the chip

Return type int

`iobuf_size`

The size of the iobuf buffers in bytes

Return type int

`ip_address`

The ip address of the chip, or None if no Ethernet

Return type str

`is_ethernet_available`

True if the Ethernet is running on this chip, False otherwise

Return type bool

`links_available`

The links that are available on the chip

Return type iterable of int

`nearest_ethernet_x`

The x-coordinate of the nearest chip with Ethernet

Return type int

`nearest_ethernet_y`

The y-coordinate of the nearest chip with Ethernet

Return type int

`physical_to_virtual_core_map`

The physical core id to virtual core id map; entries with a value of 0xFF are non-operational cores

Return type bytearray

router_table_copy_address()
The address of the copy of the router table

Return type int

sdram_base_address
The base address of the user region of SDRAM on the chip

Return type int

sdram_heap_address
The address of the base of the heap in SDRAM

Return type int

system_ram_heap_address
The address of the base of the heap in system RAM

Return type int

system_sdram_base_address
The base address of the System SDRAM region on the chip

Return type int

virtual_core_ids
A list of available cores by virtual core id (including the monitor)

Return type iterable of int

x
The x-coordinate of the chip

Return type int

x_size
The number of chips in the x-dimension

Return type int

y
The y-coordinate of the chip

Return type int

y_size
The number of chips in the y-dimension

Return type int

spinnman.model.chip_summary_info module

class spinnman.model.chip_summary_info.**ChipSummaryInfo**(*chip_summary_data*, *offset*, *x*, *y*)

Bases: object

Represents the chip summary information read via an SCP command

Parameters

- **chip_summary_data** (*bytearray*) – The data from the SCP response
- **offset** (*int*) – The offset into the data where the data starts
- **x** – The x-coordinate of the chip that this data is from

- **y** – The y-coordinate of the chip that this data is from

core_states
The state of the cores on the chip (list of one per core)

Return type list of *py:class:spinnman.model.cpu_state.CPUState*

ethernet_ip_address
The IP address of the Ethernet if up, or None if not

Return type str

is_ethernet_available
Determines if the Ethernet connection is available on this chip

Return type bool

largest_free_sdram_block
The size of the largest block of free SDRAM in bytes

Return type int

largest_free_sram_block
The size of the largest block of free SRAM in bytes

Return type int

n_cores
The number of cores working on the chip (including monitors)

Return type int

n_free_multicast_routing_entries
The number of multicast routing entries free on this chip

Return type int

nearest_ethernet_x
The x coordinate of the nearest Ethernet chip

Return type int

nearest_ethernet_y
The y coordinate of the nearest Ethernet chip

Return type int

working_links
The ids of the working links outgoing from this chip

Return type list of int

x
The x-coordinate of the chip that this data is from

Return type int

y
The y-coordinate of the chip that this data is from

Return type int

spinnman.model.cpu_info module

class `spinnman.model.cpu_info.`**`CPUInfo`**(*x*, *y*, *p*, *cpu_data*, *offset*)

Bases: `object`

Represents information about the state of a CPU

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip
- **p** (*int*) – The id of a core on the chip
- **cpu_data** (*str*) – A bytestring received from SDRAM on the board

application_id

The id of the application running on the core

Returns The id of the application

Return type int

application_mailbox_command

The command currently in the mailbox being sent from the monitor processor to the application

Returns The command

Return type *spinnman.model.enums.mailbox_command.MailboxCommand*

application_mailbox_data_address

The address of the data in SDRAM for the application mailbox

Returns The address of the data

Return type int

application_name

The name of the application running on the core

Returns The name of the application

Return type str

iobuf_address

The address of the IOBUF buffer in SDRAM

Returns The address

Return type int

link_register

The current link register value (LR)

Returns The LR value

Return type int

monitor_mailbox_command

The command currently in the mailbox being sent from the application to the monitor processor

Returns The command

Return type `spinnman.model.mailbox_command.MailboxCommand`

monitor_mailbox_data_address

The address of the data in SDRAM of the monitor mailbox

Returns The address of the data

Return type int

p

The id of the core on the chip

Returns The id of the core

Return type int

physical_cpu_id

The physical id of this processor

Returns The physical id of the processor

Return type int

processor_state_register

The value in the processor state register (PSR)

Returns The PSR value

Return type int

registers

The current register values (r0 - r7)

Returns An array of 8 values, one for each register

Return type array of int

run_time_error

The reason for a run time error

Returns The run time error

Return type *spinnman.model.enums.run_time_error.RunTimeError*

software_error_count

The number of software errors counted

Returns The number of software errors

Return type int

software_source_filename_address

The address of the filename of the software source

Returns The filename

Return type str

software_source_line_number

The line number of the software source

Returns The line number

Return type int

software_version

The software version

Returns The software version

Return type int

stack_pointer
The current stack pointer value (SP)

Returns The SP value

Return type int

state
The current state of the core

Returns The state of the core

Return type *spinnman.model.enums.cpu_state.CPUState*

time
The time at which the application started

Returns The time in seconds since 00:00:00 on the 1st January 1970

Return type long

user
The current user values (user0 - user3)

Returns An array of 4 values, one for each user value

Return type array of int

x
The x-coordinate of the chip containing the core

Returns The x-coordinate of the chip

Return type int

y
The y-coordinate of the chip containing the core

Returns The y-coordinate of the chip

Return type int

spinnman.model.cpu_infos module

class spinnman.model.cpu_infos.**CPUInfos**
Bases: object

A set of CPU information objects

add_processor(*x*, *y*, *processor_id*, *cpu_info*)
Add a processor on a given chip to the set

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip
- **processor_id** (*int*) – A processor id
- **cpu_info** (spinnman.model.enums.cpu_info.CPUInfo) – The CPU information for the core

cpu_infos
the one per core core info

Returns iterable of x,y,p core info

iteritems()
Get an iterable of (x, y, p), cpu_info

iterkeys()
Get an iterable of (x, y, p)

itervalues()
Get an iterable of cpu_info

spinnman.model.diagnostic_filter module

class spinnman.model.diagnostic_filter.**DiagnosticFilter**(*enable_interrupt_on_counter_event*, *match_emergency_routing_status_to_incoming_packet*, *destinations*, *sources*, *payload_statuses*, *default_routing_statuses*, *emergency_routing_statuses*, *packet_types*)

Bases: object

A router diagnostic counter filter, which counts packets passing through the router with certain properties. The counter will be incremented so long as the packet matches one of the values in each field i.e. one of each of the destinations, sources, payload_statuses, default_routing_statuses, emergency_routing_statuses and packet_types

Parameters

- **enable_interrupt_on_counter_event** (*bool*) – Indicates whether an interrupt should be raised when this rule matches
- **match_emergency_routing_status_to_incoming_packet** (*bool*) – Indicates whether the emergency routing statuses should be matched against packets arriving at this router (if True), or if they should be matched against packets leaving this router (if False)
- **destinations** (iterable of spinnman.model.diagnostic_filter_destination.DiagnosticFilterDestination) – Increment the counter if one or more of the given destinations match
- **sources** (iterable of spinnman.model.diagnostic_filter_source.DiagnosticFilterSource) – Increment the counter if one or more of the given sources match (or None or empty list to match all)
- **payload_statuses** (iterable of spinnman.model.diagnostic_filter_payload_status.DiagnosticFilterPayloadStatus) – Increment the counter if one or more of the given payload statuses match (or None or empty list to match all)
- **default_routing_statuses** (iterable of spinnman.model.diagnostic_filter_default_routing_status.DiagnosticFilterDefaultRoutingStatus) – Increment the counter if one or more of the given default routing statuses match (or None or empty list to match all)
- **emergency_routing_statuses** (iterable of spinnman.model.diagnostic_filter_emergency_routing_status.DiagnosticFilterEmergencyRoutingStatus) – Increment the counter if

one or more of the given emergency routing statuses match (or None or empty list to match all)

- **packet_types** (iterable of `spinnman.model.diagnostic_filter_packet_type.DiagnosticFilterPacketType`) – Increment the counter if one or more of the given packet types match (or None or empty list to match all)

default_routing_statuses

destinations

emergency_routing_statuses

enable_interrupt_on_counter_event

filter_word
A word of data that can be written to the router to set up the filter

match_emergency_routing_status_to_incoming_packet

packet_types

payload_statuses

static **read_from_int**(*int_value*)

sources

spinnman.model.executable_targets module

class `spinnman.model.executable_targets.`**ExecutableTargets**
Bases: `object`

Encapsulate the binaries and cores on which to execute them

add_processor(*binary*, *chip_x*, *chip_y*, *chip_p*)
Add a processor to the executable targets

Parameters

- **binary** – the binary path for executable
- **chip_x** – the coordinate on the machine in terms of x for the chip
- **chip_y** – the coordinate on the machine in terms of y for the chip
- **chip_p** – the processor id to place this executable on

Returns

add_subsets(*binary*, *subsets*)
Add core subsets to a binary

Parameters

- **binary** – the path to the binary needed to be executed
- **subsets** – the subset of cores that the binary needs to be loaded on

Returns

all_core_subsets
All the core subsets for all the binaries

binaries
The binaries of the executables

get_cores_for_binary(*binary*)
Get the cores that a binary is to run on

Parameters **binary** – The binary to find the cores for

total_processors
The total number of cores to be loaded

spinnman.model.heap_element module

class spinnman.model.heap_element.**HeapElement**(*block_address*, *next_address*, *free*)
Bases: object

An element of one of the heaps on SpiNNaker

Parameters

- **block_address** – The address of this element on the heap
- **next_address** – The address of the next element on the heap
- **free** – The "free" element of the block as read from the heap

app_id
The app id of the block if allocated, or None if not

block_address
The address of the block

is_free
True if this block is a free block, False otherwise

next_address
The address of the next block, or 0 if none

size
The usable size of this block (not including the header)

tag
The tag of the block if allocated, or None if not

spinnman.model.io_buffer module

class spinnman.model.io_buffer.**IOBuffer**(*x*, *y*, *p*, *iobuf*)
Bases: object

The contents of IOBUF for a core

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip
- **p** (*int*) – The p-coordinate of a chip
- **iobuf** (*str*) – The contents of the buffer for the chip

Raises **None** – No known exceptions are raised

iobuf
The contents of the buffer

Returns The contents of the buffer

Return type str

p

The id of the core on the chip

Returns The id of the core

Return type int

x

The x-coordinate of the chip containing the core

Returns The x-coordinate of the chip

Return type int

y

The y-coordinate of the chip containing the core

Returns The y-coordinate of the chip

Return type int

spinnman.model.machine_dimensions module

class `spinnman.model.machine_dimensions.`**`MachineDimensions`**(*width*, *height*)

Bases: `object`

Represents the size of a machine in chips

Parameters

- **width** (*int*) – The width of the machine in chips
- **height** (*int*) – The height of the machine in chips

Raises **None** – No known exceptions are raised

height

The height of the machine in chips

Returns The height

Return type int

width

The width of the machine in chips

Returns The width

Return type int

spinnman.model.p2p_table module

class `spinnman.model.p2p_table.`**`P2PTable`**(*width*, *height*, *column_data*)

Bases: `object`

Represents a P2P table read from the machine

static get_column_offset(*column*)

Get the offset of the next column in the table from the P2P base address

Parameters **column** – The column to be read

static **get_n_column_bytes**(*height*)
Get the number of bytes to be read for each column of the table

Parameters **height** – The height of the machine

get_route(*x*, *y*)
Get the route to follow from this chip to the given chip

Parameters

- **x** – The x-coordinate of the chip to find the route to
- **y** – The y-coordinate of the chip to find the route to

height
The height of the machine that this table represents

is_route(*x*, *y*)
Determines if there is a route in the P2P table to the given chip

Parameters

- **x** – The x-coordinate of the chip to look up
- **y** – The y-coordinate of the chip to look up

iterchips()
Get an iterator of tuples of (x, y) coordinates in the table

width
The width of the machine that this table represents

spinnman.model.router_diagnostics module

class spinnman.model.router_diagnostics.**RouterDiagnostics**(*control_register*, *error_status*, *register_values*)

Bases: object

Represents a set of diagnostic information available from a chip router

Parameters

- **control_register** (*int*) – The value of the control register
- **error_status** (*int*) – The value of the error_status
- **register_values** (*iterable of int*) – The values of the 16 router registers

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the number of register values is not 16

error_status
The error status

Returns The error status

Return type int

mon
The "mon" part of the control register

Returns The mon bits

Return type int

n_dropped_fixed_route_packets
The number of fixed-route packets received that were dropped

Returns The number of packets

Return type int

n_dropped_multicast_packets
The number of multicast packets received that were dropped

Returns The number of packets

Return type int

n_dropped_nearest_neighbour_packets
The number of nearest-neighbour packets received that were dropped

Returns The number of packets

Return type int

n_dropped_peer_to_peer_packets
The number of peer-to-peer packets received that were dropped

Returns The number of packets

Return type int

n_external_fixed_route_packets
The number of fixed-route packets received from external links

Returns The number of packets

Return type int

n_external_multicast_packets
The number of multicast packets received from external links

Returns The number of packets

Return type int

n_external_nearest_neighbour_packets
The number of nearest-neighbour packets received from external links

Returns The number of packets

Return type int

n_external_peer_to_peer_packets
The number of peer-to-peer packets received from external links

Returns The number of packets

Return type int

n_local_fixed_route_packets
The number of fixed-route packets received from local cores

Returns The number of packets

Return type int

n_local_multicast_packets
The number of multicast packets received from local cores

Returns The number of packets

Return type int

n_local_nearest_neighbour_packets
The number of nearest-neighbour packets received from local cores

Returns The number of packets

Return type int

n_local_peer_to_peer_packets
The number of peer-to-peer packets received from local cores

Returns The number of packets

Return type int

registers
The values in all of the registers. Can be used to directly access the registers if they have been programmed to give different values

Returns An array of 16 values

Return type array of int

user_0
the data gained from the user 0 router diag filter

Returns the number of packets captured by this filter.

user_1
the data gained from the user 1 router diag filter

Returns the number of packets captured by this filter.

user_2
the data gained from the user 2 router diag filter

Returns the number of packets captured by this filter.

user_3
the data gained from the user 3 router diag filter

Returns the number of packets captured by this filter.

user_registers
The values in the user control registers

Returns An array of 4 values

Return type array of int

wait_1
The wait_1 part of the control register

Returns The wait_1 bits

Return type int

wait_2
The wait_2 part of the control register

Returns The wait_2 bits

Return type int

spinnman.model.version_info module

class `spinnman.model.version_info.`**`VersionInfo`**(*version_data*, *offset=0*)

Bases: `object`

Decodes SC&MP/SARK version information as returned by the SVER command

Parameters

- **`version_data`** (*`bytearray`*) – bytes from an SCP packet containing version information
- **`offset`** – the offset in the bytes from an SCP packet containing version information

Raises *`spinnman.exceptions.SpinnmanInvalidParameterException`* – If the message does not contain valid version information

`build_date`

The build date of the software

Returns The number of seconds since 1st January 1970

Return type long

`hardware`

The hardware being run on

Returns The hardware

Return type str

`name`

The name of the software

Returns The name

Return type str

`p`

The processor id of the processor where the information was obtained

Returns the processor id

Return type int

`version_number`

The version number of the software

Returns The version

Return type float

`version_string`

The version information as text

Returns The version information

Return type str

`x`

The x-coordinate of the chip where the information was obtained

Returns the x-coordinate

Return type int

y
The y-coordinate of the chip where the information was obtained

Returns The y-coordinate

Return type int

Module contents

class spinnman.model.**ADCInfo**(*adc_data*, *offset*)
Bases: object

container for the ADC data thats been retrieved from a fpga

Parameters **adc_data** (*str*) – bytes from an SCP packet containing adc information

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the message does not contain valid adc information

fan_0
property for fan 0

fan_1
property for fan 1

temp_btm
property for temp btm

temp_ext_0
property for temp ext 0

temp_ext_1
property for temp_ext_1

temp_top
property for temp_top

voltage_1_2a
property for voltage 1 2a

voltage_1_2b
property for voltage 1 2b

voltage_1_2c
property for voltage 1 2c

voltage_1_8
property for voltage 1 8

voltage_3_3
property for voltage 3 3

voltage_supply
property for voltage supply

class spinnman.model.**BMPConnectionData**(*cabinet*, *frame*, *ip_address*, *boards*, *port_num*)
Bases: object

Contains the details of a BMP connection

boards
Iterable of integers for the boards to be addressed

Return type iterable of int

cabinet
Get the cabinet number

Return type int

frame
Get the frame number

Return type int

ip_address
Get the IP address of the BMP

Return type str

port_num
The port number associated with this BMP connection

Returns The port number

class `spinnman.model.ChipInfo`(*system_data*, *offset*)
Bases: `object`

Represents the system variables for a chip, received from the chip SDRAM

Parameters

- **system_data** (*str*) – An bytestring retrieved from SDRAM on the board
- **offset** – The offset into the bytestring where the actual data starts

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the data doesn't contain valid system data information

cpu_clock_mhz
The speed of the CPU clock in MHz

Return type int

cpu_information_base_address
The base address of the cpu information structure

Return type int

first_free_router_entry
The id of the first free routing entry on the chip

Return type int

iobuf_size
The size of the iobuf buffers in bytes

Return type int

ip_address
The ip address of the chip, or None if no Ethernet

Return type str

is_ethernet_available
True if the Ethernet is running on this chip, False otherwise

Return type bool

links_available
The links that are available on the chip

Return type iterable of int

nearest_ethernet_x
The x-coordinate of the nearest chip with Ethernet

Return type int

nearest_ethernet_y
The y-coordinate of the nearest chip with Ethernet

Return type int

physical_to_virtual_core_map
The physical core id to virtual core id map; entries with a value of 0xFF are non-operational cores

Return type bytearray

router_table_copy_address()
The address of the copy of the router table

Return type int

sdram_base_address
The base address of the user region of SDRAM on the chip

Return type int

sdram_heap_address
The address of the base of the heap in SDRAM

Return type int

system_ram_heap_address
The address of the base of the heap in system RAM

Return type int

system_sdram_base_address
The base address of the System SDRAM region on the chip

Return type int

virtual_core_ids
A list of available cores by virtual core id (including the monitor)

Return type iterable of int

x
The x-coordinate of the chip

Return type int

x_size
The number of chips in the x-dimension

Return type int

y
The y-coordinate of the chip

Return type int

y_size
The number of chips in the y-dimension

Return type int

class `spinnman.model.`**`ChipSummaryInfo`**(*chip_summary_data*, *offset*, *x*, *y*)

Bases: `object`

Represents the chip summary information read via an SCP command

Parameters

- **`chip_summary_data`** (*`bytearray`*) – The data from the SCP response
- **`offset`** (*`int`*) – The offset into the data where the data starts
- **`x`** – The x-coordinate of the chip that this data is from
- **`y`** – The y-coordinate of the chip that this data is from

`core_states`

The state of the cores on the chip (list of one per core)

Return type list of *py:class:spinnman.model.cpu_state.CPUState*

`ethernet_ip_address`

The IP address of the Ethernet if up, or None if not

Return type str

`is_ethernet_available`

Determines if the Ethernet connection is available on this chip

Return type bool

`largest_free_sdram_block`

The size of the largest block of free SDRAM in bytes

Return type int

`largest_free_sram_block`

The size of the largest block of free SRAM in bytes

Return type int

`n_cores`

The number of cores working on the chip (including monitors)

Return type int

`n_free_multicast_routing_entries`

The number of multicast routing entries free on this chip

Return type int

`nearest_ethernet_x`

The x coordinate of the nearest Ethernet chip

Return type int

`nearest_ethernet_y`

The y coordinate of the nearest Ethernet chip

Return type int

`working_links`

The ids of the working links outgoing from this chip

Return type list of int

x
The x-coordinate of the chip that this data is from

Return type int

y
The y-coordinate of the chip that this data is from

Return type int

class `spinnman.model.`**`CPUInfo`**(*x*, *y*, *p*, *cpu_data*, *offset*)
Bases: `object`

Represents information about the state of a CPU

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip
- **p** (*int*) – The id of a core on the chip
- **cpu_data** (*str*) – A bytestring received from SDRAM on the board

application_id
The id of the application running on the core

Returns The id of the application

Return type int

application_mailbox_command
The command currently in the mailbox being sent from the monitor processor to the application

Returns The command

Return type *spinnman.model.enums.mailbox_command.MailboxCommand*

application_mailbox_data_address
The address of the data in SDRAM for the application mailbox

Returns The address of the data

Return type int

application_name
The name of the application running on the core

Returns The name of the application

Return type str

iobuf_address
The address of the IOBUF buffer in SDRAM

Returns The address

Return type int

link_register
The current link register value (LR)

Returns The LR value

Return type int

monitor_mailbox_command
The command currently in the mailbox being sent from the application to the monitor processor

Returns The command

Return type `spinnman.model.mailbox_command.MailboxCommand`

monitor_mailbox_data_address
The address of the data in SDRAM of the monitor mailbox

Returns The address of the data

Return type int

p
The id of the core on the chip

Returns The id of the core

Return type int

physical_cpu_id
The physical id of this processor

Returns The physical id of the processor

Return type int

processor_state_register
The value in the processor state register (PSR)

Returns The PSR value

Return type int

registers
The current register values (r0 - r7)

Returns An array of 8 values, one for each register

Return type array of int

run_time_error
The reason for a run time error

Returns The run time error

Return type *spinnman.model.enums.run_time_error.RunTimeError*

software_error_count
The number of software errors counted

Returns The number of software errors

Return type int

software_source_filename_address
The address of the filename of the software source

Returns The filename

Return type str

software_source_line_number
The line number of the software source

Returns The line number

Return type int

software_version
The software version

Returns The software version

Return type int

stack_pointer
The current stack pointer value (SP)

Returns The SP value

Return type int

state
The current state of the core

Returns The state of the core

Return type *spinnman.model.enums.cpu_state.CPUState*

time
The time at which the application started

Returns The time in seconds since 00:00:00 on the 1st January 1970

Return type long

user
The current user values (user0 - user3)

Returns An array of 4 values, one for each user value

Return type array of int

x
The x-coordinate of the chip containing the core

Returns The x-coordinate of the chip

Return type int

y
The y-coordinate of the chip containing the core

Returns The y-coordinate of the chip

Return type int

class spinnman.model.**CPUInfos**
Bases: object

A set of CPU information objects

add_processor(*x*, *y*, *processor_id*, *cpu_info*)
Add a processor on a given chip to the set

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip
- **processor_id** (*int*) – A processor id

- **cpu_info** (spinnman.model.enums.cpu_info.CPUInfo) – The CPU information for the core

cpu_infos
the one per core core info

Returns iterable of x,y,p core info

iteritems()
Get an iterable of (x, y, p), cpu_info

iterkeys()
Get an iterable of (x, y, p)

itervalues()
Get an iterable of cpu_info

class spinnman.model.**DiagnosticFilter**(*enable_interrupt_on_counter_event*, *match_emergency_routing_status_to_incoming_packet*, *destinations*, *sources*, *payload_statuses*, *default_routing_statuses*, *emergency_routing_statuses*, *packet_types*)

Bases: object

A router diagnostic counter filter, which counts packets passing through the router with certain properties. The counter will be incremented so long as the packet matches one of the values in each field i.e. one of each of the destinations, sources, payload_statuses, default_routing_statuses, emergency_routing_statuses and packet_types

Parameters

- **enable_interrupt_on_counter_event** (*bool*) – Indicates whether an interrupt should be raised when this rule matches
- **match_emergency_routing_status_to_incoming_packet** (*bool*) – Indicates whether the emergency routing statuses should be matched against packets arriving at this router (if True), or if they should be matched against packets leaving this router (if False)
- **destinations** (iterable of spinnman.model.diagnostic_filter_destination.DiagnosticFilterDestination) – Increment the counter if one or more of the given destinations match
- **sources** (iterable of spinnman.model.diagnostic_filter_source.DiagnosticFilterSource) – Increment the counter if one or more of the given sources match (or None or empty list to match all)
- **payload_statuses** (iterable of spinnman.model.diagnostic_filter_payload_status.DiagnosticFilterPayloadStatus) – Increment the counter if one or more of the given payload statuses match (or None or empty list to match all)
- **default_routing_statuses** (iterable of spinnman.model.diagnostic_filter_default_routing_status.DiagnosticFilterDefaultRoutingStatus) – Increment the counter if one or more of the given default routing statuses match (or None or empty list to match all)
- **emergency_routing_statuses** (iterable of spinnman.model.diagnostic_filter_emergency_routing_status.DiagnosticFilterEmergencyRoutingStatus) – Increment the counter if one or more of the given emergency routing statuses match (or None or empty list to match all)

- **packet_types** (iterable of `spinnman.model.diagnostic_filter_packet_type.DiagnosticFilterPacketType`) – Increment the counter if one or more of the given packet types match (or None or empty list to match all)

default_routing_statuses

destinations

emergency_routing_statuses

enable_interrupt_on_counter_event

filter_word
A word of data that can be written to the router to set up the filter

match_emergency_routing_status_to_incoming_packet

packet_types

payload_statuses

static **read_from_int** (*int_value*)

sources

class `spinnman.model.`**ExecutableTargets**
Bases: `object`

Encapsulate the binaries and cores on which to execute them

add_processor (*binary*, *chip_x*, *chip_y*, *chip_p*)
Add a processor to the executable targets

Parameters

- **binary** – the binary path for executable
- **chip_x** – the coordinate on the machine in terms of x for the chip
- **chip_y** – the coordinate on the machine in terms of y for the chip
- **chip_p** – the processor id to place this executable on

Returns

add_subsets (*binary*, *subsets*)
Add core subsets to a binary

Parameters

- **binary** – the path to the binary needed to be executed
- **subsets** – the subset of cores that the binary needs to be loaded on

Returns

all_core_subsets
All the core subsets for all the binaries

binaries
The binaries of the executables

get_cores_for_binary (*binary*)
Get the cores that a binary is to run on

Parameters **binary** – The binary to find the cores for

total_processors
The total number of cores to be loaded

class spinnman.model.**HeapElement**(*block_address*, *next_address*, *free*)
Bases: object

An element of one of the heaps on SpiNNaker

Parameters

- **block_address** – The address of this element on the heap
- **next_address** – The address of the next element on the heap
- **free** – The "free" element of the block as read from the heap

app_id
The app id of the block if allocated, or None if not

block_address
The address of the block

is_free
True if this block is a free block, False otherwise

next_address
The address of the next block, or 0 if none

size
The usable size of this block (not including the header)

tag
The tag of the block if allocated, or None if not

class spinnman.model.**IOBuffer**(*x*, *y*, *p*, *iobuf*)
Bases: object

The contents of IOBUF for a core

Parameters

- **x** (*int*) – The x-coordinate of a chip
- **y** (*int*) – The y-coordinate of a chip
- **p** (*int*) – The p-coordinate of a chip
- **iobuf** (*str*) – The contents of the buffer for the chip

Raises **None** – No known exceptions are raised

iobuf
The contents of the buffer

Returns The contents of the buffer

Return type str

p
The id of the core on the chip

Returns The id of the core

Return type int

x
The x-coordinate of the chip containing the core

Returns The x-coordinate of the chip

Return type int

y

The y-coordinate of the chip containing the core

Returns The y-coordinate of the chip

Return type int

class spinnman.model.**MachineDimensions**(*width*, *height*)

Bases: object

Represents the size of a machine in chips

Parameters

- **width** (*int*) – The width of the machine in chips
- **height** (*int*) – The height of the machine in chips

Raises **None** – No known exceptions are raised

height

The height of the machine in chips

Returns The height

Return type int

width

The width of the machine in chips

Returns The width

Return type int

class spinnman.model.**P2PTable**(*width*, *height*, *column_data*)

Bases: object

Represents a P2P table read from the machine

static **get_column_offset**(*column*)

Get the offset of the next column in the table from the P2P base address

Parameters **column** – The column to be read

static **get_n_column_bytes**(*height*)

Get the number of bytes to be read for each column of the table

Parameters **height** – The height of the machine

get_route(*x*, *y*)

Get the route to follow from this chip to the given chip

Parameters

- **x** – The x-coordinate of the chip to find the route to
- **y** – The y-coordinate of the chip to find the route to

height

The height of the machine that this table represents

is_route(*x*, *y*)

Determines if there is a route in the P2P table to the given chip

Parameters

- **x** – The x-coordinate of the chip to look up
- **y** – The y-coordinate of the chip to look up

iterchips()
Get an iterator of tuples of (x, y) coordinates in the table

width
The width of the machine that this table represents

class spinnman.model.**RouterDiagnostics**(*control_register*, *error_status*, *register_values*)
Bases: object

Represents a set of diagnostic information available from a chip router

Parameters

- **control_register** (*int*) – The value of the control register
- **error_status** (*int*) – The value of the error_status
- **register_values** (*iterable of int*) – The values of the 16 router registers

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the number of register values is not 16

error_status
The error status

Returns The error status

Return type int

mon
The "mon" part of the control register

Returns The mon bits

Return type int

n_dropped_fixed_route_packets
The number of fixed-route packets received that were dropped

Returns The number of packets

Return type int

n_dropped_multicast_packets
The number of multicast packets received that were dropped

Returns The number of packets

Return type int

n_dropped_nearest_neighbour_packets
The number of nearest-neighbour packets received that were dropped

Returns The number of packets

Return type int

n_dropped_peer_to_peer_packets
The number of peer-to-peer packets received that were dropped

Returns The number of packets

Return type int

n_external_fixed_route_packets
The number of fixed-route packets received from external links

Returns The number of packets

Return type int

n_external_multicast_packets
The number of multicast packets received from external links

Returns The number of packets

Return type int

n_external_nearest_neighbour_packets
The number of nearest-neighbour packets received from external links

Returns The number of packets

Return type int

n_external_peer_to_peer_packets
The number of peer-to-peer packets received from external links

Returns The number of packets

Return type int

n_local_fixed_route_packets
The number of fixed-route packets received from local cores

Returns The number of packets

Return type int

n_local_multicast_packets
The number of multicast packets received from local cores

Returns The number of packets

Return type int

n_local_nearest_neighbour_packets
The number of nearest-neighbour packets received from local cores

Returns The number of packets

Return type int

n_local_peer_to_peer_packets
The number of peer-to-peer packets received from local cores

Returns The number of packets

Return type int

registers
The values in all of the registers. Can be used to directly access the registers if they have been programmed to give different values

Returns An array of 16 values

Return type array of int

user_0
the data gained from the user 0 router diag filter

Returns the number of packets captured by this filter.

user_1
the data gained from the user 1 router diag filter

Returns the number of packets captured by this filter.

user_2
the data gained from the user 2 router diag filter

Returns the number of packets captured by this filter.

user_3
the data gained from the user 3 router diag filter

Returns the number of packets captured by this filter.

user_registers
The values in the user control registers

Returns An array of 4 values

Return type array of int

wait_1
The wait_1 part of the control register

Returns The wait_1 bits

Return type int

wait_2
The wait_2 part of the control register

Returns The wait_2 bits

Return type int

class `spinnman.model.`**`VersionInfo`**(*version_data*, *offset=0*)
Bases: `object`

Decodes SC&MP/SARK version information as returned by the SVER command

Parameters

- **version_data** (*bytearray*) – bytes from an SCP packet containing version information
- **offset** – the offset in the bytes from an SCP packet containing version information

Raises *spinnman.exceptions.SpinnmanInvalidParameterException* – If the message does not contain valid version information

build_date
The build date of the software

Returns The number of seconds since 1st January 1970

Return type long

hardware
The hardware being run on

Returns The hardware

Return type str

name
The name of the software

Returns The name

Return type str

p
The processor id of the processor where the information was obtained

Returns the processor id

Return type int

version_number
The version number of the software

Returns The version

Return type float

version_string
The version information as text

Returns The version information

Return type str

x
The x-coordinate of the chip where the information was obtained

Returns the x-coordinate

Return type int

y
The y-coordinate of the chip where the information was obtained

Returns The y-coordinate

Return type int

spinnman.processes package

Submodules

spinnman.processes.abstract_multi_connection_process module

class spinnman.processes.abstract_multi_connection_process.**AbstractMultiConnectionProcess**(*next n_re time out= n_c in- ter- me- di- ate_*

Bases: *spinnman.processes.abstract_process.AbstractProcess*

A process that uses multiple connections in communication

spinnman.processes.abstract_multi_connection_process_connection_selector module

class spinnman.processes.abstract_multi_connection_process_connection_selector.**AbstractMultiC**

Bases: object

A connection selector for multi-connection processes

Parameters **connections** – The connections to be used

get_next_connection(*message*)

Get the index of the next connection for the process from a list of connections

Parameters **message** – The SCP message to be sent

Return type int

spinnman.processes.abstract_process module

class spinnman.processes.abstract_process.**AbstractProcess**

Bases: object

An abstract process for talking to SpiNNaker efficiently

check_for_error(*print_exception=False*)

is_error()

spinnman.processes.abstract_single_connection_process module

class spinnman.processes.abstract_single_connection_process.**AbstractSingleConnectionProcess**(*c*

Bases: *spinnman.processes.abstract_process.AbstractProcess*

A process that uses a single connection in communication

spinnman.processes.application_run_process module

class spinnman.processes.application_run_process.**ApplicationRunProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

run(*app_id*, *core_subsets*, *wait*)

spinnman.processes.de_alloc_sdram_process module

class spinnman.processes.de_alloc_sdram_process.**DeAllocSDRAMProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

de_alloc_sdram(*x*, *y*, *app_id*, *base_address=None*)

handle_sdram_alloc_response(*response*)

no_blocks_freed

spinnman.processes.fill_process module

class spinnman.processes.fill_process.**FillDataType**(*value*, *struct_type*, *doc=''*)

Bases: enum.Enum

BYTE = 1

HALF_WORD = 2

WORD = 4

struct_type

class spinnman.processes.fill_process.**FillProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for filling memory

fill_memory(*x*, *y*, *base_address*, *data*, *size*, *data_type*)

spinnman.processes.get_cpu_info_process module

class spinnman.processes.get_cpu_info_process.**GetCPUInfoProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_cpu_info(*core_subsets*)

handle_read_vcpu_offset(*x*, *y*, *response*)

handle_response(*x*, *y*, *p*, *response*)

spinnman.processes.get_heap_process module

class spinnman.processes.get_heap_process.**GetHeapProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_heap(*x*, *y*, *pointer=<SystemVariableDefinition.sdram_heap_address: _Definition(offset=76, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The base address of the user SDRAM heap')>*)

spinnman.processes.get_machine_process module

class spinnman.processes.get_machine_process.**GetMachineProcess**(*connection_selector*, *ignore_chips*, *ignore_cores*, *ignore_links*, *max_core_id*, *max_sdram_size=None*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for getting the machine details over a set of connections

get_chip_info()
Get the chip information for the machine. Note that get_machine_details must have been called first

get_machine_details(*boot_x*, *boot_y*, *width*, *height*)

spinnman.processes.get_routes_process module

class spinnman.processes.get_routes_process.**GetMultiCastRoutesProcess**(*connection_selector*, *app_id=None*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_routes(*x*, *y*, *base_address*)

handle_read_response(*offset*, *response*)

spinnman.processes.get_tags_process module

class spinnman.processes.get_tags_process.**GetTagsProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_tags(*connection*)

handle_get_tag_response(*tag*, *board_address*, *response*)

handle_tag_info_response(*response*)

spinnman.processes.get_version_process module

class spinnman.processes.get_version_process.**GetVersionProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_single_connection_process.AbstractSingleConnectionProcess*

A process for getting the version of the machine

get_version(*x*, *y*, *p*)

spinnman.processes.load_fixed_route_routing_entry_process module

class spinnman.processes.load_fixed_route_routing_entry_process.**LoadFixedRouteRoutingEntryPro**

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

creates the process for writing a fixed route entry to a chips router

Parameters **connection_selector** – the scamp connection selector

load_fixed_route(*x*, *y*, *fixed_route*, *app_id*)
loads a fixed route routing entry onto a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions

- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **fixed_route** – the fixed route entry
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Return type None

spinnman.processes.load_routes_process module

class spinnman.processes.load_routes_process.**LoadMultiCastRoutesProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

handle_router_alloc_response(*response*)

load_routes(*x*, *y*, *routes*, *app_id*)

spinnman.processes.malloc_sdram_process module

class spinnman.processes.malloc_sdram_process.**MallocSDRAMProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

base_address

handle_sdram_alloc_response(*response*)

malloc_sdram(*x*, *y*, *size*, *app_id*, *tag*)

spinnman.processes.most_direct_connection_selector module

class spinnman.processes.most_direct_connection_selector.**MostDirectConnectionSelector**(*machine*, *connections*)

Bases: *spinnman.processes.abstract_multi_connection_process_connection_selector.AbstractMultiConnectionProcessConnectionSelector*

MostDirectConnectionSelector: selector that goes for the most direct connection for the message

get_next_connection(*message*)

set_machine(*new_machine*)

spinnman.processes.read_fixed_route_routing_entry_process module

class spinnman.processes.read_fixed_route_routing_entry_process.**ReadFixedRouteRoutingEntryPro**

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

supports reading of a chips fixed route routing entry

creates the process for writing a fixed route entry to a chips router

Parameters **connection_selector** – the scamp connection selector

handle_read_response(*response*)

read_fixed_route(*x*, *y*, *app_id*)
reads a fixed route routing table entry

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Return type None

spinnman.processes.read_iobuf_process module

class spinnman.processes.read_iobuf_process.**ReadIOBufProcess**(*connection_selector*)
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for reading memory

handle_extra_iobuf_response(*x*, *y*, *p*, *n*, *offset*, *response*)

handle_first_iobuf_response(*x*, *y*, *p*, *n*, *base_address*, *first_read_size*, *response*)

handle_iobuf_address_response(*iobuf_size*, *x*, *y*, *p*, *response*)

read_iobuf(*iobuf_size*, *core_subsets*)

spinnman.processes.read_memory_process module

class spinnman.processes.read_memory_process.**ReadMemoryProcess**(*connection_selector*)
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for reading memory

handle_response(*offset*, *response*)

read_link_memory(*x*, *y*, *p*, *link*, *base_address*, *length*)

read_memory(*x*, *y*, *p*, *base_address*, *length*)

spinnman.processes.read_router_diagnostics_process module

class spinnman.processes.read_router_diagnostics_process.**ReadRouterDiagnosticsProcess**(*connectio*
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_router_diagnostics(*x*, *y*)

handle_control_register_response(*response*)

handle_error_status_response(*response*)

handle_register_response(*response*)

spinnman.processes.round_robin_connection_selector module

class spinnman.processes.round_robin_connection_selector.**RoundRobinConnectionSelector**(*connectio*

Bases: *spinnman.processes.abstract_multi_connection_process_connection_selector.AbstractMultiConnectionProcessConnectionSelector*

RoundRobinConnectionSelector: selector

get_next_connection(*message*)

spinnman.processes.send_single_command_process module

class spinnman.processes.send_single_command_process.**SendSingleCommandProcess**(*connection_selector*, *n_retries=3*, *time-out=0.5*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

execute(*request*)

handle_response(*response*)

spinnman.processes.write_memory_flood_process module

class spinnman.processes.write_memory_flood_process.**WriteMemoryFloodProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for writing memory

write_memory_from_bytearray(*nearest_neighbour_id*, *base_address*, *data*, *offset*, *n_bytes*)

write_memory_from_reader(*nearest_neighbour_id*, *base_address*, *data*, *n_bytes*)

spinnman.processes.write_memory_process module

class spinnman.processes.write_memory_process.**WriteMemoryProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for writing memory

write_link_memory_from_bytearray(*x*, *y*, *p*, *link*, *base_address*, *data*, *offset*, *n_bytes*)

write_link_memory_from_reader(*x*, *y*, *p*, *link*, *base_address*, *data*, *n_bytes*)

write_memory_from_bytearray(*x*, *y*, *p*, *base_address*, *data*, *offset*, *n_bytes*)

writes memory onto a spinnaker chip from a bytearray

Parameters

- **x** – the x coord of the chip in question

- **y** – the y coord of the chip in question
- **p** – the p coord of the chip in question
- **base_address** – the address in sdram to start writing
- **data** – the data to write
- **offset** – where in the data to start writing from
- **n_bytes** – how much data to write

Return type None

write_memory_from_reader(*x*, *y*, *p*, *base_address*, *data*, *n_bytes*)

Module contents

class spinnman.processes.**AbstractMultiConnectionProcess**(*next_connection_selector*, *n_retries=3*, *timeout=0.5*, *n_channels=8*, *intermediate_channel_waits=7*)

Bases: *spinnman.processes.abstract_process.AbstractProcess*

A process that uses multiple connections in communication

class spinnman.processes.**ApplicationRunProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

run(*app_id*, *core_subsets*, *wait*)

class spinnman.processes.**DeAllocSDRAMProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

de_alloc_sdram(*x*, *y*, *app_id*, *base_address=None*)

handle_sdram_alloc_response(*response*)

no_blocks_freed

class spinnman.processes.**GetCPUInfoProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_cpu_info(*core_subsets*)

handle_read_vcpu_offset(*x*, *y*, *response*)

handle_response(*x*, *y*, *p*, *response*)

class spinnman.processes.**GetHeapProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_heap(*x*, *y*, *pointer=<SystemVariableDefinition.sdram_heap_address: _Definition(offset=76, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The base address of the user SDRAM heap')>*)

class spinnman.processes.**GetMachineProcess**(*connection_selector*, *ignore_chips*, *ignore_cores*, *ignore_links*, *max_core_id*, *max_sdram_size=None*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

A process for getting the machine details over a set of connections

get_chip_info()

Get the chip information for the machine. Note that get_machine_details must have been called first

get_machine_details(*boot_x*, *boot_y*, *width*, *height*)

class spinnman.processes.**GetMultiCastRoutesProcess**(*connection_selector*, *app_id=None*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

get_routes(*x*, *y*, *base_address*)

handle_read_response(*offset*, *response*)

class spinnman.processes.**GetTagsProcess**(*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

get_tags(*connection*)

handle_get_tag_response(*tag*, *board_address*, *response*)

handle_tag_info_response(*response*)

class spinnman.processes.**GetVersionProcess**(*connection_selector*)

Bases: spinnman.processes.abstract_single_connection_process.AbstractSingleConnectionProcess

A process for getting the version of the machine

get_version(*x*, *y*, *p*)

class spinnman.processes.**FillProcess**(*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

A process for filling memory

fill_memory(*x*, *y*, *base_address*, *data*, *size*, *data_type*)

class spinnman.processes.**FillDataType**(*value*, *struct_type*, *doc=''*)

Bases: enum.Enum

BYTE = 1

HALF_WORD = 2

WORD = 4

struct_type

class spinnman.processes.**LoadMultiCastRoutesProcess**(*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

handle_router_alloc_response(*response*)

load_routes(*x*, *y*, *routes*, *app_id*)

class spinnman.processes.**MallocSDRAMProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

base_address

handle_sdram_alloc_response(*response*)

malloc_sdram(*x*, *y*, *size*, *app_id*, *tag*)

class spinnman.processes.**MostDirectConnectionSelector**(*machine*, *connections*)

Bases: *spinnman.processes.abstract_multi_connection_process_connection_selector.AbstractMultiConnectionProcessConnectionSelector*

MostDirectConnectionSelector: selector that goes for the most direct connection for the message

get_next_connection(*message*)

set_machine(*new_machine*)

class spinnman.processes.**ReadIOBufProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for reading memory

handle_extra_iobuf_response(*x*, *y*, *p*, *n*, *offset*, *response*)

handle_first_iobuf_response(*x*, *y*, *p*, *n*, *base_address*, *first_read_size*, *response*)

handle_iobuf_address_response(*iobuf_size*, *x*, *y*, *p*, *response*)

read_iobuf(*iobuf_size*, *core_subsets*)

class spinnman.processes.**ReadMemoryProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

A process for reading memory

handle_response(*offset*, *response*)

read_link_memory(*x*, *y*, *p*, *link*, *base_address*, *length*)

read_memory(*x*, *y*, *p*, *base_address*, *length*)

class spinnman.processes.**ReadRouterDiagnosticsProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_router_diagnostics(*x*, *y*)

handle_control_register_response(*response*)

handle_error_status_response(*response*)

handle_register_response(*response*)

class spinnman.processes.**RoundRobinConnectionSelector**(*connections*)

Bases: *spinnman.processes.abstract_multi_connection_process_connection_selector.AbstractMultiConnectionProcessConnectionSelector*

RoundRobinConnectionSelector: selector

get_next_connection(*message*)

class spinnman.processes.**SendSingleCommandProcess**(*connection_selector*, *n_retries=3*, *time-out=0.5*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

execute(*request*)

handle_response(*response*)

class spinnman.processes.**WriteMemoryFloodProcess**(*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

A process for writing memory

write_memory_from_bytearray(*nearest_neighbour_id*, *base_address*, *data*, *offset*, *n_bytes*)

write_memory_from_reader(*nearest_neighbour_id*, *base_address*, *data*, *n_bytes*)

class spinnman.processes.**WriteMemoryProcess**(*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

A process for writing memory

write_link_memory_from_bytearray(*x*, *y*, *p*, *link*, *base_address*, *data*, *offset*, *n_bytes*)

write_link_memory_from_reader(*x*, *y*, *p*, *link*, *base_address*, *data*, *n_bytes*)

write_memory_from_bytearray(*x*, *y*, *p*, *base_address*, *data*, *offset*, *n_bytes*)

writes memory onto a spinnaker chip from a bytearray

Parameters

- **x** – the x coord of the chip in question
- **y** – the y coord of the chip in question
- **p** – the p coord of the chip in question
- **base_address** – the address in sdram to start writing
- **data** – the data to write
- **offset** – where in the data to start writing from
- **n_bytes** – how much data to write

Return type None

write_memory_from_reader(*x*, *y*, *p*, *base_address*, *data*, *n_bytes*)

class spinnman.processes.**ReadFixedRouteRoutingEntryProcess**(*connection_selector*)

Bases: spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess

supports reading of a chips fixed route routing entry

creates the process for writing a fixed route entry to a chips router

Parameters **connection_selector** – the scamp connection selector

handle_read_response(*response*)

read_fixed_route(*x*, *y*, *app_id*)

reads a fixed route routing table entry

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Return type None

class spinnman.processes.**LoadFixedRouteRoutingEntryProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

creates the process for writing a fixed route entry to a chips router

Parameters **connection_selector** – the scamp connection selector

load_fixed_route(*x*, *y*, *fixed_route*, *app_id*)

loads a fixed route routing entry onto a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip, between 0 and 255, this is not checked due to speed restrictions
- **y** (*int*) – The y-coordinate of the chip, between 0 and 255 this is not checked due to speed restrictions
- **fixed_route** – the fixed route entry
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Return type None

spinnman.utilities package

Subpackages

spinnman.utilities.io package

Submodules

spinnman.utilities.io.abstract_io module

class spinnman.utilities.io.abstract_io.**AbstractIO**

Bases: object

address

Return the current absolute address within the region

close()

Close the IO object

closed

Indicates if the object has been closed

fill(*repeat_value*, *bytes_to_fill=None*, *data_type=<FillDataType.WORD: 4>*)
Fill the next part of the region with repeated data

Parameters

- **repeat_value** (*int*) – The value to repeat
- **bytes_to_fill** (*int*) – Optional number of bytes to fill from current position, or None to fill to the end
- **data_type** (spinnman.process.fill_process.FillProcess) – The type of the repeat value

Raises **EOFError** – If the amount of data to fill is more than the region

flush()
Flush any outstanding written data

read(*n_bytes=None*)
Read a number of bytes, or the rest of the data if n_bytes is None or negative

Parameters **n_bytes** – The number of bytes to read

Return type bytes

Raises **EOFError** – If the read will be beyond the end of the region

seek(*n_bytes*, *from_what=0*)
Seek to a position within the region

tell()
Return the current position within the region relative to the start

write(*data*)
Write some data to the region

Parameters **data** (*bytes*) – The data to write

Returns The number of bytes written

Return type int

Raises **EOFError** – If the write will go over the end of the region

spinnman.utilities.io.file_io module

class spinnman.utilities.io.file_io.**FileIO**(*file_obj*, *start_offset*, *end_offset*)
Bases: *spinnman.utilities.io.abstract_io.AbstractIO*

A file input/output interface to match the MemoryIO interface

Parameters

- **file_obj** (*str or file*) – The file handle or file name to write to
- **start_offset** (*int*) – The start offset into the file
- **end_offset** (*int or None*) – The end offset from the start of the file

address
Return the current absolute address within the region

close()
Close the IO object

closed

fill(*repeat_value*, *bytes_to_fill=None*, *data_type=<FillDataType.WORD: 4>*)

Fill the next part of the region with repeated data

Parameters

- **repeat_value** (*int*) – The value to repeat
- **bytes_to_fill** (*int*) – Optional number of bytes to fill from current position, or None to fill to the end
- **data_type** (spinnman.process.fill_process.FillProcess) – The type of the repeat value

Raises **EOFError** – If the amount of data to fill is more than the region

flush()

Flush any outstanding written data

read(*n_bytes=None*)

Read a number of bytes, or the rest of the data if n_bytes is None or negative

Parameters **n_bytes** – The number of bytes to read

Return type bytes

Raises **EOFError** – If the read will be beyond the end of the region

seek(*n_bytes*, *from_what=0*)

Seek to a position within the region

tell()

Return the current position within the region relative to the start

write(*data*)

Write some data to the region

Parameters **data** (*bytes*) – The data to write

Returns The number of bytes written

Return type int

Raises **EOFError** – If the write will go over the end of the region

spinnman.utilities.io.memory_io module

class spinnman.utilities.io.memory_io.**MemoryIO**(*transceiver*, *x*, *y*, *start_address*, *end_address*)

Bases: *spinnman.utilities.io.abstract_io.AbstractIO*

A file-like object for reading and writing memory

Parameters

- **transceiver** – The transceiver to read and write with
- **x** – The x-coordinate of the chip to write to
- **y** – The y-coordinate of the chip to write to
- **start_address** – The start address of the region to write to
- **end_address** – The end address of the region to write to. This is the first address just outside the region

address
Return the current absolute address within the region

close()
Close the IO object

closed

fill(*repeat_value*, *bytes_to_fill=None*, *data_type=<FillDataType.WORD: 4>*)
Fill the next part of the region with repeated data

Parameters

- **repeat_value** (*int*) – The value to repeat
- **bytes_to_fill** (*int*) – Optional number of bytes to fill from current position, or None to fill to the end
- **data_type** (spinnman.process.fill_process.FillProcess) – The type of the repeat value

Raises **EOFError** – If the amount of data to fill is more than the region

flush()
Flush any outstanding written data

read(*n_bytes=None*)
Read a number of bytes, or the rest of the data if n_bytes is None or negative

Parameters **n_bytes** – The number of bytes to read

Return type bytes

Raises **EOFError** – If the read will be beyond the end of the region

seek(*n_bytes*, *from_what=0*)
Seek to a position within the region

tell()
Return the current position within the region relative to the start

write(*data*)
Write some data to the region

Parameters **data** (*bytes*) – The data to write

Returns The number of bytes written

Return type int

Raises **EOFError** – If the write will go over the end of the region

Module contents

class spinnman.utilities.io.**AbstractIO**
Bases: object

address
Return the current absolute address within the region

close()
Close the IO object

closed
Indicates if the object has been closed

fill(*repeat_value*, *bytes_to_fill=None*, *data_type=<FillDataType.WORD: 4>*)
Fill the next part of the region with repeated data

Parameters

- **repeat_value** (*int*) – The value to repeat
- **bytes_to_fill** (*int*) – Optional number of bytes to fill from current position, or None to fill to the end
- **data_type** (spinnman.process.fill_process.FillProcess) – The type of the repeat value

Raises **EOFError** – If the amount of data to fill is more than the region

flush()
Flush any outstanding written data

read(*n_bytes=None*)
Read a number of bytes, or the rest of the data if n_bytes is None or negative

Parameters **n_bytes** – The number of bytes to read

Return type bytes

Raises **EOFError** – If the read will be beyond the end of the region

seek(*n_bytes*, *from_what=0*)
Seek to a position within the region

tell()
Return the current position within the region relative to the start

write(*data*)
Write some data to the region

Parameters **data** (*bytes*) – The data to write

Returns The number of bytes written

Return type int

Raises **EOFError** – If the write will go over the end of the region

class spinnman.utilities.io.**FileIO**(*file_obj*, *start_offset*, *end_offset*)
Bases: *spinnman.utilities.io.abstract_io.AbstractIO*

A file input/output interface to match the MemoryIO interface

Parameters

- **file_obj** (*str or file*) – The file handle or file name to write to
- **start_offset** (*int*) – The start offset into the file
- **end_offset** (*int or None*) – The end offset from the start of the file

address
Return the current absolute address within the region

close()
Close the IO object

closed

fill(*repeat_value*, *bytes_to_fill=None*, *data_type=<FillDataType.WORD: 4>*)
Fill the next part of the region with repeated data

Parameters

- **repeat_value** (*int*) – The value to repeat
- **bytes_to_fill** (*int*) – Optional number of bytes to fill from current position, or None to fill to the end
- **data_type** (spinnman.process.fill_process.FillProcess) – The type of the repeat value

Raises **EOFError** – If the amount of data to fill is more than the region

flush()
Flush any outstanding written data

read(*n_bytes=None*)
Read a number of bytes, or the rest of the data if n_bytes is None or negative

Parameters **n_bytes** – The number of bytes to read

Return type bytes

Raises **EOFError** – If the read will be beyond the end of the region

seek(*n_bytes*, *from_what=0*)
Seek to a position within the region

tell()
Return the current position within the region relative to the start

write(*data*)
Write some data to the region

Parameters **data** (*bytes*) – The data to write

Returns The number of bytes written

Return type int

Raises **EOFError** – If the write will go over the end of the region

class spinnman.utilities.io.**MemoryIO**(*transceiver*, *x*, *y*, *start_address*, *end_address*)
Bases: *spinnman.utilities.io.abstract_io.AbstractIO*

A file-like object for reading and writing memory

Parameters

- **transceiver** – The transceiver to read and write with
- **x** – The x-coordinate of the chip to write to
- **y** – The y-coordinate of the chip to write to
- **start_address** – The start address of the region to write to
- **end_address** – The end address of the region to write to. This is the first address just outside the region

address
Return the current absolute address within the region

close()
Close the IO object

closed

fill(*repeat_value*, *bytes_to_fill=None*, *data_type=<FillDataType.WORD: 4>*)

Fill the next part of the region with repeated data

Parameters

- **repeat_value** (*int*) – The value to repeat
- **bytes_to_fill** (*int*) – Optional number of bytes to fill from current position, or None to fill to the end
- **data_type** (spinnman.process.fill_process.FillProcess) – The type of the repeat value

Raises **EOFError** – If the amount of data to fill is more than the region

flush()

Flush any outstanding written data

read(*n_bytes=None*)

Read a number of bytes, or the rest of the data if n_bytes is None or negative

Parameters **n_bytes** – The number of bytes to read

Return type bytes

Raises **EOFError** – If the read will be beyond the end of the region

seek(*n_bytes*, *from_what=0*)

Seek to a position within the region

tell()

Return the current position within the region relative to the start

write(*data*)

Write some data to the region

Parameters **data** (*bytes*) – The data to write

Returns The number of bytes written

Return type int

Raises **EOFError** – If the write will go over the end of the region

Submodules

spinnman.utilities.appid_tracker module

class spinnman.utilities.appid_tracker.**AppIdTracker**(*app_ids_in_use=None*, *min_app_id=17*, *max_app_id=254*)

Bases: object

A tracker of AppId to make it easier to allocate new ids

Parameters

- **app_ids_in_use** (*list of int or None*) – The ids that are already in use
- **min_app_id** (*int*) – The smallest app id to use
- **max_app_id** (*int*) – The largest app id to use

allocate_id(*allocated_id*)
Allocate a given id - raises KeyError if the id is not present

Parameters **allocated_id** – The id to allocate

free_id(*id_to_free*)
Free a given id - raises KeyError if the id is out of range

Parameters **id_to_free** – The id to free

get_new_id()
Get a new unallocated id

Return type int

spinnman.utilities.locate_connected_machine_ip_address module

locates any spinnaker machines ip address from the auto transmitted packet from non-booted spinnaker machines

spinnman.utilities.reports module

spinnman.utilities.reports.**generate_machine_report**(*report_directory*, *machine*, *connections*)
Generate report on the physical structure of the target SpiNNaker machine.

Parameters

- **report_directory** (*str*) – the directory to which reports are stored
- **machine** (*spinnmachine.machine.Machine object*) – the machine python object
- **connections** (*iterable of implementations of spinnman.connections.abstract_classes.connection.AbstractConnection*) – the list of connections to the machine

Return type None

Raises **IOError** – when a file cannot be opened for some reason

spinnman.utilities.utility_functions module

spinnman.utilities.utility_functions.**get_vcpu_address**(*p*)
Get the address of the vcpu_t structure for the given core

Parameters **p** (*int*) – The core

spinnman.utilities.utility_functions.**send_port_trigger_message**(*connection*, *board_address*)
Sends a port trigger message using a connection to (hopefully) open a port in a NAT and/or firewall to allow incoming packets to be received.

Parameters

- **connection** – The UDP connection down which the trigger message should be sent
- **board_address** – The address of the SpiNNaker board to which the message should be sent

spinnman.utilities.utility_functions.**work_out_bmp_from_machine_details**(*hostname*, *number_of_boards*)

Work out the BMP connection ip address given the machine details. This is assumed to be the IP address of the machine, with 1 subtracted from the final part e.g. if the machine IP address is 192.168.0.1, the BMP IP address is assumed to be 192.168.0.0

Parameters

- **hostname** – the spinnaker machine main hostname or IP address
- **number_of_boards** – the number of boards in the machine

Returns The BMP connection data

Module contents

Submodules

spinnman.constants module

class spinnman.constants.**EIEIO_COMMAND_IDS**

Bases: enum.Enum

DATABASE_CONFIRMATION = 1

EVENT_PADDING = 2

EVENT_STOP = 3

HOST_DATA_READ = 9

HOST_DATA_READ_ACK = 12

HOST_SEND_SEQUENCED_DATA = 7

SPINNAKER_REQUEST_BUFFERS = 6

SPINNAKER_REQUEST_READ_DATA = 8

START_RESUME_NOTIFICATION = 11

START_SENDING_REQUESTS = 5

STOP_PAUSE_NOTIFICATION = 10

STOP_SENDING_REQUESTS = 4

class spinnman.constants.**IPTAG_TIME_OUT_WAIT_TIMES**

Bases: enum.Enum

TIMEOUT_10_ms = 1

TIMEOUT_1280_ms = 8

TIMEOUT_160_ms = 5

TIMEOUT_20_ms = 2

TIMEOUT_2560_ms = 9

TIMEOUT_320_ms = 6

TIMEOUT_40_ms = 3

TIMEOUT_640_ms = 7

TIMEOUT_80_ms = 4

spinnman.constants.**READ_TYPES**
alias of Read_types

spinnman.constants.**ROUTER_REGISTER_REGISTERS**
alias of Registers

spinnman.exceptions module

exception spinnman.exceptions.**SpinnmanEIEIOPacketParsingException**(*parsing_format*, *packet*)
Bases: *spinnman.exceptions.SpinnmanException*

Unable to complete the parsing of the EIEIO packet received. The routine used is invalid or the content of the packet is invalid

packet

exception spinnman.exceptions.**SpinnmanException**
Bases: exceptions.Exception

Superclass of exceptions that occur when dealing with communication with SpiNNaker

exception spinnman.exceptions.**SpinnmanGenericProcessException**(*exception*, *tb*, *x*, *y*, *p*)
Bases: *spinnman.exceptions.SpinnmanException*

Encapsulates exceptions from processes which communicate with some core/chip

exception

exception spinnman.exceptions.**SpinnmanIOException**(*problem*)
Bases: *spinnman.exceptions.SpinnmanException*

An exception that something went wrong with the underlying IO

Parameters **problem** (*str*) – The problem with the IO

problem
The problem with IO

exception spinnman.exceptions.**SpinnmanInvalidPacketException**(*packet_type*, *problem*)
Bases: *spinnman.exceptions.SpinnmanException*

An exception that indicates that a packet was not in the expected format

Parameters

- **packet_type** (*str*) – The type of packet expected
- **problem** (*str*) – The problem with the packet

packet_type
The packet type

problem
The problem with the packet

exception spinnman.exceptions.**SpinnmanInvalidParameterException**(*parameter*, *value*, *problem*)
Bases: *spinnman.exceptions.SpinnmanException*

An exception that indicates that the value of one of the parameters passed was invalid

Parameters

- **parameter** (*str*) – The name of the parameter that is invalid
- **value** (*str*) – The value of the parameter that is invalid
- **problem** (*str*) – The problem with the parameter

parameter
The parameter with an invalid value

problem
The problem with the parameter value

value
The value that is invalid

exception spinnman.exceptions.**SpinnmanInvalidParameterTypeException**(*parameter*, *param_type*, *problem*)

Bases: *spinnman.exceptions.SpinnmanException*

An exception that indicates that the type of one of the parameters passed was invalid

Parameters

- **parameter** (*str*) – The name of the parameter that is invalid
- **param_type** (*str*) – The type of the parameter that is invalid
- **problem** (*str*) – The problem with the parameter

parameter
The parameter with an invalid value

problem
The problem with the parameter value

type
The value that is invalid

exception spinnman.exceptions.**SpinnmanTimeoutException**(*operation*, *timeout*)

Bases: *spinnman.exceptions.SpinnmanException*

An exception that indicates that a timeout occurred before an operation could finish

Parameters

- **operation** (*str*) – The operation being performed
- **timeout** (*int*) – The timeout value in seconds

operation
The operation that was performed

timeout
The timeout value in seconds

exception spinnman.exceptions.**SpinnmanUnexpectedResponseCodeException**(*operation*, *command*, *response*)

Bases: *spinnman.exceptions.SpinnmanException*

Indicate that a response code returned from the board was unexpected for the current operation

Parameters

- **operation** (*str*) – The operation being performed
- **command** (*str*) – The command being executed
- **response** (*str*) – The response received in error

command
The command being executed

operation
The operation being performed

response
The unexpected response

exception spinnman.exceptions.**SpinnmanUnsupportedOperationException**(*operation*)
Bases: *spinnman.exceptions.SpinnmanException*

An exception that indicates that the given operation is not supported

Parameters **operation** (*str*) – The operation being requested

operation
The unsupported operation requested

spinnman.transceiver module

class spinnman.transceiver.**Transceiver**(*version*, *connections=None*, *ignore_chips=None*, *ignore_cores=None*, *ignore_links=None*, *max_core_id=None*, *scamp_connections=None*, *max_sdram_size=None*)
Bases: object

An encapsulation of various communications with the spinnaker board.

The methods of this class are designed to be thread-safe; thus you can make multiple calls to the same (or different) methods from multiple threads and expect each call to work as if it had been called sequentially, although the order of returns is not guaranteed. Note also that with multiple connections to the board, using multiple threads in this way may result in an increase in the overall speed of operation, since the multiple calls may be made separately over the set of given connections.

Parameters

- **version** (*int*) – The version of the board being connected to
- **connections** (iterable of *spinnman.connections.abstract_classes.connection.Connection*) – An iterable of connections to the board. If not specified, no communication will be possible until connections are found.
- **ignore_chips** (*set of (int, int of chips to ignore*) – An optional set of chips to ignore in the machine. Requests for a "machine" will have these chips excluded, as if they never existed. The processor_ids of the specified chips are ignored.
- **ignore_cores** (*set of (int, int, int) of cores to ignore*) – An optional set of cores to ignore in the machine. Requests for a "machine" will have these cores excluded, as if they never existed.
- **ignore_links** (*set of (int, int, int) of links to ignore*) – An optional set of links to ignore in the machine. Requests for a "machine" will have these links excluded, as if they never existed.

- **max_core_id** (*int*) – The maximum core id in any discovered machine. Requests for a "machine" will only have core ids up to and including this value.
- **max_sdram_size** (*int or None*) – the max size each chip can say it has for SDRAM (mainly used in debugging purposes)
- **scamp_connections** (list of spinnman.connections.socket_address_with_chip.SocketAddress_With_Chip or None) – a list of scamp connection data or None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board, or if no connections to the board can be found (if connections is None)
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

app_id_tracker

Get the app id tracker for this transceiver

Return type *spinnman.utilities.appid_tracker.AppIdTracker*

bmp_connection

boot_board(*number_of_boards=None*, *width=None*, *height=None*, *extra_boot_values=None*)

Attempt to boot the board. No check is performed to see if the board is already booted.

Parameters

- **number_of_boards** (*int*) – this parameter is deprecated
- **width** (*int or None*) – this parameter is deprecated
- **height** (*int or None*) – this parameter is deprecated
- **extra_boot_values** (*dict of SystemVariableDefinition to value*) – extra values to set during boot

Raises

- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If the board version is not known
- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board

clear_ip_tag(*tag*, *connection=None*, *board_address=None*)

Clear the setting of an ip tag

Parameters

- **tag** (*int*) – The tag id
- **connection** (*spinnman.connections.abstract_classes.SCPSender*) – Connection where the tag should be cleared. If not specified, all SCPSender connections will send the message to clear the tag

- **board_address** – Board address where the tag should be cleared. If not specified, all SCPSender connections will send the message to clear the tag

Returns Nothing is returned

Return type None

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If the tag is not a valid tag
  - If the connection cannot send SDP messages
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

clear_multicast_routes(*x*, *y*)

Remove all the multicast routes on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip on which to clear the routes
- **y** (*int*) – The y-coordinate of the chip on which to clear the routes

Returns Nothing is returned

Return type None

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* – If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

clear_router_diagnostic_counters(*x, y, enable=True, counter_ids=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]*)

Clear router diagnostic information on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip
- **enable** (*bool*) – True (default) if the counters should be enabled
- **counter_ids** (*array-like of int*) – The ids of the counters to reset (all by default) and enable if enable is True; each must be between 0 and 15

Return type None

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* – If a packet is received that has invalid parameters or a counter id is out of range
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

close(*close_original_connections=True*, *power_off_machine=False*)

Close the transceiver and any threads that are running

Parameters

- **close_original_connections** – If True, the original connections passed to the transceiver in the constructor are also closed. If False, only newly discovered connections are closed.
- **power_off_machine** (*bool*) – if true, the machine is sent a power down command via its BMP (if it has one)

Returns Nothing is returned

Return type None

Raises **None** – No known exceptions are raised

discover_scamp_connections()

Find connections to the board and store these for future use. Note that connections can be empty, in which case another local discovery mechanism will be used. Note that an exception will be thrown if no initial connections can be found to the board.

Returns An iterable of discovered connections, not including the initially given connections in the constructor

Return type iterable of *spinnman.connections.abstract_classes.connection.Connection*

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* – If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

ensure_board_is_ready(*number_of_boards=None*, *width=None*, *height=None*, *n_retries=5*, *extra_boot_values=None*)

Ensure that the board is ready to interact with this version of the transceiver. Boots the board if not already booted and verifies that the version of SCAMP running is compatible with this transceiver.

Parameters

- **number_of_boards** (*int*) – this parameter is deprecated and will be ignored
- **width** (*int or None*) – this parameter is deprecated and will be ignored
- **height** (*int or None*) – this parameter is deprecated and will be ignored
- **n_retries** (*int*) – The number of times to retry booting
- **extra_boot_values** (*dict of SystemVariableDefinition to value*) – Any additional values to set during boot

Returns The version identifier

Return type *spinnman.model.version_info.VersionInfo*

Raise spinnman.exceptions.SpinnmanIOException: * If there is a problem booting the board * If the version of software on the board is not compatible with this transceiver

execute(*x*, *y*, *processors*, *executable*, *app_id*, *n_bytes=None*, *wait=False*, *is_filename=False*)

Start an executable running on a single core

Parameters

- **x** (*int*) – The x-coordinate of the chip on which to run the executable
- **y** (*int*) – The y-coordinate of the chip on which to run the executable
- **processors** (*iterable of int*) – The cores on the chip on which to run the application
- **executable** (spinnman.data.abstract_data_reader.AbstractDataReader or bytearray or str) – The data that is to be executed. Should be one of the following: * An instance of AbstractDataReader * A bytearray * A filename of a file containing the executable (in which case is_filename must be set to True)
- **app_id** (*int*) – The id of the application with which to associate the executable
- **n_bytes** (*int*) – The size of the executable data in bytes. If not specified: * If executable is an AbstractDataReader, an error is raised * If executable is a bytearray, the length of the bytearray will be used * If executable is an int, 4 will be used * If executable is a str, the length of the file will be used
- **wait** (*bool*) – True if the binary should enter a "wait" state on loading
- **is_filename** (*bool*) – True if executable is a filename

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** –
  - If there is an error communicating with the board
  - If there is an error reading the executable
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If x, y, p does not lead to a valid core
  - If app_id is an invalid application id

- If a packet is received that has invalid parameters

- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

execute_application(*executable_targets*, *app_id*)

Execute a set of binaries that make up a complete application on specified cores, wait for them to be ready and then start all of the binaries. Note this will get the binaries into c_main but will not signal the barrier.

Parameters

- **executable_targets** (*spinnman.model.executable_targets.ExecutableTargets*) – The binaries to be executed and the cores to execute them on
- **app_id** (*int*) – The app_id to give this application

execute_flood(*core_subsets*, *executable*, *app_id*, *n_bytes=None*, *wait=False*, *is_filename=False*)

Start an executable running on multiple places on the board. This will be optimised based on the selected cores, but it may still require a number of communications with the board to execute.

Parameters

- **core_subsets** (*spinn_machine.CoreSubsets*) – Which cores on which chips to start the executable
- **executable** (spinnman.data.abstract_data_reader.AbstractDataReader or bytearray or str) – The data that is to be executed. Should be one of the following: * An instance of AbstractDataReader * A bytearray * A filename of an executable (in which case is_filename must be set to True)
- **app_id** (*int*) – The id of the application with which to associate the executable
- **n_bytes** (*int*) – The size of the executable data in bytes. If not specified: * If executable is an AbstractDataReader, an error is raised * If executable is a bytearray, the length of the bytearray will be used * If executable is an int, 4 will be used * If executable is a str, the length of the file will be used
- **wait** (*bool*) – True if the processors should enter a "wait" state on loading
- **is_filename** (*bool*) – True if the data is a filename

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** –
  - If there is an error communicating with the board
  - If there is an error reading the executable
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If one of the specified cores is not valid
  - If app_id is an invalid application id
  - If a packet is received that has invalid parameters
  - If data is an AbstractDataReader but n_bytes is not specified

- If data is an int and n_bytes is more than 4
- If n_bytes is less than 0

- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

fill_memory(*x*, *y*, *base_address*, *repeat_value*, *bytes_to_fill*, *data_type=<FillDataType.WORD: 4>*)

Fill some memory with repeated data

Parameters

- **x** (*int*) – The x-coordinate of the chip
- **y** (*int*) – The y-coordinate of the chip
- **base_address** (*int*) – The address at which to start the fill
- **repeat_value** (*int*) – The data to repeat
- **bytes_to_fill** (*int*) – The number of bytes to fill - must be compatible with the data type i.e. if the data type is WORD, the number of bytes must be divisible by 4
- **data_type** (*spinnman.processes.fill_process.FillDataType*) –

free_sdram(*x*, *y*, *base_address*, *app_id*)

Free allocated SDRAM

Parameters

- **x** (*int*) – The x-coordinate of the chip onto which to ask for memory
- **y** (*int*) – The y-coordinate of the chip onto which to ask for memory
- **base_address** (*int*) – The base address of the allocated memory
- **app_id** (*int*) – The app id of the allocated memory

free_sdram_by_app_id(*x*, *y*, *app_id*)

Free all SDRAM allocated to a given app id

Parameters

- **x** (*int*) – The x-coordinate of the chip onto which to ask for memory
- **y** (*int*) – The y-coordinate of the chip onto which to ask for memory
- **app_id** (*int*) – The app id of the allocated memory

Returns The number of blocks freed

Return type int

get_connections()

Get the currently known connections to the board, made up of those passed in to the transceiver and those that are discovered during calls to discover_connections. No further discovery is done here.

Returns An iterable of connections known to the transceiver

Return type iterable of *spinnman.connections.abstract_classes.connection.Connection*

Raises **None** – No known exceptions are raised

get_core_state_count(*app_id*, *state*)

Get a count of the number of cores which have a given state

Parameters

- **app_id** (*int*) – The id of the application from which to get the count.
- **state** (spinnman.model.CPUState) – The state count to get

Returns A count of the cores with the given status

Return type int

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If state is not a valid status
  - If app_id is not a valid application id
  - If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

get_core_status_string(*cpu_infos*)

Get a string indicating the status of the given cores

Parameters **cpu_infos** (*spinnman.model.cpu_infos.CPUInfos*) – A CPUInfos objects

get_cores_in_state(*all_core_subsets*, *states*)

Get all cores that are in a given state or set of states

Parameters

- **all_core_subsets** (*spinn_machine.CoreSubsets*) – The cores to filter
- **states** (*spinnman.model.enums.cpu_state.CPUState* or set of *spinnman.model.enums.cpu_state.CPUState*) – The state or states to filter on

Returns Core subsets object containing cores in the

get_cores_not_in_state(*all_core_subsets*, *states*)

Get all cores that are not in a given state or set of states

Parameters

- **all_core_subsets** –
- **states** –

Returns

get_cpu_information(*core_subsets=None*)

Get information about the processors on the board

Parameters **core_subsets** (*spinn_machine.CoreSubsets*) – A set of chips and cores from which to get the information. If not specified, the information from all of the cores on all of the chips on the board are obtained

Returns An iterable of the cpu information for the selected cores, or all cores if core_subsets is not specified

Return type iterable of `spinnman.model.CPUInfo`

Raises

- **`spinnman.exceptions.SpinnmanIOException`** – If there is an error communicating with the board
- **`spinnman.exceptions.SpinnmanInvalidPacketException`** – If a packet is received that is not in the valid format
- **`spinnman.exceptions.SpinnmanInvalidParameterException`** –
  - If chip_and_cores contains invalid items
  - If a packet is received that has invalid parameters
- **`spinnman.exceptions.SpinnmanUnexpectedResponseCodeException`** – If a response indicates an error during the exchange

`get_cpu_information_from_core`(*x*, *y*, *p*)

Get information about a specific processor on the board

Parameters

- **`x`** (`int`) – The x-coordinate of the chip containing the processor
- **`y`** (`int`) – The y-coordinate of the chip containing the processor
- **`p`** (`int`) – The id of the processor to get the information about

Returns The cpu information for the selected core

Return type `spinnman.model.cpu_info.CPUInfo`

Raises

- **`spinnman.exceptions.SpinnmanIOException`** – If there is an error communicating with the board
- **`spinnman.exceptions.SpinnmanInvalidPacketException`** – If a packet is received that is not in the valid format
- **`spinnman.exceptions.SpinnmanInvalidParameterException`** –
  - If x, y, p is not a valid processor
  - If a packet is received that has invalid parameters
- **`spinnman.exceptions.SpinnmanUnexpectedResponseCodeException`** – If a response indicates an error during the exchange

`get_heap`(*x*, *y*, *heap=<SystemVariableDefinition.sdram_heap_address: _Definition(offset=76, data_type=<_DataType.INT: 4>, default=0, array_size=None, doc='The base address of the user SDRAM heap')>*)

Get the contents of the given heap on a given chip

Parameters

- **`x`** (`int`) – The x-coordinate of the chip
- **`y`** (`int`) – The y-coordinate of the chip
- **`heap`** (`SystemVariableDefinition`) – The SystemVariableDefinition which is the heap to read

`get_iobuf`(*core_subsets=None*)

Get the contents of the IOBUF buffer for a number of processors

Parameters **core_subsets** (*spinn_machine.CoreSubsets*) – A set of chips and cores from which to get the buffers. If not specified, the buffers from all of the cores on all of the chips on the board are obtained

Returns An iterable of the buffers, which may not be in the order of core_subsets

Return type iterable of *spinnman.model.IOBuffer*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If chip_and_cores contains invalid items
  - If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

get_iobuf_from_core(*x*, *y*, *p*)

Get the contents of IOBUF for a given core

Parameters

- **x** (*int*) – The x-coordinate of the chip containing the processor
- **y** (*int*) – The y-coordinate of the chip containing the processor
- **p** (*int*) – The id of the processor to get the IOBUF for

Returns An IOBUF buffer

Return type *spinnman.model.IOBuffer*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If chip_and_cores contains invalid items
  - If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

get_machine_details()

Get the details of the machine made up of chips on a board and how they are connected to each other.

Returns A machine description

Return type *spinn_machine.Machine*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board

- **spinnman.exceptions.SpinnmanInvalidPacketException** – If a packet is received that is not in the valid format
- **spinnman.exceptions.SpinnmanInvalidParameterException** – If a packet is received that has invalid parameters
- **spinnman.exceptions.SpinnmanUnexpectedResponseCodeException** – If a response indicates an error during the exchange

get_machine_dimensions()

Get the maximum chip x-coordinate and maximum chip y-coordinate of the chips in the machine

Returns The dimensions of the machine

Return type *spinnman.model.machine_dimensions.MachineDimensions*

Raises

- **spinnman.exceptions.SpinnmanIOException** – If there is an error communicating with the board
- **spinnman.exceptions.SpinnmanInvalidPacketException** – If a packet is received that is not in the valid format
- **spinnman.exceptions.SpinnmanInvalidParameterException** – If a packet is received that has invalid parameters
- **spinnman.exceptions.SpinnmanUnexpectedResponseCodeException** – If a response indicates an error during the exchange

get_multicast_routes(*x*, *y*, *app_id=None*)

Get the current multicast routes set up on a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip from which to get the routes
- **y** (*int*) – The y-coordinate of the chip from which to get the routes
- **app_id** (*int*) – The id of the application to filter the routes for. If not specified, will return all routes

Returns An iterable of multicast routes

Return type iterable of spinnman.model.multicast_routing_entry.MulticastRoute

Raises

- **spinnman.exceptions.SpinnmanIOException** – If there is an error communicating with the board
- **spinnman.exceptions.SpinnmanInvalidPacketException** – If a packet is received that is not in the valid format
- **spinnman.exceptions.SpinnmanInvalidParameterException** – If a packet is received that has invalid parameters
- **spinnman.exceptions.SpinnmanUnexpectedResponseCodeException** – If a response indicates an error during the exchange

get_router_diagnostic_filter(*x*, *y*, *position*)

Gets a router diagnostic filter from a router

Parameters

- **x** (*int*) – the x address of the router from which this filter is being retrieved
- **y** (*int*) – the y address of the router from which this filter is being retrieved
- **position** (*int*) – the position in the list of filters where this filter is to be added

Returns The diagnostic filter read

Return type *spinnman.model.diagnostic_filter.DiagnosticFilter*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** –
  - If there is an error communicating with the board
  - If there is an error reading the data
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If x, y does not lead to a valid chip
  - If a packet is received that has invalid parameters
  - If position is less than 0 or more than 15
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

get_router_diagnostics(*x*, *y*)

Get router diagnostic information from a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip from which to get the information
- **y** (*int*) – The y-coordinate of the chip from which to get the information

Returns The router diagnostic information

Return type *spinnman.model.router_diagnostics.RouterDiagnostics*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

get_scamp_version(*chip_x=255*, *chip_y=255*, *connection_selector=None*)

Get the version of scamp which is running on the board

Parameters

- **connection_selector** (*a instance of a :py:class:'spinnman.processes.abstract_multi_connection_process_connection_selector.AbstractMultiConnectionProcessConnectionSelector'*) – the connection to send the scamp version or none (if none then a random scamp connection is used)
- **chip_x** (*int*) – the chip's x coordinate to query for scamp version
- **chip_y** – the chip's y coordinate to query for scamp version

Returns The version identifier

Return type *spinnman.model.version_info.VersionInfo*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If the timeout is less than 1
- ***spinnman.exceptions.SpinnmanTimeoutException*** – If none of the retries resulted in a response before the timeout (suggesting that the board is not booted)

get_tags(*connection=None*)

Get the current set of tags that have been set on the board

Parameters **connection** (*spinnman.connections.abstract_classes.SCPSender*) – Connection from which the tags should be received. If not specified, all SCPSender connections will be queried and the response will be combined.

Returns An iterable of tags

Return type iterable of *spinn_machine.tags.AbstractTag*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If the connection cannot send SDP messages
  - If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

get_user_0_register_address_from_core(*x, y, p*)

Get the address of user 0 for a given processor on the board

Parameters

- **x** (*int*) – the x-coordinate of the chip containing the processor
- **y** (*int*) – the y-coordinate of the chip containing the processor
- **p** (*int*) – The id of the processor to get the user 0 address from

Returns The address for user 0 register for this processor

Return type int

Raises

- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If x, y, p is not a valid processor
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

get_user_1_register_address_from_core(*x*, *y*, *p*)

Get the address of user 1 for a given processor on the board

Parameters

- **x** (*int*) – the x-coordinate of the chip containing the processor
- **y** (*int*) – the y-coordinate of the chip containing the processor
- **p** (*int*) – The id of the processor to get the user 1 address from

Returns The address for user 1 register for this processor

Return type int

Raises

- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If x, y, p is not a valid processor
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

get_user_2_register_address_from_core(*x*, *y*, *p*)

Get the address of user 2 for a given processor on the board

Parameters

- **x** (*int*) – the x-coordinate of the chip containing the processor
- **y** (*int*) – the y-coordinate of the chip containing the processor
- **p** (*int*) – The id of the processor to get the user 0 address from

Returns The address for user 0 register for this processor

Return type int

Raises

- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If x, y, p is not a valid processor
  - If a packet is received that has invalid parameters

- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

is_connected(*connection=None*)

Determines if the board can be contacted

Parameters **connection** (*spinnman.connections.abstract_classes.connection.Connection*) – The connection which is to be tested. If none, all connections will be tested, and the board will be considered to be connected if any one connection works.

Returns True if the board can be contacted, False otherwise

Return type bool

Raises **None** – No known exceptions are raised

static **is_scamp_version_compabible**(*version*)

Determine if the version of SCAMP is compatible with this transceiver

Parameters **version** (*(int, int, int)*) – The version to test

load_fixed_route(*x*, *y*, *fixed_route*, *app_id*)

loads a fixed route routing table entry to a chips router

Parameters

- **x** (*int*) – The x-coordinate of the chip onto which to load the routes
- **y** (*int*) – The y-coordinate of the chip onto which to load the routes
- **fixed_route** (spinn_machine.fixed_route_routing_entry) – the route for the fixed route entry on this chip
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Returns Nothing is returned

Return type None

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If any of the routes are invalid
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

load_multicast_routes(*x*, *y*, *routes*, *app_id*)

Load a set of multicast routes on to a chip

Parameters

- **x** (*int*) – The x-coordinate of the chip onto which to load the routes
- **y** (*int*) – The y-coordinate of the chip onto which to load the routes

- **routes** (iterable of *spinn_machine.MulticastRoutingEntry*) – An iterable of multicast routes to load
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If any of the routes are invalid
  - If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

locate_spinnaker_connection_for_board_address(*board_address*)

Find a connection that matches the given board IP address

Parameters **board_address** (*str*) – The IP address of the Ethernet connection on the board

Returns A connection for the given IP address, or None if no such connection exists

Return type `spinnman.connections.udp_packet_connections.udp_scamp_connection.SCAMPConnection`

malloc_sdram(*x*, *y*, *size*, *app_id*, *tag=None*)

Allocates a chunk of SDRAM on a chip on the machine

Parameters

- **x** (*int*) – The x-coordinate of the chip onto which to ask for memory
- **y** (*int*) – The y-coordinate of the chip onto which to ask for memory
- **size** (*int*) – the amount of memory to allocate in bytes
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.
- **tag** (*int*) – the tag for the SDRAM, a 8-bit (chip-wide) tag that can be looked up by a SpiNNaker application to discover the address of the allocated block. If *0* then no tag is applied.

Returns the base address of the allocated memory

Return type int

number_of_boards_located

Get the number of boards currently configured

power_off(*boards=0*, *cabinet=0*, *frame=0*)

Power off a set of boards in the machine

Parameters

- **boards** – The board or boards to power off
- **cabinet** – the id of the cabinet containing the frame, or 0 if the frame is not in a cabinet
- **frame** – the id of the frame in the cabinet containing the board(s), or 0 if the board is not in a frame

power_off_machine()
Power off the whole machine

power_on(*boards=0*, *cabinet=0*, *frame=0*)
Power on a set of boards in the machine

Parameters

- **boards** – The board or boards to power on
- **cabinet** – the id of the cabinet containing the frame, or 0 if the frame is not in a cabinet
- **frame** – the id of the frame in the cabinet containing the board(s), or 0 if the board is not in a frame

power_on_machine()
Power on the whole machine

read_adc_data(*board*, *cabinet*, *frame*)
Read the BMP ADC data

Parameters

- **cabinet** (*int*) – cabinet: the cabinet this is targeting
- **frame** (*int*) – the frame this is targeting
- **board** – which board to request the ADC data from

Returns the FPGA's ADC data object

read_bmp_version(*board*, *cabinet*, *frame*)
Read the BMP version

Parameters

- **cabinet** (*int*) – cabinet: the cabinet this is targeting
- **frame** (*int*) – the frame this is targeting
- **board** – which board to request the data from

Returns the sver from the BMP

read_fixed_route(*x*, *y*, *app_id*)
reads a fixed route routing table entry

Parameters

- **x** (*int*) – The x-coordinate of the chip onto which to load the routes
- **y** (*int*) – The y-coordinate of the chip onto which to load the routes
- **app_id** (*int*) – The id of the application with which to associate the routes. If not specified, defaults to 0.

Returns the route as a fixed route entry

read_fpga_register(*fpga_num*, *register*, *cabinet*, *frame*, *board*)

Parameters

- **fpga_num** (*int*) – FPGA number (0, 1 or 2) to communicate with.
- **register** (*int*) – Register address to read to (will be rounded down to the nearest 32-bit word boundary).
- **cabinet** (*int*) – cabinet: the cabinet this is targeting
- **frame** (*int*) – the frame this is targeting
- **board** – which board to request the FPGA register from

Returns the register data

read_memory (*x*, *y*, *base_address*, *length*, *cpu=0*)

Read some areas of SDRAM from the board

Parameters

- **x** (*int*) – The x-coordinate of the chip where the memory is to be read from
- **y** (*int*) – The y-coordinate of the chip where the memory is to be read from
- **base_address** (*int*) – The address in SDRAM where the region of memory to be read starts
- **length** (*int*) – The length of the data to be read in bytes
- **cpu** (*int*) – the core id used to read the memory of

Returns A bytearray of data read

Return type bytearray

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If one of x, y, p, base_address or length is invalid
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

read_neighbour_memory (*x*, *y*, *link*, *base_address*, *length*, *cpu=0*)

Read some areas of memory on a neighbouring chip using a LINK_READ SCP command. If sent to a BMP, this command can be used to communicate with the FPGAs' debug registers.

Parameters

- **x** (*int*) – The x-coordinate of the chip whose neighbour is to be read from
- **y** (*int*) – The y-coordinate of the chip whose neighbour is to be read from
- **cpu** (*int*) – The cpu to use, typically 0 (or if a BMP, the slot number)
- **link** (*int*) – The link index to send the request to (or if BMP, the FPGA number)
- **base_address** (*int*) – The address in SDRAM where the region of memory to be read starts

- **length** (*int*) – The length of the data to be read in bytes

Returns An iterable of chunks of data read in order

Return type iterable of bytearray

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If one of x, y, p, base_address or length is invalid
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

register_udp_listener(*callback*, *connection_class*, *local_port=None*, *local_host=None*)

Register a callback for a certain type of traffic to be received via UDP. Note that the connection class must extend *sp*
to avoid clashing with the SCAMP and BMP functionality

Parameters

- **callback** (*function(packet)*) – Function to be called when a packet is received
- **connection_class** – The class of connection to receive using
- **local_port** (*int*) – The optional port number to listen on; if not specified, an existing connection will be used if possible, otherwise a random free port number will be used
- **local_host** (*str*) – The optional hostname or IP address to listen on; if not specified, all interfaces will be used for listening

Returns The connection to be used

Return type `spinnman.connection.udp_packet_connections.UDPConnection`

scamp_connection_selector

send_multicast_message(*x*, *y*, *multicast_message*, *connection=None*)

Sends a multicast message to the board (currently unsupported)

Parameters

- **x** (*int*) – The x-coordinate of the chip where the message should first arrive on the board
- **y** (*int*) – The y-coordinate of the chip where the message should first arrive on the board
- **multicast_message** (*spinnman.messages.multicast_message.MulticastMessage*) – A multicast message to send
- **connection** (*spinnman.connections.abstract_classes.multicast_sender.MulticastSender*) – A specific connection over which to send the message. If not specified, an appropriate connection is chosen automatically

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanUnsupportedOperationException*** –
  - If there is no connection that supports sending over multicast (or the given connection does not)
  - If there is no connection that can make the packet arrive at the selected chip (ignoring routing tables)

send_scp_message(*message*, *connection=None*)

Sends an SCP message, without expecting a response

Parameters

- **message** (spinnman.messages.scp.abstract_scp_request.AbstractSCPRequest) – The message to send
- **connection** (*spinnman.connections.abstract_classes.connection.Connection*) – The connection to use

Returns The received response, or the callback if get_callback is True

Return type spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse

Raises

- ***spinnman.exceptions.SpinnmanTimeoutException*** – If there is a timeout before a message is received
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If one of the fields of the received message is invalid
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** –
  - If the message is not a recognised packet type
  - If a packet is received that is not a valid response
- ***spinnman.exceptions.SpinnmanUnsupportedOperationException*** – If no connection can send the type of message given
- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error sending the message or receiving the response
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If the response is not one of the expected codes

send_sdp_message(*message*, *connection=None*)

Sends an SDP message using one of the connections.

Parameters

- **message** (spinnman.messages.sdp.SDPMessage) – The message to send
- **connection** (*spinnman.connections.abstract_classes.connection.Connection*) – An optional connection to use

Return type None

send_signal(*app_id*, *signal*)

Send a signal to an application

Parameters

- **app_id** (*int*) – The id of the application to send to
- **signal** (spinnman.messages.scp.Signal) – The signal to send

Returns Nothing is returned

Return type None

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If signal is not a valid signal
  - If app_id is not a valid application id
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

set_ip_tag(*ip_tag*)

Set up an ip tag

Parameters **ip_tag** (*spinn_machine.tags.IPTag*) – The tag to set up; note board_address can be None, in which case, the tag will be assigned to all boards

Returns Nothing is returned

Return type None

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If the ip tag fields are incorrect
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

set_led(*led*, *action*, *board*, *cabinet*, *frame*)

Set the LED state of a board in the machine

Parameters

- **led** (*int or iterable of int*) – Number of the LED or an iterable of LEDs to set the state of (0-7)
- **action** (spinnman.messages.scp.scp_led_action.SCPLEDAction) – State to set the LED to, either on, off or toggle

- **board** (*int or iterable*) – Specifies the board to control the LEDs of. This may also be an iterable of multiple boards (in the same frame). The command will actually be sent to the first board in the iterable.
- **cabinet** (*int*) – the cabinet this is targeting
- **frame** (*int*) – the frame this is targeting

Return type None

set_leds (*x*, *y*, *cpu*, *led_states*)

Set SetLED states.

Parameters

- **x** (*int*) – The x-coordinate of the chip on which to set the LEDs
- **y** (*int*) – The x-coordinate of the chip on which to set the LEDs
- **cpu** (*int*) – The CPU of the chip on which to set the LEDs
- **led_states** (*dict*) – A dictionary mapping SetLED index to state with 0 being off, 1 on and 2 inverted.

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

set_reverse_ip_tag (*reverse_ip_tag*)

Set up a reverse ip tag

Parameters **reverse_ip_tag** (*spinn_machine.tags.ReverseIPTag*) – The reverse tag to set up; note board_address can be None, in which case, the tag will be assigned to all boards

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If the reverse ip tag fields are incorrect
  - If a packet is received that has invalid parameters

- If the UDP port is one that is already used by spiNNaker for system functions

- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

set_router_diagnostic_filter(*x*, *y*, *position*, *diagnostic_filter*)

Sets a router diagnostic filter in a router

Parameters

- **x** (*int*) – the x address of the router in which this filter is being set
- **y** (*int*) – the y address of the router in which this filter is being set
- **position** (*int*) – the position in the list of filters where this filter is to be added
- **diagnostic_filter** (*spinnman.model.diagnostic_filter.DiagnosticFilter*) – the diagnostic filter being set in the placed, between 0 and 15 (note that positions 0 to 11 are used by the default filters, and setting these positions will result in a warning).

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** –
  - If there is an error communicating with the board
  - If there is an error reading the data
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If x, y does not lead to a valid chip
  - If position is less than 0 or more than 15
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

set_watch_dog(*watch_dog*)

Enable, disable or set the value of the watch dog timer

Parameters **watch_dog** (*boolean or int*) – Either a boolean indicating whether to enable (True) or disable (False) the watch dog timer or an int value to set the timer count to

Return type None

set_watch_dog_on_chip(*x*, *y*, *watch_dog*)

Enable, disable or set the value of the watch dog timer on a specific chip

Parameters

- **x** (*int*) – chip x coord to write new watch dog param to
- **y** (*int*) – chip y coord to write new watch dog param to
- **watch_dog** (*boolean or int*) – Either a boolean indicating whether to enable (True) or disable (False) the watch dog timer or an int value to set the timer count to

Return type None

stop_application(*app_id*)

Sends a stop request for an app_id

Parameters **app_id** (*int*) – The id of the application to send to

Raises

- *spinnman.exceptions.SpinnmanIOException* – If there is an error communicating with the board
- *spinnman.exceptions.SpinnmanInvalidPacketException* – If a packet is received that is not in the valid format
- *spinnman.exceptions.SpinnmanInvalidParameterException* –
  - If app_id is not a valid application id
  - If a packet is received that has invalid parameters
- *spinnman.exceptions.SpinnmanUnexpectedResponseCodeException* – If a response indicates an error during the exchange

wait_for_cores_to_be_in_state (*all_core_subsets*, *app_id*, *cpu_states*, *timeout=None*, *time_between_polls=0.1*, *error_states=set([<CPUState.WATCHDOG: 3>, <CPUState.RUN_TIME_EXCEPTION: 2>])*, *counts_between_full_check=100*)

Parameters

- **all_core_subsets** – the cores to check are in a given sync state
- **app_id** – the app id that being used by the simulation
- **cpu_states** – The expected states once the applications are ready; success is when each application is in one of these states
- **timeout** – The amount of time to wait in seconds for the cores to reach one of the states
- **time_between_polls** – Time between checking the state
- **error_states** – Set of states that the application can be in that indicate an error, and so should raise an exception
- **counts_between_full_check** – The number of times to use the count signal before instead using the full CPU state check

write_fpga_register (*fpga_num*, *register*, *value*, *cabinet*, *frame*, *board*)

Parameters

- **fpga_num** (*int*) – FPGA number (0, 1 or 2) to communicate with.
- **register** (*int*) – Register address to read to (will be rounded down to the nearest 32-bit word boundary).
- **value** (*int*) – the value to write into the FPGA register
- **cabinet** (*int*) – cabinet: the cabinet this is targeting
- **frame** (*int*) – the frame this is targeting
- **board** – which board to write the FPGA register to

Return type None

write_memory (*x*, *y*, *base_address*, *data*, *n_bytes=None*, *offset=0*, *cpu=0*, *is_filename=False*)

Write to the SDRAM on the board

Parameters

- **x** (*int*) – The x-coordinate of the chip where the memory is to be written to
- **y** (*int*) – The y-coordinate of the chip where the memory is to be written to
- **base_address** (*int*) – The address in SDRAM where the region of memory is to be written
- **data** (spinnman.data.abstract_data_reader.AbstractDataReader or bytearray or int or str) – The data to write. Should be one of the following: * An instance of AbstractDataReader * A bytearray * A single integer - will be written using little-endian byte ordering * A filename of a data file (in which case is_filename must be set to True)
- **n_bytes** (*int*) – The amount of data to be written in bytes. If not specified: * If data is an AbstractDataReader, an error is raised * If data is a bytearray, the length of the bytearray will be used * If data is an int, 4 will be used * If data is a str, the length of the file will be used
- **offset** (*int*) – The offset from which the valid data begins
- **cpu** (*int*) – The optional cpu to write to
- **is_filename** (*bool*) – True if the data is a filename

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** –
  - If there is an error communicating with the board
  - If there is an error reading the data
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If x, y does not lead to a valid chip
  - If a packet is received that has invalid parameters
  - If base_address is not a positive integer
  - If data is an AbstractDataReader but n_bytes is not specified
  - If data is an int and n_bytes is more than 4
  - If n_bytes is less than 0
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

write_memory_flood(*base_address*, *data*, *n_bytes=None*, *offset=0*, *is_filename=False*)

Write to the SDRAM of all chips.

Parameters

- **base_address** (*int*) – The address in SDRAM where the region of memory is to be written
- **data** (spinnman.data.abstract_data_reader.AbstractDataReader or bytearray or int) – The data that is to be written. Should be one of the following: * An instance of AbstractDataReader * A bytearray or bytestring * A single integer * A file name of a file to read (in which case is_filename should be set to True)

- **n_bytes** (*int*) – The amount of data to be written in bytes. If not specified: * If data is an AbstractDataReader, an error is raised * If data is a bytearray, the length of the bytearray will be used * If data is an int, 4 will be used * If data is a str, the size of the file will be used
- **offset** (*int*) – The offset where the valid data starts, if the data is a int, then the offset will be ignored and 0 is used.
- **is_filename** (*bool*) – True if the data should be interpreted as a file name

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** –
  - If there is an error communicating with the board
  - If there is an error reading the executable
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If one of the specified chips is not valid
  - If app_id is an invalid application id
  - If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

write_neighbour_memory(*x*, *y*, *link*, *base_address*, *data*, *n_bytes=None*, *offset=0*, *cpu=0*)

Write to the memory of a neighbouring chip using a LINK_READ SCP command. If sent to a BMP, this command can be used to communicate with the FPGAs' debug registers.

Parameters

- **x** (*int*) – The x-coordinate of the chip whose neighbour is to be written to
- **y** (*int*) – The y-coordinate of the chip whose neighbour is to be written to
- **link** (*int*) – The link index to send the request to (or if BMP, the FPGA number)
- **base_address** (*int*) – The address in SDRAM where the region of memory is to be written
- **data** (`spinnman.data.abstract_data_reader.AbstractDataReader` or bytearray or int) – The data to write. Should be one of the following: * An instance of AbstractDataReader * A bytearray * A single integer - will be written using little-endian byte ordering
- **n_bytes** (*int*) – The amount of data to be written in bytes. If not specified: * If data is an AbstractDataReader, an error is raised * If data is a bytearray, the length of the bytearray will be used * If data is an int, 4 will be used
- **offset** (*int*) – The offset where the valid data starts (if the data is an int then offset will be ignored and used 0
- **cpu** (*int*) – The cpu to use, typically 0 (or if a BMP, the slot number)

Returns Nothing is returned

Return type None

Raises

- ***spinnman.exceptions.SpinnmanIOException*** –
  - If there is an error communicating with the board
  - If there is an error reading the data
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** –
  - If x, y does not lead to a valid chip
  - If a packet is received that has invalid parameters
  - If base_address is not a positive integer
  - If data is an AbstractDataReader but n_bytes is not specified
  - If data is an int and n_bytes is more than 4
  - If n_bytes is less than 0
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

spinnman.transceiver.**create_transceiver_from_hostname**(*hostname*, *version*, *bmp_connection_data=None*, *number_of_boards=None*, *ignore_chips=None*, *ignore_cores=None*, *ignored_links=None*, *max_core_id=None*, *auto_detect_bmp=False*, *scamp_connections=None*, *boot_port_no=None*, *max_sdram_size=None*)

Create a Transceiver by creating a UDPConnection to the given hostname on port 17893 (the default SCAMP port), and a optionally discovering any additional links using the UDPConnection, and then returning the transceiver created with the conjunction of the created UDPConnection and the discovered connections

Parameters

- **hostname** (*str*) – The hostname or IP address of the board
- **number_of_boards** (*int or None*) – a number of boards expected to be supported, or None, which defaults to a single board
- **ignore_chips** (*set of (int, int) of chips to ignore*) – An optional set of chips to ignore in the machine. Requests for a "machine" will have these chips excluded, as if they never existed. The processor_ids of the specified chips are ignored.
- **ignore_cores** (*set of (int, int, int) of cores to ignore*) – An optional set of cores to ignore in the machine. Requests for a "machine" will have these cores excluded, as if they never existed.
- **ignored_links** (*set of (int, int, int) of links to ignore*) – An optional set of links to ignore in the machine. Requests for a "machine" will have these links excluded, as if they never existed.

- **max_core_id** (*int*) – The maximum core id in any discovered machine. Requests for a "machine" will only have core ids up to this value.
- **version** – the type of spinnaker board used within the spinnaker machine being used. If a spinn-5 board, then the version will be 5, spinn-3 would equal 3 and so on.
- **bmp_connection_data** (iterable *spinnman.model.bmp_connection_data.BMPConnectionData*) – the details of the BMP connections used to boot multi-board systems
- **auto_detect_bmp** (*bool*) – True if the BMP of version 4 or 5 boards should be automatically determined from the board IP address
- **boot_port_no** (*int*) – the port number used to boot the machine
- **scamp_connections** (*iterable of UDPScampConnections*) – the list of connections used for scamp communications
- **max_sdram_size** (*int or None*) – the max size each chip can say it has for SDRAM (mainly used in debugging purposes)

Returns The created transceiver

Return type *spinnman.transceiver.Transceiver*

Raises

- ***spinnman.exceptions.SpinnmanIOException*** – If there is an error communicating with the board
- ***spinnman.exceptions.SpinnmanInvalidPacketException*** – If a packet is received that is not in the valid format
- ***spinnman.exceptions.SpinnmanInvalidParameterException*** – If a packet is received that has invalid parameters
- ***spinnman.exceptions.SpinnmanUnexpectedResponseCodeException*** – If a response indicates an error during the exchange

Module contents

Used to communicate with a SpiNNaker Board. The main part of this package is the *spinnman.transceiver.Transceiver* class. This can be used to send and receive packets in various SpiNNaker formats, depending on what connections are available.

Functional Requirements

- Connect to and communicate with a machine using a number of different connections.
- Boot a machine with the expected version of the software.
  - If the machine is already booted but the version is not the version expected, an exception will be thrown.
- Check the version of the software which the machine is booted with.
- Query the state of the machine to determine:
  - What the current state of the machine is in terms of the chips and cores available, the SDRAM available on the chips and the which links are available between which chips.

  - What external links to the host exist (and separately add the discovered links to the set of links used to communicate with the machine).
  - What is running on the machine and where, and what the current status of those processes are.
  - How many cores are in a given state.
  - What is in the IOBUF buffers.
  - What the current routing entries for a given router are.
  - What the routing status counter values are.
- Load application binaries on to the machine, either to individual cores or via a "flood-fill" mechanism to multiple cores simultaneously (which may be a subset of the cores on a subset of the chips).
- Write data to SDRAM, either on an individual chip, or via a "flood-fill" mechanism to multiple chips simultaneously.
- Send a signal to an application.
- Read data from SDRAM on an individual chip.
- Send and receive SpiNNaker packets where the connections allow this.
  - If no connection supports this packet type, an exception is thrown.
  - The user should be able to select which connection is used. Selection of a connection which does not support the traffic type will also result in an exception.
- Send and receive SCP and SDP packets where the connections allow this.
  - If no connection supports the packet type, an exception is thrown.
  - The user should be able to select which connection is used. Selection of a connection which does not support the traffic type will also result in an exception.
- It should be possible to call any of the functions simultaneously, including the same function more than once.
  - Where possible, multiple connections should be used to overlap calls.
  - The functions should not return until they have confirmed that any messages sent have been received, and any responses have been received.
  - Functions should not respond with the result of a different function.
  - Functions can further sub-divide the call into a number of separate calls that can be divided across the available connections, so long as the other requirements are met.
- More than one machine can be connected to the same host.
  - Once the subset of connections has been worked out for each machine, the operation of these machines should be independent.

Use Cases

- `boot_board()` and `get_scamp_version()` are used to ensure that the board is booted correctly before starting a simulation.
- `get_machine_details()` is used to get a representation of the current state of the machine, which is used to decide where executables are to be run on the board for a particular simulation, where any external peripherals are connected, and how messages between the executables and/or the external peripherals are to be routed
- `write_memory()` and `execute()` are used to write parameters and execute executables on the board

- *send_signal()* is used to send a signal which starts, stops or pauses a simulation.
- `get_core_status_count()` is used to determine if a simulation is complete or has gone into an error state.
- *get_iobuf()*, *get_cpu_information()* and *get_router_diagnostics()* are used to diagnose a problem with a simulation
- *read_memory()* is used to read some statistics recorded in SDRAM after a simulation

5.1.4 Indices and tables

- genindex
- modindex
- search

SpiNNMan_github

SpiNNMan_individual_docs

CHAPTER 6

DataSpecification

This package provides utilities for specifying binary data algorithmically, and executing the specifications to produce the data.

6.1 DataSpecification

Used to generate memory images from a set of instructions.

The main part of this package is the `data_specification.data_specification_generator.DataSpecificationGenerator` class. This is used to generate a "Data Specification", which can then be executed to produce a memory image. This package also handles this function if required, through the `data_specification.data_specification_executor.DataSpecificationExecutor` class.

6.1.1 Functional Requirements

- Creation of a Data Specification Language file which can be executed to produce a memory image.
  - Any errors that can be checked during the creation of the specification should throw an exception.
  - It will be impossible to detect all errors at creation time.
  - There should be no assumption of where the data specification is be stored, although a default provision of a way to write the specification to a file is acceptable.
- Execution of a Data Specification Language file, producing a memory image.
  - This should detect any errors during execution and report them, halting the execution.
  - There should be no assumption of where the data specification is read from, although a default provision of a way to read the specification from a file is acceptable.

6.1.2 Use Cases

There are a number of use-cases of this library:

- `DataSpecificationGenerator` is used to create a compressed memory image which can be expanded later, to reduce the amount of data that needs to be transferred over a slow connection.
- `DataSpecificationExecutor` is used to execute a previously generated specification at the receiving end of a slow connection.

6.1.3 Contents

data_specification

data_specification package

Subpackages

data_specification.enums package

Submodules

data_specification.enums.arithemetic_operation module

class `data_specification.enums.arithemetic_operation.`**`ArithmeticOperation`**(*value*, *operator*, *doc=''*)

Bases: `enum.Enum`

Arithmetic Operations

ADD = 0

MULTIPLY = 2

SUBTRACT = 1

data_specification.enums.commands module

class `data_specification.enums.commands.`**`Commands`**(*value*, *exec_function*, *doc=''*)

Bases: `enum.Enum`

Set of opcodes for the spec executor

ALIGN_WR_PTR = 101

ARITH_OP = 103

BLOCK_COPY = 69

BREAK = 0

BREAK_LOOP = 82

CONSTRUCT = 64

COPY_PARAM = 113

COPY_STRUCT = 112

DECLARE_RANDOM_DIST = 6

DECLARE_RNG = 5

ELSE = 86

END_CONSTRUCTOR = 37

END_IF = 87

END_LOOP = 83

END_SPEC = 255

END_STRUCT = 18

FREE = 3

GET_RANDOM_NUMBER = 7

GET_WR_PTR = 99

IF = 85

LOGIC_OP = 104

LOOP = 81

MV = 96

NOP = 1

PRINT_STRUCT = 130

PRINT_TXT = 129

PRINT_VAL = 128

READ = 65

READ_PARAM = 115

REFORMAT = 106

RESERVE = 2

SET_WR_PTR = 100

START_CONSTRUCTOR = 32

START_STRUCT = 16

STRUCT_ELEM = 17

SWITCH_FOCUS = 80

WRITE = 66

WRITE_ARRAY = 67

WRITE_PARAM = 114

WRITE_PARAM_COMPONENT = 116

WRITE_STRUCT = 68

data_specification.enums.condition module

class `data_specification.enums.condition.`**`Condition`**(*value*, *operator*, *doc=''*)

Bases: `enum.Enum`

Comparison Operations

EQUAL = 0

GREATER_THAN = 5

GREATER_THAN_OR_EQUAL = 4

LESS_THAN = 3

LESS_THAN_OR_EQUAL = 2

NOT_EQUAL = 1

data_specification.enums.data_type module

class `data_specification.enums.data_type.`**`DataType`**(*value*, *size*, *min_val*, *max_val*, *scale*, *struct_encoding*, *doc=''*)

Bases: `enum.Enum`

Supported data types The first value is an identifier for the enum class; The second value is the size in bytes of the type; The third value is the minimum possible value for the type; The fourth value is the maximum possible value for the type; The fifth value is the scale of the input value to convert it in integer; The sixth value is the pattern to use following the struct package encodings to convert the data in binary format; The seventh value is the text description of the type.

FLOAT_32 = 14

FLOAT_64 = 15

INT16 = 5

INT32 = 6

INT64 = 7

INT8 = 4

S015 = 21

S031 = 22

S063 = 23

S07 = 20

S1615 = 12

S3231 = 13

S87 = 11

U016 = 17

U032 = 18

U064 = 19

U08 = 16

U1616 = 9

U3232 = 10

U88 = 8

UINT16 = 1

UINT32 = 2

UINT64 = 3

UINT8 = 0

data_specification.enums.logic_operation module

class data_specification.enums.logic_operation.**LogicOperation**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Logic Operations

AND = 3

LEFT_SHIFT = 0

NOT = 5

OR = 2

RIGHT_SHIFT = 1

XOR = 4

data_specification.enums.random_number_generator module

class data_specification.enums.random_number_generator.**RandomNumberGenerator**(*value*, *doc=''*)

Bases: enum.Enum

Random number generator types

MERSENNE_TWISTER = 0

Module contents

class data_specification.enums.**ArithmeticOperation**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Arithmetic Operations

ADD = 0

MULTIPLY = 2

SUBTRACT = 1

class data_specification.enums.**Commands**(*value*, *exec_function*, *doc=''*)

Bases: enum.Enum

Set of opcodes for the spec executor

ALIGN_WR_PTR = 101

ARITH_OP = 103

BLOCK_COPY = 69

BREAK = 0

BREAK_LOOP = 82

CONSTRUCT = 64

COPY_PARAM = 113

COPY_STRUCT = 112

DECLARE_RANDOM_DIST = 6

DECLARE_RNG = 5

ELSE = 86

END_CONSTRUCTOR = 37

END_IF = 87

END_LOOP = 83

END_SPEC = 255

END_STRUCT = 18

FREE = 3

GET_RANDOM_NUMBER = 7

GET_WR_PTR = 99

IF = 85

LOGIC_OP = 104

LOOP = 81

MV = 96

NOP = 1

PRINT_STRUCT = 130

PRINT_TXT = 129

PRINT_VAL = 128

READ = 65

READ_PARAM = 115

REFORMAT = 106

RESERVE = 2

SET_WR_PTR = 100

START_CONSTRUCTOR = 32

START_STRUCT = 16

STRUCT_ELEM = 17

SWITCH_FOCUS = 80

WRITE = 66

WRITE_ARRAY = 67

WRITE_PARAM = 114

WRITE_PARAM_COMPONENT = 116

WRITE_STRUCT = 68

class data_specification.enums.**Condition**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Comparison Operations

EQUAL = 0

GREATER_THAN = 5

GREATER_THAN_OR_EQUAL = 4

LESS_THAN = 3

LESS_THAN_OR_EQUAL = 2

NOT_EQUAL = 1

class data_specification.enums.**DataType**(*value*, *size*, *min_val*, *max_val*, *scale*, *struct_encoding*, *doc=''*)

Bases: enum.Enum

Supported data types The first value is an identifier for the enum class; The second value is the size in bytes of the type; The third value is the minimum possible value for the type; The fourth value is the maximum possible value for the type; The fifth value is the scale of the input value to convert it in integer; The sixth value is the pattern to use following the struct package encodings to convert the data in binary format; The seventh value is the text description of the type.

FLOAT_32 = 14

FLOAT_64 = 15

INT16 = 5

INT32 = 6

INT64 = 7

INT8 = 4

S015 = 21

S031 = 22

S063 = 23

S07 = 20

S1615 = 12

S3231 = 13

S87 = 11

U016 = 17

U032 = 18

U064 = 19

U08 = 16

U1616 = 9

U3232 = 10

U88 = 8

UINT16 = 1

UINT32 = 2

UINT64 = 3

UINT8 = 0

class data_specification.enums.**LogicOperation**(*value*, *operator*, *doc=''*)

Bases: enum.Enum

Logic Operations

AND = 3

LEFT_SHIFT = 0

NOT = 5

OR = 2

RIGHT_SHIFT = 1

XOR = 4

class data_specification.enums.**RandomNumberGenerator**(*value*, *doc=''*)

Bases: enum.Enum

Random number generator types

MERSENNE_TWISTER = 0

Submodules

data_specification.constants module

Constants used by the Data Structure Generator (DSG) and the Spec Executor

data_specification.data_specification_executor module

class data_specification.data_specification_executor.**DataSpecificationExecutor**(*spec_reader*, *memory_space*)

Bases: object

Used to execute a data specification language file to produce a memory image

Parameters

- **spec_reader** (data_specification.abstract_data_reader.AbstractDataReader) – The object to read the specification language file from
- **memory_space** (*int*) – memory available on the destination architecture

Raises

- *spinn_storage_handlers.exceptions.DataReadException* – If a read from external storage fails

- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

dsef

execute()

Executes the specification

Returns The number of bytes used by the image and the number of bytes written by the image

Return type int

Raises

- *spinn_storage_handlers.exceptions.DataReadException* – If a read from external storage fails
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- *data_specification.exceptions.DataSpecificationException* – If there is an error when executing the specification
- **data_specification.exceptions. DataSpecificationTablePointerOutOfMemory** – If the table pointer generated as data header exceeds the size of the available memory

get_constructed_data_size()

Return the size of the data that will be written to memory

Returns size of the data that will be written to memory

Return type unsigned int

get_header()

Get the header of the data as a numpy array

get_pointer_table(*start_address*)

Get the pointer table as a numpy array

Parameters **start_address** – The base address of the data to be written

get_region(*region_id*)

Get a region with a given id

Parameters **region_id** (*int*) – The id of the region to get

Returns The region, or None if the region was not allocated

spec_reader

data_specification.data_specification_executor_functions module

class data_specification.data_specification_executor_functions.**DataSpecificationExecutorFunct**

Bases: object

This class includes the function related to each of the commands of the data specification file.

Parameters

- **spec_reader** (data_specification.abstract_data_reader.AbstractDataReader) – The object to read the specification language file from
- **memory_space** (*int*) – Memory space available for the data to be generated

current_region

data_len

dest_reg

execute_align_wr_ptr(*cmd*)

execute_arith_op(*cmd*)

execute_block_copy(*cmd*)

execute_break(*cmd*)

This command raises an exception to stop the execution of the data spec executor (DSE)

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.ExecuteBreakInstruction* – Raises the exception to break the execution of the DSE

execute_break_loop(*cmd*)

execute_construct(*cmd*)

execute_copy_param(*cmd*)

execute_copy_struct(*cmd*)

execute_declare_rng(*cmd*)

execute_else(*cmd*)

execute_end_constructor(*cmd*)

execute_end_if(*cmd*)

execute_end_loop(*cmd*)

execute_end_spec(*cmd*)

Return the value which terminates the data spec executor

Parameters **cmd** (*int*) – the command which triggered the function call

Returns constants.END_SPEC_EXECUTOR

Return type int

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If command END_SPEC != -1

execute_end_struct(*cmd*)

execute_free(*cmd*)

execute_get_random_rumber(*cmd*)

execute_get_wr_ptr(*cmd*)

execute_if(*cmd*)

execute_logic_op(*cmd*)

execute_loop(*cmd*)

execute_mv(*cmd*)

This command moves an immediate value to a register or copies the value of a register to another register

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – if the destination register is not correctly specified - the destination must be a register and the appropriate bit needs to be set in the specification

execute_nop(*cmd*)

This command executes no operation

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

execute_print_struct(*cmd*)

execute_print_txt(*cmd*)

execute_print_val(*cmd*)

execute_random_dist(*cmd*)

execute_read(*cmd*)

execute_read_param(*cmd*)

execute_reformat(*cmd*)

execute_reserve(*cmd*)

This command reserves a region and assigns some memory space to it

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises

- *data_specification.exceptions.DataSpecificationSyntaxError* – If there is an error in the command syntax
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the requested size of the region is beyond the available memory space

execute_reset_wr_ptr(*cmd*)

execute_set_wr_ptr(*cmd*)

execute_start_constructor(*cmd*)

execute_start_struct(*cmd*)

execute_struct_elem(*cmd*)

execute_switch_focus(*cmd*)

This command switches the focus to the desired, already allocated, memory region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises **data_specification.exceptions. DataSpecificationRegionUnfilledException** – If the focus is being switched to a region of memory which has been declared to be kept unfilled

execute_write(*cmd*)

This command writes the given value in the specified region a number of times as identified by either a value in the command or a register value

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If there is an error in the command syntax

execute_write_array(*cmd*)

This command writes an array of values in the specified region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

execute_write_param(*cmd*)

execute_write_param_component(*cmd*)

execute_write_struct(*cmd*)

mem_regions

mem_writer

memory_space

opcode

registers

space_allocated

spec_reader

src1_reg

src2_reg

struct_slots

use_dest_reg

use_src1_reg

use_src2_reg

data_specification.data_specification_generator module

class data_specification.data_specification_generator.**DataSpecificationGenerator**(*spec_writer*, *report_writer=Non*

Bases: object

Used to generate a data specification language file that can be executed to produce a memory image

Parameters

- **spec_writer** (Implementation of `data_specification.abstract_data_writer.AbstractDataWriter`) – The object to write the specification to
- **report_writer** (Implementation of `data_specification.abstract_data_writer.AbstractDataWriter`) – Determines if a text version of the specification is to be written

Raises *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

MAGIC_NUMBER = 2903706326

VERSION = 1

align_write_pointer(*log_block_size*, *log_block_size_is_register=False*, *return_register_id=None*)

Insert command to align the write pointer against a power-of-2 block size in bytes. Zeros are inserted in the intervening space

Parameters

- **log_block_size** (*int*) –
  - If log_block_size_is_register is False, log to base 2 of the block size (e.g. The write pointer will be moved so that it is a multiple of 2^(log_block_size)), between 0 and 32
  - If log_block_size_is_register is True, the id of the register containing log to the base 2 of the block size, between 0 and 15
- **log_block_size_is_register** (*bool*) – Indicates if log_block_size is a register id
- **return_register_id** (*int*) – The id of a register where the write pointer will be written to once it has been updated, between 0 and 15or None if no such writing is to be done

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If log_block_size_is_register is False, and log_block_size is not within the allowed range
  - If log_block_size_is_register is True and log_block_size is not a valid register id
- **data_specification.exceptions. DataSpecificationRegionOutOfBoundsException** – If the move of the pointer would put it outside of the current region
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected

break_loop()

Insert command to break out of a loop before it has completed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no loop in operation at this point

call_arithmetic_operation(*register_id*, *operand_1*, *operation*, *operand_2*, *signed*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform an arithmetic operation on two signed or unsigned values and store the result in a register

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operation** (ArithmeticOperation) – The operation to perform
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found, between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **signed** (*bool*) – Indicates if the values should be considered signed
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If operation is not a known operation

call_function (*function_id*, *structure_ids*)
Insert command to call a function

Parameters

- **function_id** (*int*) – The id of a previously defined function, between 0 and 31
- **structure_ids** (*list of int*) – A list of structure_ids that will be passed into the function, each between 0 and 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If the function id is not valid
  - If any of the structure ids are not valid
- **data_specification.exceptions. DataSpecificationNotAllocatedException** –
  - If a function has not been defined with the given id
  - If no structure has been defined with one of the ids in structure_ids
- **data_specification.exceptions. DataSpecificationWrongParameterNumberExcepti** – If a function is called with a wrong number of parameters
- **data_specification.exceptions. DataSpecificationDuplicateParameterException** – If a function is called with duplicate parameters

call_random_distribution (*distribution_id*, *register_id*)
Insert command to get the next random number from a random distribution, placing the result in a register to be used in a future call

Parameters

- **distribution_id** (*int*) – The id of the random distribution to call between 0 and 63
- **register_id** (*int*) – The id of the register to store the result in between 0 and 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the random distribution id was not previously declared

- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti**
  – If the distribution_id or register_id specified was out of range

comment (*comment*)

Write a comment to the text version of the specification. Note that this is ignored by the binary file

Parameters **comment** (*str*) – The comment to write

Returns Nothing is returned

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails

conditionals

copy_structure (*source_structure_id*, *destination_structure_id*, *source_id_is_register=False*, *destination_id_is_register=False*)

Insert command to copy a structure, possibly overwriting another structure

Parameters

- **source_structure_id** (*int*) –
  - If source_id_is_register is True, the id of the register holding the source structure id, between 0 and 15
  - Otherwise, the id of the source structure, between 0 and 15
- **destination_structure_id** (*int*) –
  - If destination_id_is_register is True, the id of the register holding the destination structure id, between 0 and 15
  - If destination_id_is_register is False, the id of the destination structure, between 0 and 15
- **source_id_is_register** (*bool*) – Indicates if source_structure_id is a register id
- **destination_id_is_register** (*bool*) – Indicates if destination_structure_id is a register id

Returns The id of the copied structure, between 0 and 15

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If source_id_is_register is True and source_structure_id is not a valid register id
  - If destination_id_is_register is True and destination_structure_id is not a valid register id
  - If source_id_is_register is False and source_structure_id is not a valid structure id

- If destination_id_is_register is False and destination_structure_id is not a valid structure id

- **data_specification.exceptions. DataSpecificationNotAllocatedException** –
  - If no structure with id source_structure_id has been allocated

copy_structure_parameter(*source_structure_id*, *source_parameter_index*, *destination_id*, *destination_parameter_index=None*, *destination_is_register=False*)

Insert command to copy the value of a parameter from one structure to another

Parameters

- **source_structure_id** (*int*) – The id of the source structure, between 0 and 15
- **source_parameter_index** (*int*) – The index of the parameter in the source structure
- **destination_id** (*int*) – The id of the destination structure, or the id of the destination register, between 0 and 15
- **destination_parameter_index** (*int*) – The index of the parameter in the destination structure. Ignored when writing to a register.
- **destination_is_register** (*boolean*) – Indicates whether the destination is a structure or a register.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If source_structure_id is not a valid structure id
  - If destination_id is not a valid structure id
  - If source_parameter_index is not a valid parameter index in the source structure
  - If destination_parameter_index is not a valid parameter index in the destination structure
- **data_specification.exceptions. DataSpecificationNotAllocatedException** –
  - If no structure with id destination_id has been allocated
  - If no structure with id source_structure_id has been allocated

create_cmd(*data*, *data_type=<DataType.UINT32: 2>*)

Creates command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Parameters

- **data** (*float*) – the data to write as a float.

- **data_type** (DataType) – the type to convert data to

Returns cmd_word_list; list of binary words to be added to the binary data specification file and cmd_string; string describing the command to be added to the report for the data specification file

Return type (bytearray, str)

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register id
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationInvalidSizeException** – If the data size is invalid

current_region

declare_random_number_generator(*rng_id*, *rng_type*, *seed*)

Insert command to declare a random number generator

Parameters

- **rng_id** (*int*) – The id of the random number generator
- **rng_type** (RandomNumberGenerator) – The type of the random number generator
- **seed** (*int*) – The seed of the random number generator >= 0

Returns The id of the created random number generator, between 0 and 15

Return type int

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions.DataSpecification** – If there is no more space for a new generator
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the rng_type is not one of the allowed values
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –

- If the seed is too big or too small
- If the rng_id is not in the allowed range

- **data_specification.exceptions .DataSpecificationRNGInUseException** – If the random number generator with the given id has already been defined

declare_uniform_random_distribution(*distribution_id*, *structure_id*, *rng_id*, *min_value*, *max_value*)

Insert commands to declare a uniform random distribution

Parameters

- **distribution_id** – id of the distribution being set up
- **distribution_id** – int
- **structure_id** (*int*) – id of an empty structure slot to fill with the uniform random distribution data
- **rng_id** (*int*) – The id of the random number generator, between 0 and 15
- **min_value** (*float*) – The minimum value that should be returned from the distribution between -32768.0 and max_value
- **max_value** (*float*) – The maximum value that should be returned from the distribution between min_value and 32767.9999847

Returns The id of the created uniform random distribution to be used in future calls of the distribution between 0 and 63

Return type int

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- *data_specification.exceptions.DataSpecificationNoMoreException* – If there is no more space for a new random distribution
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the requested rng_id has not been allocated
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If rng_id, structure_id, min_value or max_value is out of range
- **data_specification.exceptions. DataSpecificationStructureInUseException** – If structure structure_id is already defined

define_break()

Insert command to stop execution with an exception (for debugging)

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

define_structure(*structure_id*, *parameters*)

Insert commands to define a data structure

Parameters

- **structure_id** (*int*) – the id of the structure to create, between 0 and 15
- **parameters** (list of (str, DataType, float)) – A list of between 1 and 255 tuples of (label, data_type, value) where: * label is the label of the element (for debugging) * data_type is the data type of the element * value is the value of the element, or None if no value is to be assigned

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- *data_specification.exceptions.DataSpecificationNoMoreException* – If there are no more spaces for new structures
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If there are an incorrect number of parameters
  - If the size of one of the tuples is incorrect
  - If one of the values to be assigned has an integer data_type but has a fractional part
  - If one of the values to be assigned would overflow its data_type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If one of the data types in the structure is unknown

else_conditional()

Insert command for the else of an if...then...else construct. If the condition of the conditional evaluates to false, the statements up between the conditional and the insertion of this "else" are skipped, and the statements from this point until the end of the conditional are executed.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no conditional in operation at this point

end_conditional()

Insert command to mark the end of an if...then...else construct

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no conditional in operation at this point

end_function()

Insert command to mark the end of a function definition

Returns Nothing is returned

Return type None

Raises **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no function being defined at this point

end_loop()

Insert command to indicate that this is the end of the loop. Commands between the start of the loop and this command will be repeated.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no loop in operation at this point

end_specification(*close_writer=True*)

Insert a command to indicate that the specification has finished and finish writing

Parameters **close_writer** (*bool*) – Indicates whether to close the underlying writer(s)

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

free_memory_region(*region*)

Insert command to free a previously reserved memory region

Parameters **region** (*int*) – The number of the region to free, from 0 to 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the region was not reserved
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the region requested was out of the allowed range

function

get_structure_value(*destination_id*, *structure_id*, *parameter_index*, *parameter_index_is_register=False*)

Insert command to get a value from a structure. The value is copied in a register.

Parameters

- **destination_id** (*int*) – The id of the destination register
- **structure_id** (*int*) – The id of the source structure
- **parameter_index** (*int*) – The id of the parameter/element to copy
- **parameter_index_is_register** (*bool*) – True if the index of the structure is contained in a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If destination_id is not the id of a valid register
  - If parameter_index_is_register is True and parameter_index is not a valid register id
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the structure requested has not been declared

instruction_counter

logical_and(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical AND operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in

- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_left_shift (*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical left shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_not(*register_id*, *operand*, *operand_is_register=False*)

Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand** (*int*) –
  - If operand_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_is_register is False, a 32-bit value
- **operand_is_register** (*bool*) – Indicates if operand_1 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_is_register is True and operand_1 is not a valid register id

logical_or(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical OR operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_right_shift(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical right shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_xor(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –

- If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
- If operand_2_is_register is False, a 32-bit value

- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

mem_slot

no_operation()

Insert command to execute nothing

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

ongoing_function_definition

ongoing_loop

print_struct(*structure_id*, *structure_id_is_register=False*)

Insert command to print out a structure (for debugging)

Parameters

- **structure_id** (*int*) –
  - If structure_id_is_register is True, the id of the register containing the id of the structure to print, between 0 and 15
  - If structure_id_is_register is False, the id of the structure to print, between 0 and 15
- **structure_id_is_register** (*bool*) – Indicates if the structure_id is a register

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If structure_id_is_register is True and structure_id is not a valid register id
  - If structure_id_is_register is False and structure_id is not a valid structure id
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If structure_id_is_register is False and structure_id is the id of a structure that has not been allocated

print_text (*text*)

Insert command to print some text (for debugging)

Parameters **text** (*str*) – The text to write (max 12 characters)

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

print_value (*value*, *value_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to print out a value (for debugging)

Parameters

- **value** (*float*) –
  - If value_is_register is True, the id of the register containing the value to print
  - If value_is_register is False, the value to print as a float
- **value_is_register** (*bool*) – Indicates if the value is a register
- **data_type** (DataType) – The type of the data to be printed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If value_is_register is True and value is not a valid register id
  - If value_is_register is False, the data_type is an integer type and value has a fractional part
  - If value_is_register is False and the value would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** –

– If data_type is not a valid data type

random_distribution

read_value (*dest_id*, *data_type*)

Insert command to read a value from the current write pointer, causing the write pointer to move by the number of bytes read. The data is stored in a register passed as argument.

Parameters

- **dest_id** (*int*) – The id of the destination register.
- **data_type** (DataType) – The type of the data to be read.

Returns Nothing is returned

Return type None

region_sizes

A list of sizes of each region that has been reserved. Note that the list will include 0s for each non-reserved region.

report_writer

reserve_memory_region (*region*, *size*, *label=None*, *empty=False*, *shrink=True*)

Insert command to reserve a memory region

Parameters

- **region** (*int*) – The number of the region to reserve, from 0 to 15
- **size** (*int*) – The size to reserve for the region in bytes
- **label** (*str*) – An optional label for the region
- **empty** (*bool*) – Specifies if the region will be left empty
- **shrink** (*bool*) – Specifies if the region will be compressed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationRegionInUseException** – If the region was already reserved
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the region requested was out of the allowed range, or that the size was too big to fit in SDRAM

rng

save_write_pointer (*register_id*)

Insert command to save the write pointer to a register

Parameters **register_id** (*int*) – The id of the register to assign, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the register_id is not a valid register id
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected

set_register_value(*register_id*, *data*, *data_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to set the value of a register

Parameters

- **register_id** (*int*) – The id of the register to assign, between 0 and 15
- **data** (*float*) –
  - If data_is_register is True, the register id containing the data to set, between 0 and 15
  - If data_is_register is False, the data as a float
- **data_is_register** (*bool*) – Indicates if data is a register id
- **data_type** (DataType) – The type of the data to be assigned

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If register_id is not a valid register_id
  - If data_is_register is True, and data is not a valid register id
  - If data_is_register is False, data_type is an integer type and data has a fractional part
  - If data_is_register if False, and data would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known

set_structure_value(*structure_id*, *parameter_index*, *value*, *data_type*, *value_is_register=False*)

Insert command to set a value in a structure

Parameters

- **structure_id** (*int*) –
  - If called outside of a function, the id of the structure between 0 and 15
  - If called within a function, the id of the structure argument, between 0 and 4

- **parameter_index** (*int*) – The index of the value to assign in the structure, between 0 and the number of parameters declared in the structure
- **value** (*float*) –
  - If value_is_register is False, the value to assign at the selected position as a float
  - If value_is_register is True, the id of the register containing the value to assign to the position, between 0 and 15
- **data_type** (DataType) – type of the data to be stored in the structure. If parameter value_is_register is set to true, this variable is disregarded
- **value_is_register** (*bool*) – Identifies if value identifies a register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If value_is_register is False, and the data type of the structure value is an integer, and the value has a fractional part
  - If value_is_register is False, and value would overflow the position in the structure
  - If value_is_register is True, and value is not a valid register id
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the structure requested has not been declared

set_write_pointer(*address*, *address_is_register=False*, *relative_to_current=False*)

Insert command to set the position of the write pointer within the current region

Parameters

- **address** (*int*) –
  - If address_is_register is True, the id of the register containing the address to move to
  - If address_is_register is False, the address to move the write pointer to
- **address_is_register** (*bool*) – Indicates if the the address is a register id
- **relative_to_current** (*bool*) – Indicates if the address is to be added to the current address, or used as an absolute address from the start of the current region

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the address_is_register is True and address is not a valid register id
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected

spec_writer

start_conditional (*register_id*, *condition*, *value*, *value_is_register=False*)

Insert command to start a conditional if...then...else construct. If the condition evaluates to true, the statement is executed up to the next else statement, or the end of the conditional, whichever comes first.

Parameters

- **register_id** (*int*) – The id of a register to compare the value of
- **condition** (Condition) – The condition which must be true to execute the following commands
- **value** (*int*) –
  - If value_is_register is False, the value to compare to the value in the register
  - If value_is_register is True, the id of the register containing the value to compare, between 0 and 15
- **value_is_register** (*bool*) – Indicates if value is a register id

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If the register_id is not a valid register id
  - if value_is_register is True and value is not a valid register id
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the condition is not a valid condition

start_function (*function_id*, *argument_by_value*)

Insert command to start a function definition, with up to 5 arguments, which are the ids of structures to be used within the function, each of which can be passed by reference or by value. In the commands following this command up to the end_function command, structures can only be referenced using the numbers 1 to 5 which address the arguments, rather than the original structure ids

Parameters

- **function_id** (*int*) – The id of the function currently defined.

- **argument_by_value** (*list of bool*) – A list of up to 5 booleans indicating if the structure to be passed as an argument is to be passed by reference (i.e. changes made within the function are persisted) or by value (i.e. changes made within the function are lost when the function exits. The number of arguments is determined by the length of this list.

Returns The id of the function, between 0 and 31

Return type int

Raises

- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If there are too many items in the list of arguments
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is already a function being defined at this point
- **data_specification.exceptions. DataSpecificationFunctionInUse** – If the function is already defined

start_loop(*counter_register_id*, *start*, *end*, *increment=1*, *start_is_register=False*, *end_is_register=False*, *increment_is_register=False*)

Insert command to start a loop

Parameters

- **counter_register_id** (*int*) – The id of the register to use as the loop counter, between 0 and 15
- **start** (*int*) –
  - If start_is_register is False, the number at which to start the loop counter, >= 0
  - If start_is_register is True, the id of the register containing the number at which to start the loop counter, between 0 and 15
- **end** (*int*) –
  - If end_is_register is False, the number which the loop counter must reach to stop the loop i.e. the loop will run while the contents of counter_register < end, >= 0
  - If end_is_register is True, the id of the register containing the number at which to stop the loop, between 0 and 15
- **increment** (*int*) –
  - If increment_is_register is False, the amount by which to increment the loop counter on each run of the loop, >= 0
  - If increment_is_register is True, the id of the register containing the amount by which to increment the loop counter on each run of the loop, between 0 and 15
- **start_is_register** (*bool*) – Indicates if start is a register id
- **end_is_register** (*bool*) – Indicates if end is a register id
- **increment_is_register** (*bool*) – Indicates if increment is a register id

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If counter_register_id is not a valid register id
  - If start_is_register is True and increment is not a valid register_id
  - If end_is_register is True and increment is not a valid register_id
  - If increment_is_register is True, and increment is not a valid register_id
  - If start_is_register is False and start is not in the allowed range
  - If end_is_register is False and end is not in the allowed range
  - If increment_is_register is False and increment is not in the allowed range

struct_slot

switch_write_focus (*region*)

Insert command to switch the region being written to

Parameters **region** (*int*) – The id of the region to switch to, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the region identifier is not valid
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the region has not been allocated
- **data_specification.exceptions. DataSpecificationRegionUnfilledException** – If the selected region should not be filled

txt_indent

write_array (*array_values*, *data_type=<DataType.UINT32: 2>*)

Insert command to write an array, causing the write pointer to move on by (data type size * the array size), in bytes.

Parameters

- **array_values** (*list of unsigned int*) – An array of words to be written
- **data_type** (data_specification.enums.DataType) – Type of data contained in the array

Returns The position of the write pointer within the current region, in bytes from the start of the region

Return type int

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been previously selected

write_cmd(*cmd_word_list*, *cmd_string*)

Writes cmd values created previously created and cacheed

see: create_cmd which is the ONLY way the parameters should have been created.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file

Returns Nothing is returned

Return type None

Raises

- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

write_command_to_files(*cmd_word_list*, *cmd_string*, *indent=False*, *outdent=False*, *no_instruction_number=False*)

Writes the binary command to the binary output file and, if the user has requested a text output for debug purposes, also write the text version to the text file. Setting the optional parameter 'indent' to True causes subsequent commands to be indented by two spaces relative to this one. Similarly, setting 'outdent' to True, reverses this spacing.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file
- **indent** (*bool*) – if the following lines need to be indented
- **outdent** (*bool*) – if the following lines need to be out-dented
- **no_instruction_number** – if each report line should include also the address of the command in the file
- **no_instruction_number** – bool

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails

write_repeated_value(*data*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Parameters

- **data** (*float*) – the data to write as a float.
- **repeats** (*int*) –
  - If repeats_is_register is False, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
  - If repeats_is_register is True, this parameter identifies the register that contains the number of repeats.
- **repeats_is_register** (*bool*) – Indicates if the parameter repeats identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register id
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationInvalidSizeException** – If the data size is invalid
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

write_structure(*structure_id*, *repeats=1*, *repeats_is_register=False*)

Insert command to write a structure to the current write pointer, causing the current write pointer to move on by the number of bytes needed to represent the structure

Parameters

- **structure_id** (*int*) –
  - If called within a function, the id of the structure to write, between 0 and 15

- If called outside of a function, the id of the structure argument, between 0 and 5

- **repeats** (*int*) –
  - If repeats_is_register is True, the id of the register containing the number of repeats, between 0 and 15
  - If repeats_is_register is False, the number of repeats to write, between 0 and 255
- **repeats_is_register** (*bool*) – Identifies if repeats identifies a register

Returns No value returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_is_register is False and structure_id is not a valid id
  - If repeats_is_register is True and structure_id
  - If the number of repeats is out of range
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to
- **data_specification.exceptions. DataSpecificationRegionExhaustedException** – If the selected region has no more space

write_value (*data*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Note: This method used to have two extra parameters repeats and repeats_is_register. They have been removed here. If you need then use write_repeated_value

Parameters

- **data** (*float*) – the data to write as a float.
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –

- If repeats_register is None, and repeats is out of range
- If repeats_register is not a valid register id
- If data_type is an integer type, and data has a fractional part
- If data would overflow the data type

- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationInvalidSizeException** – If the data size is invalid
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

write_value_from_register(*data_register*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times at the write pointer of the current memory region, causing it to move. The data is contained in a register whose id is passed to the function

Parameters

- **data_register** (*int*) – Identifies the register in which the data is stored.
- **repeats** (*int*) –
  - If repeats_register is None, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
  - If repeats_register is not None (i.e. has an integer value), the content of this parameter is disregarded
- **repeats_is_register** (*None or int*) – Identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type of the data held in the register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register id
  - If data_register is not a valid register id
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

- **data_specification.exceptions. DataSpecificationRegionExhaustedException** – If the selected region has no more space

data_specification.exceptions module

exception data_specification.exceptions.**DataSpecificationDuplicateParameterException**(*command*, *parameters*)

Bases: *data_specification.exceptions.DataSpecificationException*

And exception that indicates that a command has been called with a duplicate parameter, which shouldn't be allowed.

Parameters

- **command** (*int*) – The command called with duplicate parameters
- **parameters** (*list*) – The parameters used to call the function

exception data_specification.exceptions.**DataSpecificationException**(*message*)

Bases: exceptions.Exception

A general purpose exception indicating that something went wrong when interacting with a Data Specification

Parameters **message** (*str*) – message reporting the exception occurred

exception data_specification.exceptions.**DataSpecificationFunctionInUse**(*function_id*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a function is already defined

Parameters **function_id** (*int*) – The id of the function

exception data_specification.exceptions.**DataSpecificationInvalidCommandException**(*command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the command being requested cannot be executed at this point in the specification

Parameters **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationInvalidOperationException**(*operation_type*, *requested_operat*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the operation of the type given type is not available

Parameters

- **operation_type** (*str*) – The type of operation requested (i.e. arithmetic or logic)
- **requested_operation_id** (*int*) – The id of the requested operation
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationInvalidSizeException**(*type_name*, *type_size*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the size of the requested type is invalid

Parameters

- **type_name** (*str*) – The name of the requested type
- **type_size** (*int*) – The size of the requested variable
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationNoMoreException**(*space_available*, *space_required*, *region*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that there is no more space for the requested item

Parameters

- **space_available** (*int*) – The space available in the region
- **space_required** (*int*) – The space requested by the write command

exception data_specification.exceptions.**DataSpecificationNoRegionSelectedException**(*command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a memory region has not been selected

Parameters **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationNotAllocatedException**(*item_type*, *item_id*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that an item is being used that has not been allocated

Parameters

- **item_type** (*str*) – The type of the item being used
- **item_id** (*int*) – The id of the item being used
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationParameterOutOfBoundsException**(*parameter*, *value*, *range_min*, *range_max*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a parameter value was outside of the allowed bounds

Parameters

- **parameter** (*str*) – The parameter that is out of bounds

- **value** (*float*) – The value specified
- **range_min** (*float*) – The minimum allowed value
- **range_max** (*float*) – The maximum allowed value
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationRNGInUseException**(*rng_id*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a random number generator is already defined

Parameters **rng_id** (*int*) – The id of the rng

exception data_specification.exceptions.**DataSpecificationRandomNumberDistributionInUseExcepti**

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a random number distribution is already defined

Parameters **rng_id** (*int*) – The id of the random number distribution

exception data_specification.exceptions.**DataSpecificationRegionExhaustedException**(*region*, *region_size*, *allocated_size*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a region has run out of memory whilst some data is being written

Parameters

- **region** (*int*) – The region that was being written to
- **region_size** (*int*) – The originally requested size of the region that has run out of space, in bytes
- **allocated_size** (*int*) – The amount of the originally requested space that has already been allocated, in bytes
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationRegionInUseException**(*region*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a region has already been allocated

Parameters **region** (*int*) – The region that was already allocated

exception data_specification.exceptions.**DataSpecificationRegionNotAllocated**(*region*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when trying to write to an unallocated region of memory

Parameters

- **region** –
- **command** –

exception data_specification.exceptions.**DataSpecificationRegionOutOfBoundsException**(*region*, *region_size*, *requested_offset*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that an offset into a region is out of bounds for that region

Parameters

- **region** (*int*) – The region that was being offset into
- **region_size** (*int*) – The originally requested size of the region in question, in bytes
- **requested_offset** (*int*) – The offset being requested, in bytes
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationRegionUnfilledException**(*region*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a memory region is being used that was originally requested to be unfilled

Parameters

- **region** (*int*) – The region that was requested as unfilled
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationStructureInUseException**(*structure*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a structure has already been defined

Parameters **structure** (*int*) – The structure that was already allocated

exception data_specification.exceptions.**DataSpecificationSyntaxError**(*message*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when a command read from the data specification file shows an inconsistency in the binary content

Parameters **message** (*str*) – message describing the error occurred

exception data_specification.exceptions.**DataSpecificationTablePointerOutOfMemory**(*memory_available*, *memory_required*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when building the table pointer as header of the data generated by the spec executor. This message is printed in case the memory available is not enough to contain the pointer table for each of the allocated regions

Parameters

- **memory_available** (*int*) – on-chip memory available
- **memory_required** (*int*) – on-chip memory required to complete the execution of the specification file

exception data_specification.exceptions.**DataSpecificationTypeMismatchException**(*command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a type mismatch has occurred

Parameters **command** (*int*) – The command that generated the exception

exception data_specification.exceptions.**DataSpecificationUnknownConditionException**(*condition_id*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

And exception which is triggered in case the condition in an IF test does not exist in the list of possible conditions

Parameters

- **condition_id** (*int*) – id of the condition being requested
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationUnknownTypeException**(*type_id*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the value of the requested type is unknown

Parameters

- **type_id** (*int*) – The id of the requested type
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationUnknownTypeLengthException**(*data_length*, *command*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that the value of the requested type is unknown

Parameters

- **data_length** (*int*) – the length of the requested type
- **command** (*str*) – The command being executed

exception data_specification.exceptions.**DataSpecificationWrongParameterNumberException**(*function_ no_of_pa pa- ram- e- ters*)

Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a function has been called with a wrong number of parameters.

Parameters

- **function_id** (*int*) – The id of the function
- **parameters** (*list*) – The parameters used in the function call
- **no_of_parameters_required** – The number of parameters required by the function

exception data_specification.exceptions.**DataUndefinedWriterException**(*message*)

Bases: exceptions.Exception

an exception that indicates that the file data writer has not been initialised

Parameters **message** (*str*) – A message to indicate what when wrong

exception data_specification.exceptions.**ExecuteBreakInstruction**(*address*, *filename*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when a BREAK instruction is found in the data specification

Parameters

- **address** (*int*) – address of the data specification being executed at the time of breakpoint
- **filename** – file being parsed
- **filename** – str

exception data_specification.exceptions.**NestedFunctionException**
Bases: *data_specification.exceptions.DataSpecificationException*

An exception that indicates that a function is being defined within the context of another function definition

exception data_specification.exceptions.**UnimplementedDSECommand**(*command*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when trying to execute an unimplemented command

Parameters **command** (*str*) – Command attempted to be executed by the DSE

exception data_specification.exceptions.**UnimplementedDSGCommand**(*command*)
Bases: *data_specification.exceptions.DataSpecificationException*

An exception which occurs when trying to write an unimplemented command

Parameters **command** (*str*) – Command attempted to be generated by the DSG

data_specification.memory_region module

class data_specification.memory_region.**MemoryRegion**(*memory_pointer*, *unfilled*, *size*)
Bases: object

Memory region storage object

Parameters

- **memory_pointer** – the write pointer position
- **unfilled** – if the region needs to be filled when written

Return type None

Raises **None** – this method does not raise any known exception

allocated_size
property method for the size of the region

Returns the size of the region

Return type int

Raises **None** – this method does not raise any known exception

increment_write_pointer(*n_bytes*)

max_write_pointer

memory_pointer
property method to retrieve memory write pointer

Returns the memory pointer of the region

Return type int

Raises **None** – this method does not raise any known exception

region_data
the container which holds the data written in this region

Returns the region data container

Return type bytearray

Raises **None** – this method does not raise any known exception

unfilled
property method to retrieve if the region is filled

Returns the the

Return type None

Raises **None** – this method does not raise any known exception

write_pointer

data_specification.memory_region_collection module

class `data_specification.memory_region_collection.`**MemoryRegionCollection**(*n_regions*)
Bases: `object`

Collection of memory regions.

Create a new MemoryRegionCollection with the given number of regions.

count_used_regions()

is_empty(*region*)

is_unfilled(*region*)

needs_to_write_region(*region_id*)
helper method which determines if a unfilled region actually needs to be written (optimisation to stop large data files)

Parameters **region_id** – the region id to which the test is being ran on

Returns a boolean stating if the region needs to be written

Return type boolean

Raises ***DataSpecificationNoRegionSelectedException*** – when the id is beyond the expected region range

regions

data_specification.utility_calls module

utility calls for interpreting bits of the dsg

`data_specification.utility_calls.`**`get_data_spec_and_file_writer_filename`**(*processor_chip_x, processor_chip_y, processor_id, hostname, report_directory, write_text_specs, application_run_time_report_folde*

Encapsulates the creation of the dsg writer and the file paths

Parameters

- **processor_chip_x** (*int*) – x-coordinate of the chip
- **processor_chip_y** (*int*) – y-coordinate of the chip
- **processor_id** (*int*) – The processor ID
- **hostname** (*str*) – The hostname of the spinnaker machine
- **report_directory** (*file path*) – the directory for the reports folder
- **write_text_specs** (*bool*) – True if a textual version of the specification should be written
- **application_run_time_report_folder** (*str*) – The folder to contain the resulting specification files

Returns the filename of the data writer and the data specification object

Return type str, *data_specification.DataSpecificationGenerator*

`data_specification.utility_calls.`**`get_data_spec_file_path`**(*processor_chip_x, processor_chip_y, processor_id, hostname, application_run_time_folder*)

Gets the file path for storing the dsg data

Parameters

- **processor_chip_x** (*int*) – The x-coordinate of a chip
- **processor_chip_y** – The y-coordinate of a chip :type processor_chip_y: int
- **processor_id** (*int*) – The processor ID
- **hostname** (*str*) – The hostname of the spinnaker machine

Returns the filename of the data writer and the data specification object

Return type str, *data_specification.DataSpecificationGenerator*

`data_specification.utility_calls.`**`get_region_base_address_offset`**(*app_data_base_address, region*)

Find the address of the of a given region for the dsg

Parameters

- **app_data_base_address** – base address for the core
- **region** – the region id we're looking for

Module contents

Used to generate memory images from a set of instructions.

The main part of this package is the *data_specification.data_specification_generator.DataSpecificationGenerator* class. This is used to generate a "Data Specification", which can then be executed to produce a memory image. This package also handles this function if required, through the *data_specification.data_specification_executor.DataSpecificationExecutor* class.

Functional Requirements

- Creation of a Data Specification Language file which can be executed to produce a memory image.
  - Any errors that can be checked during the creation of the specification should throw an exception.
  - It will be impossible to detect all errors at creation time.
  - There should be no assumption of where the data specification is be stored, although a default provision of a way to write the specification to a file is acceptable.
- Execution of a Data Specification Language file, producing a memory image.
  - This should detect any errors during execution and report them, halting the execution.
  - There should be no assumption of where the data specification is read from, although a default provision of a way to read the specification from a file is acceptable.

Use Cases

There are a number of use-cases of this library:

- `DataSpecificationGenerator` is used to create a compressed memory image which can be expanded later, to reduce the amount of data that needs to be transferred over a slow connection.
- `DataSpecificationExecutor` is used to execute a previously generated specification at the receiving end of a slow connection.

class `data_specification.`**`DataSpecificationExecutor`**(*spec_reader*, *memory_space*)

Bases: `object`

Used to execute a data specification language file to produce a memory image

Parameters

- **spec_reader** (`data_specification.abstract_data_reader.AbstractDataReader`) – The object to read the specification language file from
- **memory_space** (*int*) – memory available on the destination architecture

Raises

- ***spinn_storage_handlers.exceptions.DataReadException*** – If a read from external storage fails

- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

dsef

execute()

Executes the specification

Returns The number of bytes used by the image and the number of bytes written by the image

Return type int

Raises

- *spinn_storage_handlers.exceptions.DataReadException* – If a read from external storage fails
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- *data_specification.exceptions.DataSpecificationException* – If there is an error when executing the specification
- **data_specification.exceptions. DataSpecificationTablePointerOutOfMemory** – If the table pointer generated as data header exceeds the size of the available memory

get_constructed_data_size()

Return the size of the data that will be written to memory

Returns size of the data that will be written to memory

Return type unsigned int

get_header()

Get the header of the data as a numpy array

get_pointer_table(*start_address*)

Get the pointer table as a numpy array

Parameters **start_address** – The base address of the data to be written

get_region(*region_id*)

Get a region with a given id

Parameters **region_id** (*int*) – The id of the region to get

Returns The region, or None if the region was not allocated

spec_reader

class data_specification.**DataSpecificationExecutorFunctions**(*spec_reader*, *memory_space*)

Bases: object

This class includes the function related to each of the commands of the data specification file.

Parameters

- **spec_reader** (data_specification.abstract_data_reader.AbstractDataReader) – The object to read the specification language file from
- **memory_space** (*int*) – Memory space available for the data to be generated

current_region

data_len

dest_reg

execute_align_wr_ptr(*cmd*)

execute_arith_op(*cmd*)

execute_block_copy(*cmd*)

execute_break(*cmd*)

This command raises an exception to stop the execution of the data spec executor (DSE)

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.ExecuteBreakInstruction* – Raises the exception to break the execution of the DSE

execute_break_loop(*cmd*)

execute_construct(*cmd*)

execute_copy_param(*cmd*)

execute_copy_struct(*cmd*)

execute_declare_rng(*cmd*)

execute_else(*cmd*)

execute_end_constructor(*cmd*)

execute_end_if(*cmd*)

execute_end_loop(*cmd*)

execute_end_spec(*cmd*)

Return the value which terminates the data spec executor

Parameters **cmd** (*int*) – the command which triggered the function call

Returns constants.END_SPEC_EXECUTOR

Return type int

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If command END_SPEC != -1

execute_end_struct(*cmd*)

execute_free(*cmd*)

execute_get_random_rumber(*cmd*)

execute_get_wr_ptr(*cmd*)

execute_if(*cmd*)

execute_logic_op(*cmd*)

execute_loop(*cmd*)

execute_mv(*cmd*)

This command moves an immediate value to a register or copies the value of a register to another register

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – if the destination register is not correctly specified - the destination must be a register and the appropriate bit needs to be set in the specification

execute_nop(*cmd*)

This command executes no operation

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

execute_print_struct(*cmd*)

execute_print_txt(*cmd*)

execute_print_val(*cmd*)

execute_random_dist(*cmd*)

execute_read(*cmd*)

execute_read_param(*cmd*)

execute_reformat(*cmd*)

execute_reserve(*cmd*)

This command reserves a region and assigns some memory space to it

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises

- *data_specification.exceptions.DataSpecificationSyntaxError* – If there is an error in the command syntax
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the requested size of the region is beyond the available memory space

execute_reset_wr_ptr(*cmd*)

execute_set_wr_ptr(*cmd*)

execute_start_constructor(*cmd*)

execute_start_struct(*cmd*)

execute_struct_elem(*cmd*)

execute_switch_focus(*cmd*)

This command switches the focus to the desired, already allocated, memory region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises **data_specification.exceptions. DataSpecificationRegionUnfilledException** – If the focus is being switched to a region of memory which has been declared to be kept unfilled

execute_write(*cmd*)

This command writes the given value in the specified region a number of times as identified by either a value in the command or a register value

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

Raises *data_specification.exceptions.DataSpecificationSyntaxError* – If there is an error in the command syntax

execute_write_array(*cmd*)

This command writes an array of values in the specified region

Parameters **cmd** (*int*) – the command which triggered the function call

Returns No value returned

Return type None

execute_write_param(*cmd*)

execute_write_param_component(*cmd*)

execute_write_struct(*cmd*)

mem_regions

mem_writer

memory_space

opcode

registers

space_allocated

spec_reader

src1_reg

src2_reg

struct_slots

use_dest_reg

use_src1_reg

use_src2_reg

class data_specification.**DataSpecificationGenerator**(*spec_writer*, *report_writer=None*)

Bases: object

Used to generate a data specification language file that can be executed to produce a memory image

Parameters

- **spec_writer** (Implementation of data_specification.abstract_data_writer.AbstractDataWriter) – The object to write the specification to
- **report_writer** (Implementation of data_specification.abstract_data_writer.AbstractDataWriter) – Determines if a text version of the specification is to be written

Raises *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

MAGIC_NUMBER = 2903706326

VERSION = 1

align_write_pointer(*log_block_size*, *log_block_size_is_register=False*, *return_register_id=None*)

Insert command to align the write pointer against a power-of-2 block size in bytes. Zeros are inserted in the intervening space

Parameters

- **log_block_size** (*int*) –
  - If log_block_size_is_register is False, log to base 2 of the block size (e.g. The write pointer will be moved so that it is a multiple of 2^(log_block_size)), between 0 and 32
  - If log_block_size_is_register is True, the id of the register containing log to the base 2 of the block size, between 0 and 15
- **log_block_size_is_register** (*bool*) – Indicates if log_block_size is a register id
- **return_register_id** (*int*) – The id of a register where the write pointer will be written to once it has been updated, between 0 and 15or None if no such writing is to be done

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If log_block_size_is_register is False, and log_block_size is not within the allowed range
  - If log_block_size_is_register is True and log_block_size is not a valid register id
- **data_specification.exceptions. DataSpecificationRegionOutOfBoundsException** – If the move of the pointer would put it outside of the current region
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected

break_loop()

Insert command to break out of a loop before it has completed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no loop in operation at this point

call_arithmetic_operation(*register_id*, *operand_1*, *operation*, *operand_2*, *signed*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform an arithmetic operation on two signed or unsigned values and store the result in a register

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operation** (ArithmeticOperation) – The operation to perform
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found, between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **signed** (*bool*) – Indicates if the values should be considered signed
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If operation is not a known operation

call_function(*function_id*, *structure_ids*)

Insert command to call a function

Parameters

- **function_id** (*int*) – The id of a previously defined function, between 0 and 31
- **structure_ids** (*list of int*) – A list of structure_ids that will be passed into the function, each between 0 and 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If the function id is not valid
  - If any of the structure ids are not valid
- **data_specification.exceptions. DataSpecificationNotAllocatedException** –
  - If a function has not been defined with the given id
  - If no structure has been defined with one of the ids in structure_ids
- **data_specification.exceptions. DataSpecificationWrongParameterNumberExcepti** – If a function is called with a wrong number of parameters
- **data_specification.exceptions. DataSpecificationDuplicateParameterException** – If a function is called with duplicate parameters

call_random_distribution(*distribution_id*, *register_id*)

Insert command to get the next random number from a random distribution, placing the result in a register to be used in a future call

Parameters

- **distribution_id** (*int*) – The id of the random distribution to call between 0 and 63
- **register_id** (*int*) – The id of the register to store the result in between 0 and 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the random distribution id was not previously declared
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the distribution_id or register_id specified was out of range

comment(*comment*)

Write a comment to the text version of the specification. Note that this is ignored by the binary file

Parameters **comment** (*str*) – The comment to write

Returns Nothing is returned

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails

conditionals

copy_structure(*source_structure_id*, *destination_structure_id*, *source_id_is_register=False*, *destination_id_is_register=False*)

Insert command to copy a structure, possibly overwriting another structure

Parameters

- **source_structure_id** (*int*) –
  - If source_id_is_register is True, the id of the register holding the source structure id, between 0 and 15
  - Otherwise, the id of the source structure, between 0 and 15
- **destination_structure_id** (*int*) –
  - If destination_id_is_register is True, the id of the register holding the destination structure id, between 0 and 15
  - If destination_id_is_register is False, the id of the destination structure, between 0 and 15
- **source_id_is_register** (*bool*) – Indicates if source_structure_id is a register id
- **destination_id_is_register** (*bool*) – Indicates if destination_structure_id is a register id

Returns The id of the copied structure, between 0 and 15

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If source_id_is_register is True and source_structure_id is not a valid register id
  - If destination_id_is_register is True and destination_structure_id is not a valid register id
  - If source_id_is_register is False and source_structure_id is not a valid structure id
  - If destination_id_is_register is False and destination_structure_id is not a valid structure id
- **data_specification.exceptions. DataSpecificationNotAllocatedException** –
  - If no structure with id source_structure_id has been allocated

copy_structure_parameter(*source_structure_id*, *source_parameter_index*, *destination_id*, *destination_parameter_index=None*, *destination_is_register=False*)

Insert command to copy the value of a parameter from one structure to another

Parameters

- **source_structure_id** (*int*) – The id of the source structure, between 0 and 15
- **source_parameter_index** (*int*) – The index of the parameter in the source structure
- **destination_id** (*int*) – The id of the destination structure, or the id of the destination register, between 0 and 15
- **destination_parameter_index** (*int*) – The index of the parameter in the destination structure. Ignored when writing to a register.
- **destination_is_register** (*boolean*) – Indicates whether the destination is a structure or a register.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If source_structure_id is not a valid structure id
  - If destination_id is not a valid structure id
  - If source_parameter_index is not a valid parameter index in the source structure
  - If destination_parameter_index is not a valid parameter index in the destination structure
- **data_specification.exceptions. DataSpecificationNotAllocatedException** –
  - If no structure with id destination_id has been allocated
  - If no structure with id source_structure_id has been allocated

create_cmd(*data*, *data_type=<DataType.UINT32: 2>*)

Creates command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Parameters

- **data** (*float*) – the data to write as a float.
- **data_type** (DataType) – the type to convert data to

Returns cmd_word_list; list of binary words to be added to the binary data specification file and cmd_string; string describing the command to be added to the report for the data specification file

Return type (bytearray, str)

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register id
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationInvalidSizeException** – If the data size is invalid

current_region

declare_random_number_generator(*rng_id*, *rng_type*, *seed*)

Insert command to declare a random number generator

Parameters

- **rng_id** (*int*) – The id of the random number generator
- **rng_type** (`RandomNumberGenerator`) – The type of the random number generator
- **seed** (*int*) – The seed of the random number generator >= 0

Returns The id of the created random number generator, between 0 and 15

Return type int

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions.DataSpecification** – If there is no more space for a new generator
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the rng_type is not one of the allowed values
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If the seed is too big or too small
  - If the rng_id is not in the allowed range
- **data_specification.exceptions .DataSpecificationRNGInUseException** – If the random number generator with the given id has already been defined

declare_uniform_random_distribution(*distribution_id*, *structure_id*, *rng_id*, *min_value*, *max_value*)

Insert commands to declare a uniform random distribution

Parameters

- **distribution_id** – id of the distribution being set up

- **distribution_id** – int
- **structure_id** (*int*) – id of an empty structure slot to fill with the uniform random distribution data
- **rng_id** (*int*) – The id of the random number generator, between 0 and 15
- **min_value** (*float*) – The minimum value that should be returned from the distribution between -32768.0 and max_value
- **max_value** (*float*) – The maximum value that should be returned from the distribution between min_value and 32767.9999847

Returns The id of the created uniform random distribution to be used in future calls of the distribution between 0 and 63

Return type int

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- *data_specification.exceptions.DataSpecificationNoMoreException* – If there is no more space for a new random distribution
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the requested rng_id has not been allocated
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If rng_id, structure_id, min_value or max_value is out of range
- **data_specification.exceptions. DataSpecificationStructureInUseException** – If structure structure_id is already defined

define_break()

Insert command to stop execution with an exception (for debugging)

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

define_structure(*structure_id*, *parameters*)

Insert commands to define a data structure

Parameters

- **structure_id** (*int*) – the id of the structure to create, between 0 and 15
- **parameters** (list of (str, `DataType`, float)) – A list of between 1 and 255 tuples of (label, data_type, value) where: * label is the label of the element (for debugging) * data_type is the data type of the element * value is the value of the element, or None if no value is to be assigned

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- *data_specification.exceptions.DataSpecificationNoMoreException* – If there are no more spaces for new structures
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If there are an incorrect number of parameters
  - If the size of one of the tuples is incorrect
  - If one of the values to be assigned has an integer data_type but has a fractional part
  - If one of the values to be assigned would overflow its data_type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If one of the data types in the structure is unknown

else_conditional()

Insert command for the else of an if...then...else construct. If the condition of the conditional evaluates to false, the statements up between the conditional and the insertion of this "else" are skipped, and the statements from this point until the end of the conditional are executed.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no conditional in operation at this point

end_conditional()

Insert command to mark the end of an if...then...else construct

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no conditional in operation at this point

end_function()

Insert command to mark the end of a function definition

Returns Nothing is returned

Return type None

Raises **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no function being defined at this point

end_loop()

Insert command to indicate that this is the end of the loop. Commands between the start of the loop and this command will be repeated.

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is no loop in operation at this point

end_specification(*close_writer=True*)

Insert a command to indicate that the specification has finished and finish writing

Parameters **close_writer** (*bool*) – Indicates whether to close the underlying writer(s)

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

free_memory_region(*region*)

Insert command to free a previously reserved memory region

Parameters **region** (*int*) – The number of the region to free, from 0 to 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the region was not reserved

- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti**
  – If the region requested was out of the allowed range

function

get_structure_value(*destination_id*, *structure_id*, *parameter_index*, *parameter_index_is_register=False*)

Insert command to get a value from a structure. The value is copied in a register.

Parameters

- **destination_id** (*int*) – The id of the destination register
- **structure_id** (*int*) – The id of the source structure
- **parameter_index** (*int*) – The id of the parameter/element to copy
- **parameter_index_is_register** (*bool*) – True if the index of the structure is contained in a register

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti**
  –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If destination_id is not the id of a valid register
  - If parameter_index_is_register is True and parameter_index is not a valid register id
- **data_specification.exceptions. DataSpecificationNotAllocatedException**
  – If the structure requested has not been declared

instruction_counter

logical_and(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical AND operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15

- If operand_2_is_register is False, a 32-bit value

- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_left_shift (*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical left shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_not (*register_id*, *operand*, *operand_is_register=False*)

Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand** (*int*) –
  - If operand_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_is_register is False, a 32-bit value
- **operand_is_register** (*bool*) – Indicates if operand_1 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_is_register is True and operand_1 is not a valid register id

logical_or (*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical OR operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_right_shift(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical right shift operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

logical_xor(*register_id*, *operand_1*, *operand_2*, *operand_1_is_register=False*, *operand_2_is_register=False*)

Insert command to perform a logical xor operation, using the _call_logic_operation.

Parameters

- **register_id** (*int*) – The id of the register to store the result in
- **operand_1** (*int*) –
  - If operand_1_is_register is True, the id of a register where the first operand can be found, between 0 and 15
  - If operand_1_is_register is False, a 32-bit value
- **operand_2** (*int*) –
  - If operand_2_is_register is True, the id of a register where the second operand can be found. between 0 and 15
  - If operand_2_is_register is False, a 32-bit value
- **operand_1_is_register** (*bool*) – Indicates if operand_1 is a register id
- **operand_2_is_register** (*bool*) – Indicates if operand_2 is a register id

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If operand_1_is_register is True and operand_1 is not a valid register id
  - If operand_2_is_register is True and operand_2 is not a valid register id

mem_slot

no_operation()

Insert command to execute nothing

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

ongoing_function_definition

ongoing_loop

print_struct(*structure_id*, *structure_id_is_register=False*)

Insert command to print out a structure (for debugging)

Parameters

- **structure_id** (*int*) –
  - If structure_id_is_register is True, the id of the register containing the id of the structure to print, between 0 and 15
  - If structure_id_is_register is False, the id of the structure to print, between 0 and 15
- **structure_id_is_register** (*bool*) – Indicates if the structure_id is a register

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If structure_id_is_register is True and structure_id is not a valid register id
  - If structure_id_is_register is False and structure_id is not a valid structure id

- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If structure_id_is_register is False and structure_id is the id of a structure that has not been allocated

print_text (*text*)

Insert command to print some text (for debugging)

Parameters **text** (*str*) – The text to write (max 12 characters)

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails

print_value (*value*, *value_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to print out a value (for debugging)

Parameters

- **value** (*float*) –
  - If value_is_register is True, the id of the register containing the value to print
  - If value_is_register is False, the value to print as a float
- **value_is_register** (*bool*) – Indicates if the value is a register
- **data_type** (DataType) – The type of the data to be printed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If value_is_register is True and value is not a valid register id
  - If value_is_register is False, the data_type is an integer type and value has a fractional part
  - If value_is_register is False and the value would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** –
  - If data_type is not a valid data type

random_distribution

read_value (*dest_id*, *data_type*)

Insert command to read a value from the current write pointer, causing the write pointer to move by the number of bytes read. The data is stored in a register passed as argument.

Parameters

- **dest_id** (*int*) – The id of the destination register.
- **data_type** (DataType) – The type of the data to be read.

Returns Nothing is returned

Return type None

region_sizes

A list of sizes of each region that has been reserved. Note that the list will include 0s for each non-reserved region.

report_writer

reserve_memory_region (*region*, *size*, *label=None*, *empty=False*, *shrink=True*)

Insert command to reserve a memory region

Parameters

- **region** (*int*) – The number of the region to reserve, from 0 to 15
- **size** (*int*) – The size to reserve for the region in bytes
- **label** (*str*) – An optional label for the region
- **empty** (*bool*) – Specifies if the region will be left empty
- **shrink** (*bool*) – Specifies if the region will be compressed

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationRegionInUseException** – If the region was already reserved
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the region requested was out of the allowed range, or that the size was too big to fit in SDRAM

rng

save_write_pointer (*register_id*)

Insert command to save the write pointer to a register

Parameters **register_id** (*int*) – The id of the register to assign, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the register_id is not a valid register id
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected

set_register_value(*register_id*, *data*, *data_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to set the value of a register

Parameters

- **register_id** (*int*) – The id of the register to assign, between 0 and 15
- **data** (*float*) –
  - If data_is_register is True, the register id containing the data to set, between 0 and 15
  - If data_is_register is False, the data as a float
- **data_is_register** (*bool*) – Indicates if data is a register id
- **data_type** (*DataType*) – The type of the data to be assigned

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If register_id is not a valid register_id
  - If data_is_register is True, and data is not a valid register id
  - If data_is_register is False, data_type is an integer type and data has a fractional part
  - If data_is_register if False, and data would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known

set_structure_value(*structure_id*, *parameter_index*, *value*, *data_type*, *value_is_register=False*)

Insert command to set a value in a structure

Parameters

- **structure_id** (*int*) –
  - If called outside of a function, the id of the structure between 0 and 15
  - If called within a function, the id of the structure argument, between 0 and 4

- **parameter_index** (*int*) – The index of the value to assign in the structure, between 0 and the number of parameters declared in the structure
- **value** (*float*) –
  - If value_is_register is False, the value to assign at the selected position as a float
  - If value_is_register is True, the id of the register containing the value to assign to the position, between 0 and 15
- **data_type** (DataType) – type of the data to be stored in the structure. If parameter value_is_register is set to true, this variable is disregarded
- **value_is_register** (*bool*) – Identifies if value identifies a register

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If structure_id is not in the allowed range
  - If parameter_index is larger than the number of parameters declared in the original structure
  - If value_is_register is False, and the data type of the structure value is an integer, and the value has a fractional part
  - If value_is_register is False, and value would overflow the position in the structure
  - If value_is_register is True, and value is not a valid register id
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the structure requested has not been declared

set_write_pointer(*address*, *address_is_register=False*, *relative_to_current=False*)

Insert command to set the position of the write pointer within the current region

Parameters

- **address** (*int*) –
  - If address_is_register is True, the id of the register containing the address to move to
  - If address_is_register is False, the address to move the write pointer to
- **address_is_register** (*bool*) – Indicates if the the address is a register id
- **relative_to_current** (*bool*) – Indicates if the address is to be added to the current address, or used as an absolute address from the start of the current region

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the address_is_register is True and address is not a valid register id
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected

spec_writer

start_conditional (*register_id*, *condition*, *value*, *value_is_register=False*)

Insert command to start a conditional if...then...else construct. If the condition evaluates to true, the statement is executed up to the next else statement, or the end of the conditional, whichever comes first.

Parameters

- **register_id** (*int*) – The id of a register to compare the value of
- **condition** (Condition) – The condition which must be true to execute the following commands
- **value** (*int*) –
  - If value_is_register is False, the value to compare to the value in the register
  - If value_is_register is True, the id of the register containing the value to compare, between 0 and 15
- **value_is_register** (*bool*) – Indicates if value is a register id

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If the register_id is not a valid register id
  - if value_is_register is True and value is not a valid register id
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the condition is not a valid condition

start_function (*function_id*, *argument_by_value*)

Insert command to start a function definition, with up to 5 arguments, which are the ids of structures to be used within the function, each of which can be passed by reference or by value. In the commands following this command up to the end_function command, structures can only be referenced using the numbers 1 to 5 which address the arguments, rather than the original structure ids

Parameters

- **function_id** (*int*) – The id of the function currently defined.

- **argument_by_value** (*list of bool*) – A list of up to 5 booleans indicating if the structure to be passed as an argument is to be passed by reference (i.e. changes made within the function are persisted) or by value (i.e. changes made within the function are lost when the function exits. The number of arguments is determined by the length of this list.

Returns The id of the function, between 0 and 31

Return type int

Raises

- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If there are too many items in the list of arguments
- **data_specification.exceptions. DataSpecificationInvalidCommandException** – If there is already a function being defined at this point
- **data_specification.exceptions. DataSpecificationFunctionInUse** – If the function is already defined

start_loop(*counter_register_id*, *start*, *end*, *increment=1*, *start_is_register=False*, *end_is_register=False*, *increment_is_register=False*)

Insert command to start a loop

Parameters

- **counter_register_id** (*int*) – The id of the register to use as the loop counter, between 0 and 15
- **start** (*int*) –
  - If start_is_register is False, the number at which to start the loop counter, >= 0
  - If start_is_register is True, the id of the register containing the number at which to start the loop counter, between 0 and 15
- **end** (*int*) –
  - If end_is_register is False, the number which the loop counter must reach to stop the loop i.e. the loop will run while the contents of counter_register < end, >= 0
  - If end_is_register is True, the id of the register containing the number at which to stop the loop, between 0 and 15
- **increment** (*int*) –
  - If increment_is_register is False, the amount by which to increment the loop counter on each run of the loop, >= 0
  - If increment_is_register is True, the id of the register containing the amount by which to increment the loop counter on each run of the loop, between 0 and 15
- **start_is_register** (*bool*) – Indicates if start is a register id
- **end_is_register** (*bool*) – Indicates if end is a register id
- **increment_is_register** (*bool*) – Indicates if increment is a register id

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If counter_register_id is not a valid register id
  - If start_is_register is True and increment is not a valid register_id
  - If end_is_register is True and increment is not a valid register_id
  - If increment_is_register is True, and increment is not a valid register_id
  - If start_is_register is False and start is not in the allowed range
  - If end_is_register is False and end is not in the allowed range
  - If increment_is_register is False and increment is not in the allowed range

struct_slot

switch_write_focus (*region*)

Insert command to switch the region being written to

Parameters **region** (*int*) – The id of the region to switch to, between 0 and 15

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** – If the region identifier is not valid
- **data_specification.exceptions. DataSpecificationNotAllocatedException** – If the region has not been allocated
- **data_specification.exceptions. DataSpecificationRegionUnfilledException** – If the selected region should not be filled

txt_indent

write_array (*array_values*, *data_type=<DataType.UINT32: 2>*)

Insert command to write an array, causing the write pointer to move on by (data type size * the array size), in bytes.

Parameters

- **array_values** (*list of unsigned int*) – An array of words to be written
- **data_type** (data_specification.enums.DataType) – Type of data contained in the array

Returns The position of the write pointer within the current region, in bytes from the start of the region

Return type int

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been previously selected

write_cmd(*cmd_word_list*, *cmd_string*)

Writes cmd values created previously created and cacheed

see: create_cmd which is the ONLY way the parameters should have been created.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file

Returns Nothing is returned

Return type None

Raises

- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

write_command_to_files(*cmd_word_list*, *cmd_string*, *indent=False*, *outdent=False*, *no_instruction_number=False*)

Writes the binary command to the binary output file and, if the user has requested a text output for debug purposes, also write the text version to the text file. Setting the optional parameter 'indent' to True causes subsequent commands to be indented by two spaces relative to this one. Similarly, setting 'outdent' to True, reverses this spacing.

Parameters

- **cmd_word_list** (*bytearray*) – list of binary words to be added to the binary data specification file
- **cmd_string** (*str*) – string describing the command to be added to the report for the data specification file
- **indent** (*bool*) – if the following lines need to be indented
- **outdent** (*bool*) – if the following lines need to be out-dented
- **no_instruction_number** – if each report line should include also the address of the command in the file
- **no_instruction_number** – bool

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised

- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails

write_repeated_value(*data*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Parameters

- **data** (*float*) – the data to write as a float.
- **repeats** (*int*) –
  - If repeats_is_register is False, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
  - If repeats_is_register is True, this parameter identifies the register that contains the number of repeats.
- **repeats_is_register** (*bool*) – Indicates if the parameter repeats identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register id
  - If data_type is an integer type, and data has a fractional part
  - If data would overflow the data type
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationInvalidSizeException** – If the data size is invalid
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

write_structure(*structure_id*, *repeats=1*, *repeats_is_register=False*)

Insert command to write a structure to the current write pointer, causing the current write pointer to move on by the number of bytes needed to represent the structure

Parameters

- **structure_id** (*int*) –
  - If called within a function, the id of the structure to write, between 0 and 15

- If called outside of a function, the id of the structure argument, between 0 and 5

- **repeats** (*int*) –
  - If repeats_is_register is True, the id of the register containing the number of repeats, between 0 and 15
  - If repeats_is_register is False, the number of repeats to write, between 0 and 255
- **repeats_is_register** (*bool*) – Identifies if repeats identifies a register

Returns No value returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_is_register is False and structure_id is not a valid id
  - If repeats_is_register is True and structure_id
  - If the number of repeats is out of range
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to
- **data_specification.exceptions. DataSpecificationRegionExhaustedException** – If the selected region has no more space

write_value (*data*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times to the current write pointer, causing the write pointer to move on by the number of bytes required to represent the data type. The data is passed as a parameter to this function

Note: This method used to have two extra parameters repeats and repeats_is_register. They have been removed here. If you need then use write_repeated_value

Parameters

- **data** (*float*) – the data to write as a float.
- **data_type** (DataType) – the type to convert data to

Returns Nothing is returned

Return type None

Raises

- *data_specification.exceptions.DataUndefinedWriterException* – If the binary specification file writer has not been initialised
- *spinn_storage_handlers.exceptions.DataWriteException* – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –

- If repeats_register is None, and repeats is out of range
- If repeats_register is not a valid register id
- If data_type is an integer type, and data has a fractional part
- If data would overflow the data type

- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationInvalidSizeException** – If the data size is invalid
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

write_value_from_register(*data_register*, *repeats=1*, *repeats_is_register=False*, *data_type=<DataType.UINT32: 2>*)

Insert command to write a value one or more times at the write pointer of the current memory region, causing it to move. The data is contained in a register whose id is passed to the function

Parameters

- **data_register** (*int*) – Identifies the register in which the data is stored.
- **repeats** (*int*) –
  - If repeats_register is None, this parameter identifies the number of times to repeat the data, between 1 and 255 (default 1)
  - If repeats_register is not None (i.e. has an integer value), the content of this parameter is disregarded
- **repeats_is_register** (*None or int*) – Identifies the register containing the number of repeats of the value to write
- **data_type** (DataType) – the type of the data held in the register

Returns Nothing is returned

Return type None

Raises

- ***data_specification.exceptions.DataUndefinedWriterException*** – If the binary specification file writer has not been initialised
- ***spinn_storage_handlers.exceptions.DataWriteException*** – If a write to external storage fails
- **data_specification.exceptions. DataSpecificationParameterOutOfBoundsExcepti** –
  - If repeats_register is None, and repeats is out of range
  - If repeats_register is not a valid register id
  - If data_register is not a valid register id
- **data_specification.exceptions. DataSpecificationUnknownTypeException** – If the data type is not known
- **data_specification.exceptions. DataSpecificationNoRegionSelectedException** – If no region has been selected to write to

- **data_specification.exceptions. DataSpecificationRegionExhaustedException** – If the selected region has no more space

class data_specification.**MemoryRegion**(*memory_pointer*, *unfilled*, *size*)

Bases: object

Memory region storage object

Parameters

- **memory_pointer** – the write pointer position
- **unfilled** – if the region needs to be filled when written

Return type None

Raises **None** – this method does not raise any known exception

allocated_size

property method for the size of the region

Returns the size of the region

Return type int

Raises **None** – this method does not raise any known exception

increment_write_pointer(*n_bytes*)

max_write_pointer

memory_pointer

property method to retrieve memory write pointer

Returns the memory pointer of the region

Return type int

Raises **None** – this method does not raise any known exception

region_data

the container which holds the data written in this region

Returns the region data container

Return type bytearray

Raises **None** – this method does not raise any known exception

unfilled

property method to retrieve if the region is filled

Returns the the

Return type None

Raises **None** – this method does not raise any known exception

write_pointer

class data_specification.**MemoryRegionCollection**(*n_regions*)

Bases: object

Collection of memory regions.

Create a new MemoryRegionCollection with the given number of regions.

count_used_regions()

is_empty(*region*)

is_unfilled(*region*)

needs_to_write_region(*region_id*)

helper method which determines if a unfilled region actually needs to be written (optimisation to stop large data files)

Parameters **region_id** – the region id to which the test is being ran on

Returns a boolean stating if the region needs to be written

Return type boolean

Raises *DataSpecificationNoRegionSelectedException* – when the id is beyond the expected region range

regions

6.1.4 Indices and tables

- genindex
- modindex
- search

DataSpecification_github

DataSpecification_individual_docs

CHAPTER 7

spalloc

Spalloc is a Python library and set of command-line programs for requesting SpiNNaker machines from a spalloc server.

The spalloc module uses a different documentation style so is not included here.

Their documenation can be found at: spalloc_readthedocs

spalloc_github

CHAPTER 8

SpiNNFrontEndCommon

This package provides functionality which are common to front ends that translate application level programs into executables which run on a SpiNNaker machine.

8.1 SpiNNFrontEndCommon

8.1.1 Contents

spinn_front_end_common

spinn_front_end_common package

Subpackages

spinn_front_end_common.abstract_models package

Subpackages

spinn_front_end_common.abstract_models.impl package

Submodules

spinn_front_end_common.abstract_models.impl.machine_data_specable_vertex module

class spinn_front_end_common.abstract_models.impl.machine_data_specable_vertex.**MachineDataSpe**

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

generate_data_specification(*obj*, **args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

generate_machine_data_specification(*spec*, *placement*, *machine_graph*, *routing_info*, *iptags*, *reverse_iptags*, *machine_time_step*, *time_scale_factor*)

spinn_front_end_common.abstract_models.impl.needs_n_machine_time_steps module

class spinn_front_end_common.abstract_models.impl.needs_n_machine_time_steps.**NeedsNMachineTim**

Bases: object

A class that uses the number of machine time steps

set_n_machine_time_steps(*obj*, *arg*)

spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl module

class spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.**ProvidesK**
Bases: *spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping.AbstractProvidesKeyToAtomMapping*

routing_key_partition_atom_mapping(*obj*, **args*, ***kwargs*)
returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

Module contents

class spinn_front_end_common.abstract_models.impl.**MachineDataSpecableVertex**(**args*, ***kwargs*)
Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

generate_data_specification(*obj*, **args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

generate_machine_data_specification(*spec*, *placement*, *machine_graph*, *routing_info*, *iptags*, *reverse_iptags*, *machine_time_step*, *time_scale_factor*)

class spinn_front_end_common.abstract_models.impl.**NeedsNMachineTimeSteps**(**args*, ***kwargs*)

Bases: object

A class that uses the number of machine time steps

set_n_machine_time_steps(*obj*, *arg*)

class spinn_front_end_common.abstract_models.impl.**ProvidesKeyToAtomMappingImpl**

Bases: *spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping.AbstractProvidesKeyToAtomMapping*

routing_key_partition_atom_mapping(*obj*, **args*, ***kwargs*)

returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

Submodules

spinn_front_end_common.abstract_models.abstract_changable_after_run module

class spinn_front_end_common.abstract_models.abstract_changable_after_run.**AbstractChangableAf**

Bases: object

An item that can be changed after a call to run, the changes to which might or might not require mapping

mark_no_changes()

Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

requires_mapping

True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

spinn_front_end_common.abstract_models.abstract_generates_data_specification module

class spinn_front_end_common.abstract_models.abstract_generates_data_specification.**AbstractGe**

Bases: object

generate_data_specification(*spec*, *placement*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

spinn_front_end_common.abstract_models.abstract_has_associated_binary module

class spinn_front_end_common.abstract_models.abstract_has_associated_binary.**AbstractHasAsso**
Bases: object

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

spinn_front_end_common.abstract_models.abstract_machine_allocation_controller module

class spinn_front_end_common.abstract_models.abstract_machine_allocation_controller.**AbstractM**
Bases: object

An object that controls the allocation of a machine

close()
Indicate that the use of the machine is complete

extend_allocation(*new_total_run_time*)
Extend the allocation of the machine from the original run time

Parameters **new_total_run_time** – The total run time that is now required starting from when the machine was first allocated

spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraints module

class spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraints
Bases: object

A vertex that can provide constraints for its incoming edge partitions

get_incoming_partition_constraints(*partition*)
Get constraints to be added to the given edge that goes in to a vertex of this vertex

Parameters **partition** (pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition) – An partition that goes in to this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping module

class spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping.**AbstractPr**
Bases: object

interface to provide a mapping between routing key partitions and atom ids

routing_key_partition_atom_mapping(*routing_info*, *partition*)
returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition module

class spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition.**AbstractP**
Bases: object

Allows a vertex to provide the number of keys for a partition of edges, rather than relying on the number of atoms in the pre-vertex

get_n_keys_for_partition(*partition*, *graph_mapper*)
Get the number of keys required by the given partition of edges

Parameters

- **partition** (pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition) – An partition that comes out of this vertex
- **graph_mapper** (pacman.model.graph.graph_mapper.GraphMapper) – A mapper between the graphs

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints module

class spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints
Bases: object

A vertex that can provide constraints for its outgoing edge partitions

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

spinn_front_end_common.abstract_models.abstract_recordable module

class spinn_front_end_common.abstract_models.abstract_recordable.**AbstractRecordable**
Bases: object

Indicates that an object might record some data in to SDRAM

is_recording()
Deduce if the recorder is actually recording

spinn_front_end_common.abstract_models.abstract_rewrites_data_specification module

class spinn_front_end_common.abstract_models.abstract_rewrites_data_specification.**AbstractRew**
Bases: object

Indicates an object that allows data to be changed after run, and so can rewrite the data specification

mark_regions_reloaded()
Indicate that the regions have been reloaded

regenerate_data_specification(*spec*, *placement*)
Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_memory_regions_to_be_reloaded()
Return true if any data region needs to be reloaded

Return type bool

spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex module

class spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.**Abstr**
Bases: object

A vertex which wants to commands to be sent to it as multicast packets at fixed points in the simulation

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

spinn_front_end_common.abstract_models.abstract_supports_database_injection module

class spinn_front_end_common.abstract_models.abstract_supports_database_injection.**AbstractSup**
Bases: object

Marks a machine vertex as supporting injection of information via a database running on the controlling host.

is_in_injection_mode
Whether this vertex is actually in injection mode.

spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices module

class spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.**Abstract**
Bases: object

A vertex with a dependent vertices, which should be connected to this vertex by an edge directly to each of them

dependent_vertices()
Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)
Return the dependent edge identifiers for this vertex

Module contents

class spinn_front_end_common.abstract_models.**AbstractChangableAfterRun**
Bases: object

An item that can be changed after a call to run, the changes to which might or might not require mapping

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

requires_mapping
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

class spinn_front_end_common.abstract_models.**AbstractGeneratesDataSpecification**
Bases: object

generate_data_specification(*spec*, *placement*)
Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

class spinn_front_end_common.abstract_models.**AbstractHasAssociatedBinary**
Bases: object

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

class spinn_front_end_common.abstract_models.**AbstractMachineAllocationController**
Bases: object

An object that controls the allocation of a machine

close()
Indicate that the use of the machine is complete

extend_allocation(*new_total_run_time*)
Extend the allocation of the machine from the original run time

Parameters **new_total_run_time** – The total run time that is now required starting from when the machine was first allocated

class spinn_front_end_common.abstract_models.**AbstractProvidesIncomingPartitionConstraints**
Bases: object

A vertex that can provide constraints for its incoming edge partitions

get_incoming_partition_constraints(*partition*)
Get constraints to be added to the given edge that goes in to a vertex of this vertex

Parameters **partition** (pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition) – An partition that goes in to this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

class spinn_front_end_common.abstract_models.**AbstractProvidesKeyToAtomMapping**
Bases: object

interface to provide a mapping between routing key partitions and atom ids

routing_key_partition_atom_mapping(*routing_info*, *partition*)
returns a list of atom to key mapping.

Parameters

- **routing_info** – the routing info object to consider
- **partition** – the routing partition to handle.

Returns a iterable of tuples of atom ids to keys.

class spinn_front_end_common.abstract_models.**AbstractProvidesNKeysForPartition**
Bases: object

Allows a vertex to provide the number of keys for a partition of edges, rather than relying on the number of atoms in the pre-vertex

get_n_keys_for_partition(*partition*, *graph_mapper*)
Get the number of keys required by the given partition of edges

Parameters

- **partition** (pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition) – An partition that comes out of this vertex
- **graph_mapper** (pacman.model.graph.graph_mapper.GraphMapper) – A mapper between the graphs

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

class spinn_front_end_common.abstract_models.**AbstractProvidesOutgoingPartitionConstraints**
Bases: object

A vertex that can provide constraints for its outgoing edge partitions

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

class spinn_front_end_common.abstract_models.**AbstractRecordable**
Bases: object

Indicates that an object might record some data in to SDRAM

is_recording()
Deduce if the recorder is actually recording

class spinn_front_end_common.abstract_models.**AbstractRewritesDataSpecification**
Bases: object

Indicates an object that allows data to be changed after run, and so can rewrite the data specification

mark_regions_reloaded()
Indicate that the regions have been reloaded

regenerate_data_specification(*spec*, *placement*)
Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_memory_regions_to_be_reloaded()
Return true if any data region needs to be reloaded

Return type bool

class spinn_front_end_common.abstract_models.**AbstractSendMeMulticastCommandsVertex**
Bases: object

A vertex which wants to commands to be sent to it as multicast packets at fixed points in the simulation

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spinn_front_end_common.abstract_models.**AbstractSupportsDatabaseInjection**
Bases: object

Marks a machine vertex as supporting injection of information via a database running on the controlling host.

is_in_injection_mode
Whether this vertex is actually in injection mode.

class spinn_front_end_common.abstract_models.**AbstractVertexWithEdgeToDependentVertices**
Bases: object

A vertex with a dependent vertices, which should be connected to this vertex by an edge directly to each of them

dependent_vertices()
Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)
Return the dependent edge identifiers for this vertex

spinn_front_end_common.common_model_binaries package

Module contents

This module contains no python code

spinn_front_end_common.interface package

Subpackages

spinn_front_end_common.interface.buffer_management package

Subpackages

spinn_front_end_common.interface.buffer_management.buffer_models package

Submodules

spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host module

class spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffe
Bases: object

Indicates that this object can receive buffers

get_minimum_buffer_sdram_usage()
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_region_ids()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers_from_host module

class spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers
Bases: object

Interface to an object that sends buffers of keys to be transmitted at given timestamps in the simulation

buffering_input()
Return True if the input of this vertex is to be buffered

get_max_buffer_size_possible(*region*)
Gets the max possible size of a buffered region

Parameters **region** (*int*) – the region to find the max possible size of

Returns the max possible size of the buffered region

Return type int

get_next_key(*region*)
Get the next key in the given region

Parameters **region** (*int*) – The region to get the next key from

Returns The next key, or None if there are no more keys

Return type int

get_next_timestamp(*region*)
Get the next timestamp at which there are still keys to be sent for the given region

Parameters **region** (*int*) – The region to get the timestamp for

Returns The timestamp of the next available keys

Return type int

get_region_buffer_size(*region*)
Get the size of the buffer to be used in SDRAM on the machine for the region in bytes

Parameters **region** (*int*) – The region to get the buffer size of

Returns The size of the buffer space in bytes

Return type int

get_regions()
Get the set of regions for which there are keys to be sent

Returns Iterable of region ids

Return type iterable of int

is_empty(*region*)
Return true if there are no spikes to be buffered for the specified region

Parameters **region** (*int*) – The region to get the next key from

Returns True if there are no keys to send for the region, False otherwise

Return type bool

is_next_key(*region*, *timestamp*)
Determine if there are still keys to be sent at the given timestamp for the given region

Parameters

- **region** (*int*) – The region to determine if there are keys for
- **timestamp** (*int*) – The timestamp to determine if there are more keys for

Returns True if there are more keys to send for the parameters, False otherwise

Return type bool

is_next_timestamp(*region*)
Determine if there is another timestamp with data to be sent

Parameters **region** (*int*) – The region to determine if there is more data for

Returns True if there is more data, False otherwise

Return type int

rewind(*region*)

Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (*int*) – The region to rewind

spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_host_pre_buffered_in module

class spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_hos
Bases: *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers_from_host.AbstractSendsBuffersFromHost*

Implementation of the AbstractSendsBuffersFromHost which uses an existing set of buffers for the details

buffering_input()

get_max_buffer_size_possible(*region*)
Return the max possible size of a buffered region

Parameters **region** – the region to find the max possible size of

Returns the max possible size of the buffered region

get_next_key(*region*)
Get the next key for a given region

Parameters **region** – the region to get the next key from

get_next_timestamp(*region*)
Return the next time stamp available in the buffered region

Parameters **region** – the region id which is being asked

Returns the next time stamp

get_region_buffer_size(*region*)
Return the size of a given regions buffer

Parameters **region** – the region to find the size of

Returns the size of the buffer

get_regions()
Return the regions which has buffers to send

is_empty(*region*)
Check if a region is empty

Parameters **region** – the region id to check

Returns bool

is_next_key(*region*, *timestamp*)
Check if there is more keys to transmit for a given region in a given timestamp

Parameters

- **region** – the region id to check
- **timestamp** – the timestamp to check

Returns bool

is_next_timestamp(*region*)
Check if there are more time stamps which need transmitting

Parameters **region** – the region to check

Returns boolean

rewind(*region*)
Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (*int*) – The region to rewind

send_buffers

Module contents

class spinn_front_end_common.interface.buffer_management.buffer_models.**AbstractReceiveBuffers**
Bases: object

Indicates that this object can receive buffers

get_minimum_buffer_sdram_usage()
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_region_ids()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

class spinn_front_end_common.interface.buffer_management.buffer_models.**AbstractSendsBuffersFr**
Bases: object

Interface to an object that sends buffers of keys to be transmitted at given timestamps in the simulation

buffering_input()
Return True if the input of this vertex is to be buffered

get_max_buffer_size_possible(*region*)
Gets the max possible size of a buffered region

Parameters **region** (*int*) – the region to find the max possible size of

Returns the max possible size of the buffered region

Return type int

get_next_key(*region*)
Get the next key in the given region

Parameters **region** (*int*) – The region to get the next key from

Returns The next key, or None if there are no more keys

Return type int

get_next_timestamp(*region*)
Get the next timestamp at which there are still keys to be sent for the given region

Parameters **region** (*int*) – The region to get the timestamp for

Returns The timestamp of the next available keys

Return type int

get_region_buffer_size(*region*)
Get the size of the buffer to be used in SDRAM on the machine for the region in bytes

Parameters **region** (*int*) – The region to get the buffer size of

Returns The size of the buffer space in bytes

Return type int

get_regions()
Get the set of regions for which there are keys to be sent

Returns Iterable of region ids

Return type iterable of int

is_empty(*region*)
Return true if there are no spikes to be buffered for the specified region

Parameters **region** (*int*) – The region to get the next key from

Returns True if there are no keys to send for the region, False otherwise

Return type bool

is_next_key(*region*, *timestamp*)
Determine if there are still keys to be sent at the given timestamp for the given region

Parameters

- **region** (*int*) – The region to determine if there are keys for
- **timestamp** (*int*) – The timestamp to determine if there are more keys for

Returns True if there are more keys to send for the parameters, False otherwise

Return type bool

is_next_timestamp(*region*)
Determine if there is another timestamp with data to be sent

Parameters **region** (*int*) – The region to determine if there is more data for

Returns True if there is more data, False otherwise

Return type int

rewind(*region*)

Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (*int*) – The region to rewind

class spinn_front_end_common.interface.buffer_management.buffer_models.**SendsBuffersFromHostPr**
Bases: *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers_from_host.AbstractSendsBuffersFromHost*

Implementation of the AbstractSendsBuffersFromHost which uses an existing set of buffers for the details

buffering_input()

get_max_buffer_size_possible(*region*)
Return the max possible size of a buffered region

Parameters **region** – the region to find the max possible size of

Returns the max possible size of the buffered region

get_next_key(*region*)
Get the next key for a given region

Parameters **region** – the region to get the next key from

get_next_timestamp(*region*)
Return the next time stamp available in the buffered region

Parameters **region** – the region id which is being asked

Returns the next time stamp

get_region_buffer_size(*region*)
Return the size of a given regions buffer

Parameters **region** – the region to find the size of

Returns the size of the buffer

get_regions()
Return the regions which has buffers to send

is_empty(*region*)
Check if a region is empty

Parameters **region** – the region id to check

Returns bool

is_next_key(*region*, *timestamp*)
Check if there is more keys to transmit for a given region in a given timestamp

Parameters

- **region** – the region id to check
- **timestamp** – the timestamp to check

Returns bool

is_next_timestamp(*region*)
Check if there are more time stamps which need transmitting

Parameters **region** – the region to check

Returns boolean

rewind(*region*)
Rewinds the internal buffer in preparation of re-sending the spikes

Parameters **region** (*int*) – The region to rewind

send_buffers

spinn_front_end_common.interface.buffer_management.storage_objects package

Submodules

spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data module

class spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_d
Bases: object

Stores the information received through the buffering output technique from the SpiNNaker system. The data kept includes the last sent packet and last received packet, their correspondent sequence numbers, the data retrieved, a flag to identify if the data from a core has been flushed and the final state of the buffering output state machine

Parameters **store_to_file** (*bool*) – A boolean to identify if the data will be stored in memory using a byte array or in a temporary file on the disk

clear(*x*, *y*, *p*, *region_id*)
clears the data from a given data region (only clears things associated with a given data recording region).

Parameters

- **x** – placement x coord

- **y** – placement y coord
- **p** – placement p coord
- **region_id** – the recording region id to clear data from

Return type None

flushing_data_from_region (*x*, *y*, *p*, *region*, *data*)

Store flushed data from a region of a core on a chip, and mark it as being flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

get_end_buffering_sequence_number (*x*, *y*, *p*)

Get the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The last sequence number

Return type int

get_end_buffering_state (*x*, *y*, *p*, *region*)

Get the end state of the buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The end state

get_region_data (*x*, *y*, *p*, *region*)

Get the data stored for a given region of a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns an array contained all the data received during the, simulation, and a flag indicating if any data was missing

Return type (bytearray, bool)

get_region_data_pointer(*x*, *y*, *p*, *region*)
Get the data received during the simulation for a region of a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns all the data received during the simulation, and a flag indicating if any data was lost

Return type (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage, bool)

is_data_from_region_flushed(*x*, *y*, *p*, *region*)
Check if the data region has been flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns True if the region has been flushed. False otherwise

Return type bool

is_end_buffering_sequence_number_stored(*x*, *y*, *p*)
Determine if the last sequence number has been retrieved

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the number has been retrieved

Return type bool

is_end_buffering_state_recovered(*x*, *y*, *p*, *region*)
Determine if the end state has been stored

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the state has been stored

last_received_packet_from_core(*x*, *y*, *p*)
Get the last packet received for a given core

Parameters

- **x** (*int*) – x coordinate of the chip

- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns SpinnakerRequestReadData packet received

Return type *spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData*

last_sent_packet_to_core(*x*, *y*, *p*)

Retrieve the last packet sent to a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns last HostDataRead packet sent

Return type *spinnman.messages.eieio.command_messages.host_data_read.HostDataRead*

last_sequence_no_for_core(*x*, *y*, *p*)

Get the last sequence number for a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns last sequence number used

Return type int

resume()

Resets states so that it can behave in a resumed mode

store_data_in_region_buffer(*x*, *y*, *p*, *region*, *data*)

Store some information in the correspondent buffer class for a specific chip, core and region

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

store_end_buffering_sequence_number(*x*, *y*, *p*, *sequence*)

Store the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

- **sequence** (*int*) – The last sequence number

store_end_buffering_state (*x*, *y*, *p*, *region*, *state*)
Store the end state of buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **state** – The end state

store_last_received_packet_from_core (*x*, *y*, *p*, *packet*)
Store the most recent packet received from SpiNNaker for a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (*spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData*) – SpinnakerRequestReadData packet received

store_last_sent_packet_to_core (*x*, *y*, *p*, *packet*)
Store the last packet sent to the given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (*spinnman.messages.eieio.command_messages.host_data_read.HostDataRead*) – last HostDataRead packet sent

update_sequence_no_for_core (*x*, *y*, *p*, *sequence_no*)
Set the last sequence number used

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **sequence_no** (*int*) – last sequence number used

Return type None

spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region module

class spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_reg
Bases: object

A set of keys to be sent at given timestamps for a given region of data. Note that keys must be added in timestamp order or else an exception will be raised

add_key (*timestamp*, *key*)
Add a key to be sent at a given time

Parameters

- **timestamp** (*int*) – The time at which the key is to be sent
- **key** (*int*) – The key to send

add_keys (*timestamp*, *keys*)
Add a set of keys to be sent at the given time

Parameters

- **timestamp** (*int*) – The time at which the keys are to be sent
- **keys** (*iterable of int*) – The keys to send

buffer_size
property method for getting the max size of this buffer

clear ()
Clears the buffer

current_timestamp
Get the current timestamp in the iterator

get_n_bytes (*n_keys*)
Get the number of bytes used by a given number of keys

Parameters **n_keys** (*int*) – The number of keys

get_n_keys (*timestamp*)
Get the number of keys for a given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

is_next_key (*timestamp*)
Determine if there is another key for the given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

Return type bool

is_next_timestamp
Determines if the region is empty

Returns True if the region is empty, false otherwise

Return type bool

max_buffer_size_possible
Get the max possible size of a buffer from this region

max_packets_in_timestamp
The maximum number of packets in any time stamp

n_timestamps
The number of timestamps available

Return type int

next_key
The next key to be sent

Return type int

next_timestamp
The next timestamp of the data to be sent, or None if no more data

Return type int or None

rewind()
Rewind the buffer to initial position.

timestamps
The timestamps for which there are keys

Return type iterable of int

total_region_size
Get the max size of this region

spinn_front_end_common.interface.buffer_management.storage_objects.buffers_sent_deque module

class spinn_front_end_common.interface.buffer_management.storage_objects.buffers_sent_deque.B

Bases: object

A tracker of buffers sent / to send for a region

Parameters

- **region** (*int*) – The region being managed
- **sent_stop_message** (*bool*) – True if the stop message has been sent
- **n_sequences_per_tranmission** (*int*) – The number of sequences allowed in each transmission set

add_message_to_send(*message*)
Add a message to send. The message is converted to a sequenced message.

Parameters **message** (*spinnman.messages.eieio.abstract_messages.AbstractEIEIOMessage*) – The message to be added

is_empty()
Determine if there are no messages

Return type int

is_full
Determine if the number of messages sent is at the limit for the sequencing system

Return type bool

messages
The messages that have been added to the set

Return type iterable of *spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData*

send_stop_message()
Send a message to indicate the end of all the messages

update_last_received_sequence_number(*last_received_sequence_no*)

Updates the last received sequence number. If the sequence number is within the valid window, packets before the sequence number within the window are removed, and the last received sequence number is updated, thus moving the window for the next call. If the sequence number is not within the valid window, it is assumed to be invalid and so is ignored.

Parameters **last_received_sequence_no** (*int*) – The new sequence number

Returns True if update went ahead, False if it was ignored

Return type bool

spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state module

class spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state

Bases: object

Stores information related to a single channel output buffering state, as it is retrieved at the end of a simulation on the SpiNNaker system.

Parameters

- **start_address** – start buffering area memory address (32 bits)
- **current_write** – address where data was last written (32 bits)
- **current_read** – address where data was last read (32 bits)
- **end_address** – The address of first byte after the buffer (32 bits)
- **region_id** – The id of the region (8 bits)
- **missing_info** – True if the region overflowed during the simulation (8 bits)
- **last_buffer_operation** – Last operation performed on the buffer - read or write (8 bits)

ChannelBufferStateSize = 24

static create_from_bytearray(*data*)

current_read

current_write

end_address

is_state_updated

last_buffer_operation

missing_info

region_id

set_update_completed()

static **size_of_channel_state**()

start_address

update_last_operation(*operation*)

update_read_pointer(*read_ptr*)

spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state module

class spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state.

Bases: object

Stores the buffering state at the end of a simulation

Parameters

- **buffering_out_fsm_state** – Final sequence number received
- **list_channel_buffer_state** – a list of channel state, where each channel is stored in a ChannelBufferState object

buffering_out_fsm_state

channel_buffer_state(*i*)

get_missing_info_for_region(*region_id*)

get_state_for_region(*region_id*)

static **size_of_region**(*n_regions_to_record*)

Module contents

class spinn_front_end_common.interface.buffer_management.storage_objects.**BufferedReceivingDat**

Bases: object

Stores the information received through the buffering output technique from the SpiNNaker system. The data kept includes the last sent packet and last received packet, their correspondent sequence numbers, the data retrieved, a flag to identify if the data from a core has been flushed and the final state of the buffering output state machine

Parameters **store_to_file**(*bool*) – A boolean to identify if the data will be stored in memory using a byte array or in a temporary file on the disk

clear(*x*, *y*, *p*, *region_id*)

clears the data from a given data region (only clears things associated with a given data recording region).

Parameters

- **x** – placement x coord
- **y** – placement y coord
- **p** – placement p coord

- **region_id** – the recording region id to clear data from

Return type None

flushing_data_from_region (*x*, *y*, *p*, *region*, *data*)
Store flushed data from a region of a core on a chip, and mark it as being flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

get_end_buffering_sequence_number (*x*, *y*, *p*)
Get the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The last sequence number

Return type int

get_end_buffering_state (*x*, *y*, *p*, *region*)
Get the end state of the buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns The end state

get_region_data (*x*, *y*, *p*, *region*)
Get the data stored for a given region of a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns an array contained all the data received during the, simulation, and a flag indicating if any data was missing

Return type (bytearray, bool)

get_region_data_pointer (*x*, *y*, *p*, *region*)
Get the data received during the simulation for a region of a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns all the data received during the simulation, and a flag indicating if any data was lost

Return type (`spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage`, bool)

is_data_from_region_flushed(*x*, *y*, *p*, *region*)

Check if the data region has been flushed

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data

Returns True if the region has been flushed. False otherwise

Return type bool

is_end_buffering_sequence_number_stored(*x*, *y*, *p*)

Determine if the last sequence number has been retrieved

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the number has been retrieved

Return type bool

is_end_buffering_state_recovered(*x*, *y*, *p*, *region*)

Determine if the end state has been stored

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns True if the state has been stored

last_received_packet_from_core(*x*, *y*, *p*)

Get the last packet received for a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns SpinnakerRequestReadData packet received

Return type *spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData*

last_sent_packet_to_core(*x*, *y*, *p*)
Retrieve the last packet sent to a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns last HostDataRead packet sent

Return type *spinnman.messages.eieio.command_messages.host_data_read.HostDataRead*

last_sequence_no_for_core(*x*, *y*, *p*)
Get the last sequence number for a core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip

Returns last sequence number used

Return type int

resume()
Resets states so that it can behave in a resumed mode

store_data_in_region_buffer(*x*, *y*, *p*, *region*, *data*)
Store some information in the correspondent buffer class for a specific chip, core and region

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **region** (*int*) – Region containing the data to be stored
- **data** (*bytearray*) – data to be stored

store_end_buffering_sequence_number(*x*, *y*, *p*, *sequence*)
Store the last sequence number sent by the core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **sequence** (*int*) – The last sequence number

store_end_buffering_state(*x*, *y*, *p*, *region*, *state*)
Store the end state of buffering

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **state** – The end state

store_last_received_packet_from_core (*x*, *y*, *p*, *packet*)
Store the most recent packet received from SpiNNaker for a given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (*spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData*) – SpinnakerRequestReadData packet received

store_last_sent_packet_to_core (*x*, *y*, *p*, *packet*)
Store the last packet sent to the given core

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **packet** (*spinnman.messages.eieio.command_messages.host_data_read.HostDataRead*) – last HostDataRead packet sent

update_sequence_no_for_core (*x*, *y*, *p*, *sequence_no*)
Set the last sequence number used

Parameters

- **x** (*int*) – x coordinate of the chip
- **y** (*int*) – y coordinate of the chip
- **p** (*int*) – Core within the specified chip
- **sequence_no** (*int*) – last sequence number used

Return type None

class spinn_front_end_common.interface.buffer_management.storage_objects.**BufferedSendingRegio**
Bases: object

A set of keys to be sent at given timestamps for a given region of data. Note that keys must be added in timestamp order or else an exception will be raised

add_key (*timestamp*, *key*)
Add a key to be sent at a given time

Parameters

- **timestamp** (*int*) – The time at which the key is to be sent
- **key** (*int*) – The key to send

add_keys(*timestamp*, *keys*)
Add a set of keys to be sent at the given time

Parameters

- **timestamp** (*int*) – The time at which the keys are to be sent
- **keys** (*iterable of int*) – The keys to send

buffer_size
property method for getting the max size of this buffer

clear()
Clears the buffer

current_timestamp
Get the current timestamp in the iterator

get_n_bytes(*n_keys*)
Get the number of bytes used by a given number of keys

Parameters **n_keys** (*int*) – The number of keys

get_n_keys(*timestamp*)
Get the number of keys for a given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

is_next_key(*timestamp*)
Determine if there is another key for the given timestamp

Parameters **timestamp** – the time stamp to check if there's still keys to transmit

Return type bool

is_next_timestamp
Determines if the region is empty

Returns True if the region is empty, false otherwise

Return type bool

max_buffer_size_possible
Get the max possible size of a buffer from this region

max_packets_in_timestamp
The maximum number of packets in any time stamp

n_timestamps
The number of timestamps available

Return type int

next_key
The next key to be sent

Return type int

next_timestamp
The next timestamp of the data to be sent, or None if no more data

Return type int or None

rewind()
Rewind the buffer to initial position.

timestamps
The timestamps for which there are keys

Return type iterable of int

total_region_size
Get the max size of this region

class spinn_front_end_common.interface.buffer_management.storage_objects.**BuffersSentDeque**(*regi*
sent
n_se

Bases: object

A tracker of buffers sent / to send for a region

Parameters

- **region** (*int*) – The region being managed
- **sent_stop_message** (*bool*) – True if the stop message has been sent
- **n_sequences_per_tranmission** (*int*) – The number of sequences allowed in each transmission set

add_message_to_send(*message*)
Add a message to send. The message is converted to a sequenced message.

Parameters **message** (*spinnman.messages.eieio.abstract_messages.AbstractEIEIOMessage*) – The message to be added

is_empty()
Determine if there are no messages

Return type int

is_full
Determine if the number of messages sent is at the limit for the sequencing system

Return type bool

messages
The messages that have been added to the set

Return type iterable of *spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData*

send_stop_message()
Send a message to indicate the end of all the messages

update_last_received_sequence_number(*last_received_sequence_no*)
Updates the last received sequence number. If the sequence number is within the valid window, packets before the sequence number within the window are removed, and the last received sequence number is updated, thus moving the window for the next call. If the sequence number is not within the valid window, it is assumed to be invalid and so is ignored.

Parameters **last_received_sequence_no** (*int*) – The new sequence number

Returns True if update went ahead, False if it was ignored

Return type bool

class spinn_front_end_common.interface.buffer_management.storage_objects.**ChannelBufferState**(*s*
c
r
c
r
c
r
e
r
g
n
i
l

Bases: object

Stores information related to a single channel output buffering state, as it is retrieved at the end of a simulation on the SpiNNaker system.

Parameters

- **start_address** – start buffering area memory address (32 bits)
- **current_write** – address where data was last written (32 bits)
- **current_read** – address where data was last read (32 bits)
- **end_address** – The address of first byte after the buffer (32 bits)
- **region_id** – The id of the region (8 bits)
- **missing_info** – True if the region overflowed during the simulation (8 bits)
- **last_buffer_operation** – Last operation performed on the buffer - read or write (8 bits)

ChannelBufferStateSize = 24

static create_from_bytearray(*data*)

current_read

current_write

end_address

is_state_updated

last_buffer_operation

missing_info

region_id

set_update_completed()

static size_of_channel_state()

start_address

update_last_operation(*operation*)

update_read_pointer(*read_ptr*)

class spinn_front_end_common.interface.buffer_management.storage_objects.**EndBufferingState**(*bu*
lis

Bases: object

Stores the buffering state at the end of a simulation

Parameters

- **buffering_out_fsm_state** – Final sequence number received
- **list_channel_buffer_state** – a list of channel state, where each channel is stored in a ChannelBufferState object

buffering_out_fsm_state

channel_buffer_state(*i*)

get_missing_info_for_region(*region_id*)

get_state_for_region(*region_id*)

static size_of_region(*n_regions_to_record*)

Submodules

spinn_front_end_common.interface.buffer_management.buffer_manager module

class spinn_front_end_common.interface.buffer_management.buffer_manager.**BufferManager**(*placement*
tags,
transceive
store_to_f

Bases: object

Manager of send buffers

Parameters

- **placements** (*pacman.model.placements.Placements*) – The placements of the vertices
- **tags** (*pacman.model.tags.Tags*) – The tags assigned to the vertices
- **transceiver** (*spinnman.transceiver.Transceiver*) – The transceiver to use for sending and receiving information
- **store_to_file** (*bool*) – True if the data should be temporarily stored in a file instead of in RAM (default uses RAM)

add_receiving_vertex(*vertex*)

Add a vertex into the managed list for vertices which require buffers to be received from them during runtime

add_sender_vertex(*vertex*)

Add a vertex into the managed list for vertices which require buffers to be sent to them during runtime

Parameters **vertex** (spinnaker.pyNN.models.abstract_models.buffer_models.AbstractSendsBuffersFromHost) – the vertex to be managed

clear_recorded_data(*x*, *y*, *p*, *recording_region_id*)

Removes the recorded data stored in memory.

Parameters

- **x** – placement x coord
- **y** – placement y coord

- **p** – placement p coord
- **recording_region_id** – the recording region id

get_data_for_vertex(*placement*, *recording_region_id*)

Get a pointer to the data container for all the data retrieved during the simulation from a specific region area of a core

Parameters

- **placement** (pacman.model.placements.Placement) – the placement to get the data from
- **recording_region_id** (*int*) – desired recording data region

Returns pointer to a class which inherits from AbstractBufferedDataStorage

Return type spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage

load_initial_buffers()

Load the initial buffers for the senders using mem writes

receive_buffer_command_message(*packet*)

Handle an EIEIO command message for the buffers

Parameters **packet** (*spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*) – The eieio message received

reload_buffer_files

The file paths for each buffered region for each sender vertex

reset()

Resets the buffered regions to start transmitting from the beginning of its expected regions and clears the buffered out data files

resume()

Resets any data structures needed before starting running again

sender_vertices

The vertices which are buffered

stop()

Indicates that the simulation has finished, so no further outstanding requests need to be processed

spinn_front_end_common.interface.buffer_management.recording_utilities module

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_last_sequence_nu**

Read the last sequence number from the data

Parameters

- **placement** – The placement from which to read the sequence number
- **transceiver** – The transceiver to use to read the sequence number
- **recording_data_address** – The address of the recording data from which to read the number

Return type int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_minimum_buffer_s**

Get the minimum buffer SDRAM

Parameters

- **buffered_sdram_per_timestep** (*list of int*) – The maximum number of bytes to use per timestep of recording, per recorded region. Disabled regions can specify 0.
- **n_machine_time_steps** (*int*) – The number of machine time steps for the simulation. Can be None if use_auto_pause_and_resume is True
- **minimum_sdram_for_buffering** (*int*) – The minimum SDRAM to reserve per recorded region for buffering

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_n_timesteps_in_b**

Get the number of time steps of data that can be stored in a given buffers space

Parameters

- **buffer_space** (*int*) – The space that will hold the data
- **buffered_sdram_per_timestep** (*list of int*) – The maximum SDRAM used by each region per timestep

Return type int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recorded_region_**

Get the ids of regions where recording is enabled

Parameters **buffered_sdram_per_timestep** (*list of int*) – The maximum SDRAM used by each region per timestep, where 0 indicates a disabled region

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recorded_region_**

Get the size of each recording region to be passed in to get_recording_header_array

Parameters

- **n_machine_time_steps** (*int*) – The duration of the simulation segment in time steps
- **buffered_sdram_per_timestep** (*list of int*) – The maximum SDRAM used per timestep in bytes per region
- **maximum_sdram_for_buffering** (*None or list of int*) – The maximum size of each buffer, or None if no maximum

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_data_s**

Get the size of the recorded data to be reserved

Parameters **recorded_region_sizes** – A list of sizes of each region to be recorded. A size of 0 is acceptable.

Return type int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_header**

Get data to be written for the recording header

Parameters

- **recorded_region_sizes** – A list of sizes of each region to be recorded. A size of 0 is acceptable.
- **time_between_triggers** – The minimum time between requesting reads of any region
- **buffer_size_before_request** – The amount of buffer to fill before a read request is sent
- **ip_tags** – A list of ip tags to extract the buffer tag from
- **buffering_tag** – The tag to use for buffering requests

Returns An array of values to be written as the header

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_header**

Get the size of the data to be written for the recording header

Parameters **n_recorded_regions** – The number of regions to be recorded

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_region**

Get the size of each recording region to be passed in to get_recording_resources, based on the details of the simulation

Parameters

- **buffered_sdram_per_timestep** (*list of int*) – The maximum number of bytes to use per timestep of recording, per recorded region. Disabled regions can specify 0.
- **n_machine_time_steps** (*int*) – The number of machine time steps for the simulation. Can be None if use_auto_pause_and_resume is True
- **minimum_sdram_for_buffering** (*int*) – The minimum SDRAM to reserve per recorded region for buffering
- **maximum_sdram_for_buffering** (*None or list of int*) – The maximum size of each buffer, or None if no maximum

- **use_auto_pause_and_resume** (*bool*) – True if automatic pause and resume is to be used for buffering

Return type list of int

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_recording_resour**

Get the resources for recording

Parameters

- **region_sizes** (*list of int*) – A list of the sizes of each region. A size of 0 is acceptable to indicate an empty region
- **buffering_ip_address** (*str*) – The ip address to receive buffering messages on, or None if buffering is not in use
- **buffering_port** (*int*) – The port to receive buffering messages on, or None if a port is to be assigned
- **notification_tag** (*int*) – The tag to send buffering messages with, or None to use a default tag

Return type *pacman.model.resources.ResourceContainer*

spinn_front_end_common.interface.buffer_management.recording_utilities.**get_region_pointer**(*p*
t
r
i
r
&

Get a pointer to a recording region

Parameters

- **placement** – The placement from which to read the pointer
- **transceiver** – The transceiver to use to read the pointer
- **recording_data_address** – The address of the recording data from which to read the pointer
- **region** – The index of the region to get the pointer of

Return type int

Module contents

class spinn_front_end_common.interface.buffer_management.**BufferManager**(*placements*, *tags*, *transceiver*, *store_to_file=False*)

Bases: object

Manager of send buffers

Parameters

- **placements** (*pacman.model.placements.Placements*) – The placements of the vertices
- **tags** (*pacman.model.tags.Tags*) – The tags assigned to the vertices
- **transceiver** (*spinnman.transceiver.Transceiver*) – The transceiver to use for sending and receiving information
- **store_to_file** (*bool*) – True if the data should be temporarily stored in a file instead of in RAM (default uses RAM)

add_receiving_vertex(*vertex*)

Add a vertex into the managed list for vertices which require buffers to be received from them during runtime

add_sender_vertex(*vertex*)

Add a vertex into the managed list for vertices which require buffers to be sent to them during runtime

Parameters **vertex** (spinnaker.pyNN.models.abstract_models.buffer_models.AbstractSendsBuffersFromHost) – the vertex to be managed

clear_recorded_data(*x*, *y*, *p*, *recording_region_id*)

Removes the recorded data stored in memory.

Parameters

- **x** – placement x coord
- **y** – placement y coord
- **p** – placement p coord
- **recording_region_id** – the recording region id

get_data_for_vertex(*placement*, *recording_region_id*)

Get a pointer to the data container for all the data retrieved during the simulation from a specific region area of a core

Parameters

- **placement** (pacman.model.placements.Placement) – the placement to get the data from
- **recording_region_id** (*int*) – desired recording data region

Returns pointer to a class which inherits from AbstractBufferedDataStorage

Return type spinn_front_end_common.interface.buffer_management.buffer_models.AbstractBufferedDataStorage

load_initial_buffers()

Load the initial buffers for the senders using mem writes

receive_buffer_command_message(*packet*)

Handle an EIEIO command message for the buffers

Parameters **packet** (*spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage*) – The eieio message received

reload_buffer_files
The file paths for each buffered region for each sender vertex

reset()
Resets the buffered regions to start transmitting from the beginning of its expected regions and clears the buffered out data files

resume()
Resets any data structures needed before starting running again

sender_vertices
The vertices which are buffered

stop()
Indicates that the simulation has finished, so no further outstanding requests need to be processed

spinn_front_end_common.interface.interface_functions package

Submodules

spinn_front_end_common.interface.interface_functions.application_finisher module

class spinn_front_end_common.interface.interface_functions.application_finisher.**ApplicationFi**
Bases: object

spinn_front_end_common.interface.interface_functions.application_runner module

class spinn_front_end_common.interface.interface_functions.application_runner.**ApplicationRunn**
Bases: object

Ensures all cores are initialised correctly, ran, and completed successfully.

run_application(*buffer_manager*, *notifier*, *executable_targets*, *executable_types*, *app_id*, *txrx*, *runtime*, *time_scale_factor*, *no_sync_changes*, *time_threshold*, *run_until_complete*)

spinn_front_end_common.interface.interface_functions.buffer_extractor module

class spinn_front_end_common.interface.interface_functions.buffer_extractor.**BufferExtractor**
Bases: object

Extracts data in between runs

spinn_front_end_common.interface.interface_functions.buffer_manager_creator module

class spinn_front_end_common.interface.interface_functions.buffer_manager_creator.**BufferManag**
Bases: object

spinn_front_end_common.interface.interface_functions.chip_iobuf_clearer module

class spinn_front_end_common.interface.interface_functions.chip_iobuf_clearer.**ChipIOBufCleare**
Bases: object

Updates the runtime of an application running on a spinnaker machine

spinn_front_end_common.interface.interface_functions.chip_iobuf_extractor module

class spinn_front_end_common.interface.interface_functions.chip_iobuf_extractor.**ChipIOBufExtr**
Bases: object

Extract iobuf buffers from the machine, and separates lines based on their prefix

spinn_front_end_common.interface.interface_functions.chip_provenance_updater module

class spinn_front_end_common.interface.interface_functions.chip_provenance_updater.**ChipProven**
Bases: object

Forces all cores to generate provenance data, and then exit

spinn_front_end_common.interface.interface_functions.chip_runtime_updater module

class spinn_front_end_common.interface.interface_functions.chip_runtime_updater.**ChipRuntimeUp**
Bases: object

Updates the runtime of an application running on a spinnaker machine

spinn_front_end_common.interface.interface_functions.database_interface module

class spinn_front_end_common.interface.interface_functions.database_interface.**DatabaseInterfa**
Bases: object

Writes a database of the graph(s) and other information

database_file_path

needs_database

spinn_front_end_common.interface.interface_functions.dsg_region_reloader module

class spinn_front_end_common.interface.interface_functions.dsg_region_reloader.**DSGRegionReloa**
Bases: object

Regenerates Data Specifications

spinn_front_end_common.interface.interface_functions.edge_to_n_keys_mapper module

class spinn_front_end_common.interface.interface_functions.edge_to_n_keys_mapper.**EdgeToNKeysM**
Bases: object

Works out the number of keys needed for each edge

spinn_front_end_common.interface.interface_functions.graph_binary_gatherer module

class spinn_front_end_common.interface.interface_functions.graph_binary_gatherer.**GraphBinaryG**
Bases: object

Extracts binaries to be executed

spinn_front_end_common.interface.interface_functions.graph_data_specification_writer module

class spinn_front_end_common.interface.interface_functions.graph_data_specification_writer.**Gr**
Bases: object

Executes data specification generation

spinn_front_end_common.interface.interface_functions.graph_measurer module

class spinn_front_end_common.interface.interface_functions.graph_measurer.**GraphMeasurer**
Bases: object

Works out how many chips a machine graph needs

spinn_front_end_common.interface.interface_functions.graph_provenance_gatherer module

class spinn_front_end_common.interface.interface_functions.graph_provenance_gatherer.**GraphPro**
Bases: object

spinn_front_end_common.interface.interface_functions.hbp_allocator module

class spinn_front_end_common.interface.interface_functions.hbp_allocator.**HBPAllocator**
Bases: object

Request a machine from the HBP remote access server that will fit a number of chips

spinn_front_end_common.interface.interface_functions.hbp_max_machine_generator module

class spinn_front_end_common.interface.interface_functions.hbp_max_machine_generator.**HBPMaxMa**
Bases: object

Generates the width and height of the maximum machine a given HBP server can generate

spinn_front_end_common.interface.interface_functions.host_execute_data_specification module

class spinn_front_end_common.interface.interface_functions.host_execute_data_specification.**Ho**
Bases: object

Executes the host based data specification

spinn_front_end_common.interface.interface_functions.insert_chip_power_monitors_to_graphs module

class spinn_front_end_common.interface.interface_functions.insert_chip_power_monitors_to_grap
Bases: object

function to add chip power monitors into a given graph

spinn_front_end_common.interface.interface_functions.insert_edges_to_live_packet_gatherers module

class spinn_front_end_common.interface.interface_functions.insert_edges_to_live_packet_gather
Bases: object

Add edges from the recorded vertices to the local Live PacketGatherers

spinn_front_end_common.interface.interface_functions.insert_extra_monitor_vertices_to_graphs module

class spinn_front_end_common.interface.interface_functions.insert_extra_monitor_vertices_to_g
Bases: object

inserts the extra monitor vertices into the graph.

spinn_front_end_common.interface.interface_functions.insert_live_packet_gatherers_to_graphs module

class spinn_front_end_common.interface.interface_functions.insert_live_packet_gatherers_to_gr
Bases: object

function to add LPG's as required into a given graph

spinn_front_end_common.interface.interface_functions.load_executable_images module

class spinn_front_end_common.interface.interface_functions.load_executable_images.**LoadExecuta**
Bases: object

spinn_front_end_common.interface.interface_functions.load_fixed_routes module

class spinn_front_end_common.interface.interface_functions.load_fixed_routes.**LoadFixedRoutes**
Bases: object

spinn_front_end_common.interface.interface_functions.locate_executable_start_type module

class spinn_front_end_common.interface.interface_functions.locate_executable_start_type.**Locat**
Bases: object

spinn_front_end_common.interface.interface_functions.machine_execute_data_specification module

class spinn_front_end_common.interface.interface_functions.machine_execute_data_specificatior
Bases: object

Executes the machine based data specification

spinnaker_based_data_specification_execution(*write_memory_map_report*, *dsg_targets*, *transceiver*, *app_id*)

Parameters

- **write_memory_map_report** –
- **dsg_targets** –
- **transceiver** –
- **app_id** –

Returns True

Return type bool

spinn_front_end_common.interface.interface_functions.machine_generator module

class spinn_front_end_common.interface.interface_functions.machine_generator.**MachineGenerator**

Bases: object

Interface to make a transceiver and a spinn_machine object

spinn_front_end_common.interface.interface_functions.notification_protocol module

class spinn_front_end_common.interface.interface_functions.notification_protocol.**Notification**

Bases: object

The notification protocol for external device interaction

send_read_notification(*database_directory*)

Send the read notifications via the notification protocol

Parameters **database_directory** – the path to the database

Return type None:

send_start_resume_notification()

Send the start notifications via the notification protocol

Return type None:

send_stop_pause_notification()

Send the stop or pause notifications via the notification protocol

Return type None:

stop()

Ends the notification protocol

Return type None:

wait_for_confirmation()

Waits for devices to confirm they have read the database via the notification protocol

Return type None:

spinn_front_end_common.interface.interface_functions.placements_provenance_gatherer module

class spinn_front_end_common.interface.interface_functions.placements_provenance_gatherer.**Pla**

Bases: object

spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_chip_power_monitor module

class spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_chip_po
Bases: object

Adds chip power monitor resources as required for a machine

spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_live_packet_gatherers module

class spinn_front_end_common.interface.interface_functions.pre_allocate_resources_for_live_pa
Bases: object

Adds Live Packet Gatherer resources as required for a machine

spinn_front_end_common.interface.interface_functions.preallocate_resources_for_extra_monitor_support module

class spinn_front_end_common.interface.interface_functions.preallocate_resources_for_extra_mo
Bases: object

allocate resources needed for the extra monitor support

spinn_front_end_common.interface.interface_functions.profile_data_gatherer module

class spinn_front_end_common.interface.interface_functions.profile_data_gatherer.**ProfileDataG**
Bases: object

spinn_front_end_common.interface.interface_functions.provenance_json_writer module

class spinn_front_end_common.interface.interface_functions.provenance_json_writer.**ProvenanceJ**
Bases: object

Write provenance data into JSON

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', 'A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H

spinn_front_end_common.interface.interface_functions.provenance_xml_writer module

class spinn_front_end_common.interface.interface_functions.provenance_xml_writer.**ProvenanceXM**
Bases: object

Write provenance data into XML

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', 'A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H

spinn_front_end_common.interface.interface_functions.router_provenance_gatherer module

class spinn_front_end_common.interface.interface_functions.router_provenance_gatherer.**RouterP**
Bases: object

RouterProvenanceGatherer: gathers diagnostics from the routers.

spinn_front_end_common.interface.interface_functions.routing_setup module

class spinn_front_end_common.interface.interface_functions.routing_setup.**RoutingSetup**
Bases: object

spinn_front_end_common.interface.interface_functions.routing_table_loader module

class spinn_front_end_common.interface.interface_functions.routing_table_loader.**RoutingTableL**
Bases: object

spinn_front_end_common.interface.interface_functions.spalloc_allocator module

class spinn_front_end_common.interface.interface_functions.spalloc_allocator.**SpallocAllocator**
Bases: object

Request a machine from a SPALLOC server that will fit the given number of chips

spinn_front_end_common.interface.interface_functions.spalloc_max_machine_generator module

class spinn_front_end_common.interface.interface_functions.spalloc_max_machine_generator.**Spal**
Bases: object

Generates the width and height of the maximum machine a given allocation server can generate

spinn_front_end_common.interface.interface_functions.tags_loader module

class spinn_front_end_common.interface.interface_functions.tags_loader.**TagsLoader**
Bases: object

Loads tags onto the machine

static load_iptags (*iptags*, *transceiver*, *progress_bar*)
Loads all the iptags individually.

Parameters

- **iptags** – the iptags to be loaded.
- **transceiver** – the transceiver object

Return type None

static load_reverse_iptags (*reverse_ip_tags*, *transceiver*, *progress_bar*)
Loads all the reverse iptags individually.

Parameters

- **reverse_ip_tags** – the reverse iptags to be loaded

- **transceiver** – the transceiver object

Return type None

spinn_front_end_common.interface.interface_functions.tdma_agenda_builder module

class spinn_front_end_common.interface.interface_functions.tdma_agenda_builder.**TDMAAgendaBuil**
Bases: object

algorithm that builds an agenda for transmissions. It uses a TDMA (Time division multiple access) system and graph colouring to deduce the agenda set up. Ensures parallel transmissions so that the destination should never be overloaded

spinn_front_end_common.interface.interface_functions.virtual_machine_generator module

class spinn_front_end_common.interface.interface_functions.virtual_machine_generator.**VirtualM**
Bases: object

Module contents

class spinn_front_end_common.interface.interface_functions.**ApplicationFinisher**
Bases: object

class spinn_front_end_common.interface.interface_functions.**ApplicationRunner**
Bases: object

Ensures all cores are initialised correctly, ran, and completed successfully.

run_application(*buffer_manager*, *notifier*, *executable_targets*, *executable_types*, *app_id*, *txrx*, *runtime*, *time_scale_factor*, *no_sync_changes*, *time_threshold*, *run_until_complete*)

class spinn_front_end_common.interface.interface_functions.**BufferExtractor**
Bases: object

Extracts data in between runs

class spinn_front_end_common.interface.interface_functions.**BufferManagerCreator**
Bases: object

class spinn_front_end_common.interface.interface_functions.**ChipIOBufClearer**
Bases: object

Updates the runtime of an application running on a spinnaker machine

class spinn_front_end_common.interface.interface_functions.**ChipIOBufExtractor**
Bases: object

Extract iobuf buffers from the machine, and separates lines based on their prefix

class spinn_front_end_common.interface.interface_functions.**ChipProvenanceUpdater**
Bases: object

Forces all cores to generate provenance data, and then exit

class spinn_front_end_common.interface.interface_functions.**ChipRuntimeUpdater**
Bases: object

Updates the runtime of an application running on a spinnaker machine

class spinn_front_end_common.interface.interface_functions.**DatabaseInterface**
Bases: object

Writes a database of the graph(s) and other information

database_file_path

needs_database

class spinn_front_end_common.interface.interface_functions.**DSGRegionReloader**
Bases: object

Regenerates Data Specifications

class spinn_front_end_common.interface.interface_functions.**EdgeToNKeysMapper**
Bases: object

Works out the number of keys needed for each edge

class spinn_front_end_common.interface.interface_functions.**GraphBinaryGatherer**
Bases: object

Extracts binaries to be executed

class spinn_front_end_common.interface.interface_functions.**GraphDataSpecificationWriter**
Bases: object

Executes data specification generation

class spinn_front_end_common.interface.interface_functions.**GraphMeasurer**
Bases: object

Works out how many chips a machine graph needs

class spinn_front_end_common.interface.interface_functions.**GraphProvenanceGatherer**
Bases: object

class spinn_front_end_common.interface.interface_functions.**HBPAllocator**
Bases: object

Request a machine from the HBP remote access server that will fit a number of chips

class spinn_front_end_common.interface.interface_functions.**HBPMaxMachineGenerator**
Bases: object

Generates the width and height of the maximum machine a given HBP server can generate

class spinn_front_end_common.interface.interface_functions.**HostExecuteDataSpecification**
Bases: object

Executes the host based data specification

class spinn_front_end_common.interface.interface_functions.**InsertEdgesToLivePacketGatherers**
Bases: object

Add edges from the recorded vertices to the local Live PacketGatherers

class spinn_front_end_common.interface.interface_functions.**InsertLivePacketGathererersToGraphs**
Bases: object

function to add LPG's as required into a given graph

class spinn_front_end_common.interface.interface_functions.**LoadExecutableImages**
Bases: object

class spinn_front_end_common.interface.interface_functions.**LocateExecutableStartType**
Bases: object

class spinn_front_end_common.interface.interface_functions.**MachineExecuteDataSpecification**
Bases: object

Executes the machine based data specification

spinnaker_based_data_specification_execution(*write_memory_map_report*, *dsg_targets*, *transceiver*, *app_id*)

Parameters

- **write_memory_map_report** –
- **dsg_targets** –
- **transceiver** –
- **app_id** –

Returns True

Return type bool

class spinn_front_end_common.interface.interface_functions.**MachineGenerator**
Bases: object

Interface to make a transceiver and a spinn_machine object

class spinn_front_end_common.interface.interface_functions.**NotificationProtocol**
Bases: object

The notification protocol for external device interaction

send_read_notification(*database_directory*)
Send the read notifications via the notification protocol

Parameters **database_directory** – the path to the database

Return type None:

send_start_resume_notification()
Send the start notifications via the notification protocol

Return type None:

send_stop_pause_notification()
Send the stop or pause notifications via the notification protocol

Return type None:

stop()
Ends the notification protocol

Return type None:

wait_for_confirmation()
Waits for devices to confirm they have read the database via the notification protocol

Return type None:

class spinn_front_end_common.interface.interface_functions.**PlacementsProvenanceGatherer**
Bases: object

class spinn_front_end_common.interface.interface_functions.**PreAllocateResourcesForLivePacketG**
Bases: object

Adds Live Packet Gatherer resources as required for a machine

class spinn_front_end_common.interface.interface_functions.**ProvenanceJSONWriter**
Bases: object

Write provenance data into JSON

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', 'A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H

class spinn_front_end_common.interface.interface_functions.**ProvenanceXMLWriter**
Bases: object

Write provenance data into XML

VALID_CHARS = frozenset([' ', ')', '(', '-', '.', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', 'A', 'C', 'B', 'E', 'D', 'G', 'F', 'I', 'H

class spinn_front_end_common.interface.interface_functions.**RouterProvenanceGatherer**
Bases: object

RouterProvenanceGatherer: gathers diagnostics from the routers.

class spinn_front_end_common.interface.interface_functions.**RoutingSetup**
Bases: object

class spinn_front_end_common.interface.interface_functions.**RoutingTableLoader**
Bases: object

class spinn_front_end_common.interface.interface_functions.**SpallocAllocator**
Bases: object

Request a machine from a SPALLOC server that will fit the given number of chips

class spinn_front_end_common.interface.interface_functions.**SpallocMaxMachineGenerator**
Bases: object

Generates the width and height of the maximum machine a given allocation server can generate

class spinn_front_end_common.interface.interface_functions.**TagsLoader**
Bases: object

Loads tags onto the machine

static load_iptags (*iptags*, *transceiver*, *progress_bar*)
Loads all the iptags individually.

Parameters

- **iptags** – the iptags to be loaded.
- **transceiver** – the transceiver object

Return type None

static load_reverse_iptags (*reverse_ip_tags*, *transceiver*, *progress_bar*)
Loads all the reverse iptags individually.

Parameters

- **reverse_ip_tags** – the reverse iptags to be loaded
- **transceiver** – the transceiver object

Return type None

class spinn_front_end_common.interface.interface_functions.**TDMAAgendaBuilder**
Bases: object

algorithm that builds an agenda for transmissions. It uses a TDMA (Time division multiple access) system and graph colouring to deduce the agenda set up. Ensures parallel transmissions so that the destination should never be overloaded

class `spinn_front_end_common.interface.interface_functions.`**`VirtualMachineGenerator`**
Bases: `object`

spinn_front_end_common.interface.profiling package

Submodules

spinn_front_end_common.interface.profiling.abstract_has_profile_data module

class `spinn_front_end_common.interface.profiling.abstract_has_profile_data.`**`AbstractHasProfile`**
Bases: `object`

Indicates an object that can record a profile

`get_profile_data`(*transceiver*, *placement*)
Get the profile data recorded during simulation

Return type *`spinn_front_end_common.interface.profiling.profile_data.ProfileData`*

spinn_front_end_common.interface.profiling.profile_data module

class `spinn_front_end_common.interface.profiling.profile_data.`**`ProfileData`**(*tag_labels*)
Bases: `object`

A container for profile data

Parameters **`tag_labels`** (*`list (str)`*) – A list of labels indexed by tag id

`DURATION` = 1

`START_TIME` = 0

`add_data`(*data*)
Add profiling data read from the profile section

Parameters **`data`** (*`bytearray`*) – Data read from the profile section on the machine

`get_mean_ms`(*tag*)
Get the mean time in milliseconds spent on operations with the given tag

Parameters **`tag`** (*`str`*) – The tag to get the mean time for

Return type float

`get_mean_ms_per_ts`(*tag*, *run_time_ms*, *machine_time_step_ms*)
Get the mean time in milliseconds spent on operations with the given tag per timestep

Parameters

- **`tag`** (*`str`*) – The tag to get the data for
- **`machine_time_step_ms`** (*`int`*) – The time step of the simulation in microseconds
- **`run_time_ms`** (*`float`*) – The run time of the simulation in milliseconds

Return type float

get_mean_n_calls_per_ts (*tag*, *run_time_ms*, *machine_time_step_ms*)
Get the mean number of times the given tag was recorded per timestep

Parameters

- **tag** (*str*) – The tag to get the data for
- **machine_time_step_ms** (*int*) – The time step of the simulation in microseconds
- **run_time_ms** (*float*) – The run time of the simulation in milliseconds

Return type float

get_n_calls (*tag*)
Get the number of times the given tag was recorded

Parameters **tag** (*str*) – The tag to get the number of calls of

Return type int

tags
The tags recorded as labels

Return type list of str

spinn_front_end_common.interface.profiling.profile_utils module

spinn_front_end_common.interface.profiling.profile_utils.**get_profile_region_size** (*n_samples*)
Get the size of the region of the profile data

Parameters **n_samples** – number of different samples to record

Returns the size in bytes used by the profile region

spinn_front_end_common.interface.profiling.profile_utils.**get_profiling_data** (*profile_region*, *tag_labels*, *txrx*, *placement*)
Utility function to get profile data from a profile region

Parameters

- **profile_region** – dsg region to get profiling data out of sdram
- **tag_labels** – labels for the profiling data
- **txrx** – transceiver code
- **placement** – placement

Returns ProfileData

spinn_front_end_common.interface.profiling.profile_utils.**reserve_profile_region** (*spec*, *region*, *n_samples*)
Reserves the profile region for recording the profile data

Parameters

- **spec** – the dsg specification writer
- **region** – region id for the profile data

- **n_samples** – n elements being sampled

Return type None

spinn_front_end_common.interface.profiling.profile_utils.**write_profile_region_data**(*spec*, *region*, *n_samples*)

Writes the profile region data

Parameters

- **spec** – the dsg specification writer
- **region** – region id for the profile data
- **n_samples** – n elements being sampled

Return type None

Module contents

spinn_front_end_common.interface.provenance package

Submodules

spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data module

class spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data.**Abs**

Bases: object

Indicates an object that provides locally obtained provenance data

get_local_provenance_data()

Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_from_machine module

class spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_from_mach

Bases: object

Indicates that an object provides provenance data retrieved from the machine

get_provenance_data_from_machine(*transceiver*, *placement*)

Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

spinn_front_end_common.interface.provenance.pacman_provenance_extractor module

class spinn_front_end_common.interface.provenance.pacman_provenance_extractor.**PacmanProvenanc**
Bases: object

Extracts Provenance data from a PACMANAlgorithmExecutor

clear()
clears the provenance data store

Return type None

data_items
returns the provenance data items

Returns list of provenance data items.

Return type iterable of ProvenanceDataItem

extract_provenance(*executor*)
acquires the timings from pacman algorithms (provenance data)

Parameters **executor** – the pacman workflow executor

Return type None

spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl module

class spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.
Bases: *spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_fro*
AbstractProvidesProvenanceDataFromMachine

An implementation that gets provenance data from a region of ints on the machine

class PROVENANCE_DATA_ENTRIES
Bases: enum.Enum

CALLBACK_QUEUE_OVERLOADED = 1

DMA_QUEUE_OVERLOADED = 2

MAX_NUMBER_OF_TIMER_TIC_OVERRUN = 4

TIMER_TIC_HAS_OVERRUN = 3

TRANSMISSION_EVENT_OVERFLOW = 0

get_provenance_data_from_machine(*transceiver*, *placement*)

static get_provenance_data_size(*n_additional_data_items*)

reserve_provenance_data_region(*spec*)

Module contents

class spinn_front_end_common.interface.provenance.**AbstractProvidesLocalProvenanceData**
Bases: object

Indicates an object that provides locally obtained provenance data

get_local_provenance_data()
Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

class spinn_front_end_common.interface.provenance.**AbstractProvidesProvenanceDataFromMachine**
Bases: object

Indicates that an object provides provenance data retrieved from the machine

get_provenance_data_from_machine(*transceiver*, *placement*)
Get an iterable of provenance data items

Parameters

- **transceiver** – the SpinnMan interface object
- **placement** – the placement of the object

Returns iterable of ProvenanceDataItem

class spinn_front_end_common.interface.provenance.**PacmanProvenanceExtractor**
Bases: object

Extracts Provenance data from a PACMANAlgorithmExecutor

clear()
clears the provenance data store

Return type None

data_items
returns the provenance data items

Returns list of provenance data items.

Return type iterable of ProvenanceDataItem

extract_provenance(*executor*)
acquires the timings from pacman algorithms (provenance data)

Parameters **executor** – the pacman workflow executor

Return type None

class spinn_front_end_common.interface.provenance.**ProvidesProvenanceDataFromMachineImpl**
Bases: *spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_fro*
AbstractProvidesProvenanceDataFromMachine

An implementation that gets provenance data from a region of ints on the machine

class PROVENANCE_DATA_ENTRIES
Bases: enum.Enum

CALLBACK_QUEUE_OVERLOADED = 1

DMA_QUEUE_OVERLOADED = 2

MAX_NUMBER_OF_TIMER_TIC_OVERRUN = 4

TIMER_TIC_HAS_OVERRUN = 3

TRANSMISSION_EVENT_OVERFLOW = 0

get_provenance_data_from_machine(*transceiver*, *placement*)

static get_provenance_data_size(*n_additional_data_items*)

reserve_provenance_data_region(*spec*)

spinn_front_end_common.interface.simulation package

Submodules

spinn_front_end_common.interface.simulation.simulation_utilities module

spinn_front_end_common.interface.simulation.simulation_utilities.**get_simulation_header_arra**

Get data to be written to the simulation header

Parameters

- **binary_file_name** – The name of the binary of the application
- **machine_time_step** – The time step of the simulation
- **time_scale_factor** – The time scaling of the simulation

Returns An array of values to be written as the simulation header

Module contents

Submodules

spinn_front_end_common.interface.abstract_spinnaker_base module

main interface for the spinnaker tools

class spinn_front_end_common.interface.abstract_spinnaker_base.**AbstractSpinnakerBase**(*configfile*, *ex-*
e-
cutable_fin
graph_labe
database_s
ex-
tra_algorith
n_chips_re
de-
fault_config
val-
i-
da-
tion_cfg=N
front_end_v

Bases: *spinn_front_end_common.utilities.simulator_interface.SimulatorInterface*

Main interface into the tools logic flow

add_application_edge(*edge_to_add*, *partition_identifier*)

Parameters

- **edge_to_add** –
- **partition_identifier** – the partition identifier for the outgoing edge partition

Return type None

add_application_vertex(*vertex_to_add*)

Parameters **vertex_to_add** – the vertex to add to the graph

Return type None

Raises ConfigurationException when both graphs contain vertices

add_extraction_timing(*timing*)

add_live_packet_gatherer_parameters(*live_packet_gatherer_params*, *vertex_to_record_from*)

adds params for a new LPG if needed, or adds to the tracker for same params.

Parameters

- **live_packet_gatherer_params** – params to look for a LPG
- **vertex_to_record_from** – the vertex that needs to send to a given LPG

Return type None

add_machine_edge(*edge*, *partition_id*)

Parameters

- **edge** – the edge to add to the graph
- **partition_id** – the partition identifier for the outgoing edge partition

Return type None

add_machine_vertex(*vertex*)

Parameters **vertex** – the vertex to add to the graph

Return type None

Raises ConfigurationException when both graphs contain vertices

add_socket_address(*socket_address*)

Parameters **socket_address** –

Return type None

application_graph

buffer_manager

The buffer manager being used for loading/extracting buffers

config

helper method for the front end implementations until we remove config

dsg_algorithm

The dsg algorithm used by the tools

exception_handler(*exctype*, *value*, *traceback_obj*)

handler of exceptions

Parameters

- **exctype** – the type of execution received
- **value** – the value of the exception
- **traceback_obj** – the trace back stuff

extend_extra_load_algorithms(*extra_load_algorithms*)

extend_extra_mapping_algorithms(*extra_mapping_algorithms*)

extend_extra_post_run_algorithms(*extra_post_run_algorithms*)

generate_file_machine()

get_current_time()

get_number_of_available_cores_on_machine

returns the number of available cores on the machine after taking into account pre allocated resources

Returns number of available cores

Return type int

graph_mapper

has_ran

has_reset_last

increment_none_labelled_edge_count()

Increment the number of new edges which have not been labelled.

increment_none_labelled_vertex_count()

Increment the number of new vertices which have not been labelled.

machine

The python machine object

Return type *spinn_machine.Machine*

machine_graph

machine_time_step

no_machine_time_steps

none_labelled_edge_count

The number of times edges have not been labelled.

none_labelled_vertex_count

The number of times vertices have not been labelled.

placements

prepend_extra_pre_run_algorithms(*extra_pre_run_algorithms*)

reset()

Code that puts the simulation back at time zero

routing_infos

run(*run_time*)

Run a simulation for a fixed amount of time

Parameters **run_time** – the run duration in milliseconds.

run_until_complete()

Run a simulation until it completes

set_n_chips_required(*n_chips_required*)

set_up_machine_specifics(*hostname*)
Adds machine specifics for the different modes of execution

Parameters **hostname** – machine name

Return type None

set_up_timings(*machine_time_step=None*, *time_scale_factor=None*)
Set up timings of the machine

Parameters

- **machine_time_step** – An explicitly specified time step for the machine. If None, the value is read from the config
- **time_scale_factor** – An explicitly specified time scale factor for the simulation. If None, the value is read from the config

signal_handler(*signal*, *frame*)
handles closing down of script via keyboard interrupt

Parameters

- **signal** – the signal received
- **frame** – frame executed in

Returns None

stop(*turn_off_machine=None*, *clear_routing_tables=None*, *clear_tags=None*)

Parameters

- **turn_off_machine** (*bool*) – decides if the machine should be powered down after running the execution. Note that this powers down all boards connected to the BMP connections given to the transceiver
- **clear_routing_tables** (*bool*) – informs the tool chain if it should turn off the clearing of the routing tables
- **clear_tags** (*boolean*) – informs the tool chain if it should clear the tags off the machine at stop

Return type None

tags

timescale_factor

transceiver

update_extra_inputs(*extra_inputs*)

update_extra_mapping_inputs(*extra_mapping_inputs*)

use_virtual_board
True if this run is using a virtual machine

verify_not_running()

spinn_front_end_common.interface.simulator_state module

class spinn_front_end_common.interface.simulator_state.**Simulator_State**(*value*, *doc=''*)

Bases: enum.Enum

Different States the SimulATOR COULD BE IN

FINISHED = 3

INIT = 0

IN_RUN = 1

RUN_FOREVER = 2

SHUTDOWN = 4

Module contents

spinn_front_end_common.mapping_algorithms package

Subpackages

spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression package

Submodules

spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression.mundy_on_chip_router_comp module

class spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression.mundy_on_chi

Bases: object

Compressor that uses a on chip router compressor

OVER_RUN_THRESHOLD_BEFORE_ERROR = 1000

SIZE_OF_A_SDRAM_ENTRY = 16

SURPLUS_DATA_ENTRIES = 12

TIME_EXPECTED_TO_RUN = 1000

Module contents

Module contents

spinn_front_end_common.utilities package

Subpackages

spinn_front_end_common.utilities.connections package

Submodules

spinn_front_end_common.utilities.connections.live_event_connection module

class `spinn_front_end_common.utilities.connections.live_event_connection.`**`LiveEventConnection`**(

Bases: *spinn_front_end_common.utilities.database.database_connection.DatabaseConnection*

A connection for receiving and sending live events from and to SpiNNaker

Parameters

- **live_packet_gather_label** – The label of the LivePacketGather vertex to which received events are being sent
- **receive_labels** (*iterable of str*) – Labels of vertices from which live events will be received.
- **send_labels** (*iterable of str*) – Labels of vertices to which live events will be sent
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_init_callback(*label*, *init_callback*)

Add a callback to be called to initialise a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **init_callback** (*function(str, int, float, float) -> None*) – A function to be called to initialise the vertex. This should take as parameters the label

of the vertex, the number of neurons in the population, the run time of the simulation in milliseconds, and the simulation timestep in milliseconds

add_pause_stop_callback(*label*, *pause_stop_callback*)
Add a callback for the pause and stop state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **pause_stop_callback** (function(str, `SpynnakerLiveEventConnection`) -> None) – A function to be called when the pause or stop message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events

Return type None

add_receive_callback(*label*, *live_event_callback*)
Add a callback for the reception of live events from a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **live_event_callback** (function*(str, int, [int]) -> None*) – A function to be called when events are received. This should take as parameters the label of the vertex, the simulation timestep when the event occurred, and an array-like of atom ids.

add_start_callback(*label*, *start_callback*)
Add a callback for the start of the simulation

Parameters

- **start_callback** (function(str, `SpynnakerLiveEventConnection`) -> None) – A function to be called when the start message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events
- **label** (*str*) – the label of the function to be sent

add_start_resume_callback(*label*, *start_resume_callback*)

close()

send_event(*label*, *atom_id*, *send_full_keys=False*)
Send an event from a single atom

Parameters

- **label** (*str*) – The label of the vertex from which the event will originate
- **atom_id** (*int*) – The id of the atom sending the event
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

send_events(*label*, *atom_ids*, *send_full_keys=False*)
Send a number of events

Parameters

- **label** (*str*) – The label of the vertex from which the events will originate
- **atom_ids** (*[int]*) – array-like of atom ids sending events

- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

Module contents

class spinn_front_end_common.utilities.connections.**LiveEventConnection**(*live_packet_gather_label*, *receive_labels=None*, *send_labels=None*, *local_host=None*, *local_port=19999*, *machine_vertices=False*)

Bases: *spinn_front_end_common.utilities.database.database_connection.DatabaseConnection*

A connection for receiving and sending live events from and to SpiNNaker

Parameters

- **live_packet_gather_label** – The label of the LivePacketGather vertex to which received events are being sent
- **receive_labels** (*iterable of str*) – Labels of vertices from which live events will be received.
- **send_labels** (*iterable of str*) – Labels of vertices to which live events will be sent
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_init_callback(*label*, *init_callback*)

Add a callback to be called to initialise a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **init_callback** (function*(str, int, float, float) -> None*) – A function to be called to initialise the vertex. This should take as parameters the label of the vertex, the number of neurons in the population, the run time of the simulation in milliseconds, and the simulation timestep in milliseconds

add_pause_stop_callback(*label*, *pause_stop_callback*)

Add a callback for the pause and stop state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **pause_stop_callback** (function(str, SpynnakerLiveEventConnection) -> None) – A function to be called when the pause or stop message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events

Return type None

add_receive_callback(*label*, *live_event_callback*)

Add a callback for the reception of live events from a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **live_event_callback** (function*(str, int, [int]) -> None*) – A function to be called when events are received. This should take as parameters the label of the vertex, the simulation timestep when the event occurred, and an array-like of atom ids.

add_start_callback(*label*, *start_callback*)

Add a callback for the start of the simulation

Parameters

- **start_callback** (function(str, SpynnakerLiveEventConnection) -> None) – A function to be called when the start message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events
- **label** (*str*) – the label of the function to be sent

add_start_resume_callback(*label*, *start_resume_callback*)

close()

send_event(*label*, *atom_id*, *send_full_keys=False*)

Send an event from a single atom

Parameters

- **label** (*str*) – The label of the vertex from which the event will originate
- **atom_id** (*int*) – The id of the atom sending the event
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

send_events(*label*, *atom_ids*, *send_full_keys=False*)

Send a number of events

Parameters

- **label** (*str*) – The label of the vertex from which the events will originate
- **atom_ids** (*[int]*) – array-like of atom ids sending events
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each atom from the database, or whether to send 16-bit atom ids directly

spinn_front_end_common.utilities.database package

Submodules

spinn_front_end_common.utilities.database.database_connection module

class spinn_front_end_common.utilities.database.database_connection.**DatabaseConnection**(*start_res stop_pa lo- cal_host lo- cal_port*

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, threading.Thread

A connection from the toolchain which will be notified when the database has been written, and can then respond when the database has been read, and further wait for notification that the simulation has started.

Parameters

- **start_resume_callback_function** (*function() -> None*) – A function to be called when the start message has been received. This function should not take any parameters or return anything.
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_database_callback(*database_callback_function*)

Add a database callback to be called when the database is ready

Parameters **database_callback_function** (function(spynnaker_external_devices.pyNN.connections.database_reader.DatabaseReader) -> None) – A function to be called when the database message has been received. This function should take a single parameter, which will be a DatabaseReader object. Once the function returns, it will be assumed that the database has been read, and the return response will be sent.

close()

run()

spinn_front_end_common.utilities.database.database_reader module

class spinn_front_end_common.utilities.database.database_reader.**DatabaseReader**(*database_path*)

Bases: object

A reader for the database

Parameters **database_path** (*str*) – The path to the database

close()

cursor

The database cursor. Allows custom SQL queries to be performed.

Return type sqlite3.Cursor

get_atom_id_to_key_mapping(*label*)
Get a mapping of atom id to event key for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns dictionary of event keys indexed by atom id

get_configuration_parameter_value(*parameter_name*)
Get the value of a configuration parameter

Parameters **parameter_name** (*str*) – The name of the parameter

Returns The value of the parameter

Return type float

get_key_to_atom_id_mapping(*label*)
Get a mapping of event key to atom id for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns dictionary of atom ids indexed by event key

Return type dict

get_live_input_details(*label*)
Get the ip address and port where live input should be sent for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_input_details(*label*)
Get the ip address and port where live input should be sent for a given machine vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_machine_live_input_key(*label*)

get_machine_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a machine vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_output_key(*label*, *receiver_label*)

get_n_atoms(*label*)
Get the number of atoms in a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns The number of atoms

Return type int

spinn_front_end_common.utilities.database.database_writer module

class spinn_front_end_common.utilities.database.database_writer.**DatabaseWriter**(*database_directory*)

Bases: object

The interface for the database system for main front ends. Any special tables needed from a front end should be done by sub classes of this interface.

add_application_vertices(*application_graph*)

Parameters **application_graph** –

Return type None

add_machine_objects(*machine*)

Store the machine object into the database

Parameters **machine** – the machine object.

Return type None

add_placements(*placements*)

Adds the placements objects into the database

Parameters

- **placements** – the placements object
- **machine_graph** – the machine graph object

Return type None

add_routing_infos(*routing_infos*, *machine_graph*)

Adds the routing information (key masks etc) into the database

Parameters

- **routing_infos** – the routing information object
- **machine_graph** – the machine graph object

Return type None:

add_routing_tables(*routing_tables*)

Adds the routing tables into the database

Parameters **routing_tables** – the routing tables object

Return type None

add_system_params(*time_scale_factor*, *machine_time_step*, *runtime*)

Write system params into the database

Parameters

- **time_scale_factor** – the time scale factor used in timing
- **machine_time_step** – the machine time step used in timing
- **runtime** – the amount of time the application is to run for

add_tags (*machine_graph*, *tags*)
Adds the tags into the database

Parameters

- **machine_graph** – the machine graph object
- **tags** – the tags object

Return type None

add_vertices (*machine_graph*, *graph_mapper*, *application_graph*)
Add the machine graph, graph mapper and application graph into the database.

Parameters

- **machine_graph** – the machine graph object
- **graph_mapper** – the graph mapper object
- **application_graph** – the application graph object

Return type None

static auto_detect_database (*machine_graph*)
Auto detects if there is a need to activate the database system

Parameters **machine_graph** – the machine graph of the application problem space.

Returns a bool which represents if the database is needed

create_atom_to_event_id_mapping (*application_graph*, *machine_graph*, *routing_infos*, *graph_mapper*)

Parameters

- **application_graph** –
- **machine_graph** –
- **routing_infos** –
- **graph_mapper** –

Return type None

create_schema ()

database_path

Module contents

class spinn_front_end_common.utilities.database.**DatabaseConnection** (*start_resume_callback_function=None*, *stop_pause_callback_function=None*, *local_host=None*, *local_port=19999*)

Bases: *spinnman.connections.udp_packet_connections.udp_connection.UDPConnection*, threading.Thread

A connection from the toolchain which will be notified when the database has been written, and can then respond when the database has been read, and further wait for notification that the simulation has started.

Parameters

- **start_resume_callback_function** (*function() -> None*) – A function to be called when the start message has been received. This function should not take any parameters or return anything.
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

add_database_callback(*database_callback_function*)

Add a database callback to be called when the database is ready

Parameters **database_callback_function** (function(spynnaker_external_devices.pyNN.connections.database_reader.DatabaseReader) -> None) – A function to be called when the database message has been received. This function should take a single parameter, which will be a DatabaseReader object. Once the function returns, it will be assumed that the database has been read, and the return response will be sent.

close()

run()

class spinn_front_end_common.utilities.database.**DatabaseReader**(*database_path*)

Bases: object

A reader for the database

Parameters **database_path** (*str*) – The path to the database

close()

cursor

The database cursor. Allows custom SQL queries to be performed.

Return type sqlite3.Cursor

get_atom_id_to_key_mapping(*label*)

Get a mapping of atom id to event key for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns dictionary of event keys indexed by atom id

get_configuration_parameter_value(*parameter_name*)

Get the value of a configuration parameter

Parameters **parameter_name** (*str*) – The name of the parameter

Returns The value of the parameter

Return type float

get_key_to_atom_id_mapping(*label*)

Get a mapping of event key to atom id for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns dictionary of atom ids indexed by event key

Return type dict

get_live_input_details(*label*)

Get the ip address and port where live input should be sent for a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_input_details(*label*)
Get the ip address and port where live input should be sent for a given machine vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port)

Return type (str, int)

get_machine_live_input_key(*label*)

get_machine_live_output_details(*label*, *receiver_label*)
Get the ip address, port and whether the SDP headers are to be stripped from the output from a machine vertex

Parameters **label** (*str*) – The label of the vertex

Returns tuple of (ip address, port, strip SDP)

Return type (str, int, bool)

get_machine_live_output_key(*label*, *receiver_label*)

get_n_atoms(*label*)
Get the number of atoms in a given vertex

Parameters **label** (*str*) – The label of the vertex

Returns The number of atoms

Return type int

class spinn_front_end_common.utilities.database.**DatabaseWriter**(*database_directory*)
Bases: object

The interface for the database system for main front ends. Any special tables needed from a front end should be done by sub classes of this interface.

add_application_vertices(*application_graph*)

Parameters **application_graph** –

Return type None

add_machine_objects(*machine*)
Store the machine object into the database

Parameters **machine** – the machine object.

Return type None

add_placements(*placements*)
Adds the placements objects into the database

Parameters

- **placements** – the placements object
- **machine_graph** – the machine graph object

Return type None

add_routing_infos (*routing_infos*, *machine_graph*)
Adds the routing information (key masks etc) into the database

Parameters

- **routing_infos** – the routing information object
- **machine_graph** – the machine graph object

Return type None:

add_routing_tables (*routing_tables*)
Adds the routing tables into the database

Parameters **routing_tables** – the routing tables object

Return type None

add_system_params (*time_scale_factor*, *machine_time_step*, *runtime*)
Write system params into the database

Parameters

- **time_scale_factor** – the time scale factor used in timing
- **machine_time_step** – the machine time step used in timing
- **runtime** – the amount of time the application is to run for

add_tags (*machine_graph*, *tags*)
Adds the tags into the database

Parameters

- **machine_graph** – the machine graph object
- **tags** – the tags object

Return type None

add_vertices (*machine_graph*, *graph_mapper*, *application_graph*)
Add the machine graph, graph mapper and application graph into the database.

Parameters

- **machine_graph** – the machine graph object
- **graph_mapper** – the graph mapper object
- **application_graph** – the application graph object

Return type None

static auto_detect_database (*machine_graph*)
Auto detects if there is a need to activate the database system

Parameters **machine_graph** – the machine graph of the application problem space.

Returns a bool which represents if the database is needed

create_atom_to_event_id_mapping (*application_graph*, *machine_graph*, *routing_infos*, *graph_mapper*)

Parameters

- **application_graph** –
- **machine_graph** –
- **routing_infos** –
- **graph_mapper** –

Return type None

create_schema()

database_path

spinn_front_end_common.utilities.notification_protocol package

Submodules

spinn_front_end_common.utilities.notification_protocol.notification_protocol module

class spinn_front_end_common.utilities.notification_protocol.notification_protocol.**Notificati**

Bases: object

The protocol which hand shakes with external devices about the database and starting execution

close()

Closes the thread pool

send_read_notification(*database_path*)

sends notifications to all devices which have expressed an interest in when the database has been written

Parameters **database_path** – the path to the database file

Return type None

send_start_resume_notification()

either waits till all sources have confirmed read the database and are configured, and/or just sends the start notification (when the system is executing)

Return type None

send_stop_pause_notification()

sends the pause / stop notifications when the script has either finished or paused

Return type None

wait_for_confirmation()

if asked to wait for confirmation, waits for all external systems to confirm that they are configured and have read the database

Return type None

spinn_front_end_common.utilities.notification_protocol.socket_address module

class spinn_front_end_common.utilities.notification_protocol.socket_address.**SocketAddress**(*notify_listen_*

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

Module contents

class spinn_front_end_common.utilities.notification_protocol.**NotificationProtocol**(*socket_addresse wait_for_read_*

Bases: object

The protocol which hand shakes with external devices about the database and starting execution

close()
Closes the thread pool

send_read_notification(*database_path*)
sends notifications to all devices which have expressed an interest in when the database has been written

Parameters **database_path** – the path to the database file

Return type None

send_start_resume_notification()
either waits till all sources have confirmed read the database and are configured, and/or just sends the start notification (when the system is executing)

Return type None

send_stop_pause_notification()
sends the pause / stop notifications when the script has either finished or paused

Return type None

wait_for_confirmation()
if asked to wait for confirmation, waits for all external systems to confirm that they are configured and have read the database

Return type None

class spinn_front_end_common.utilities.notification_protocol.**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

spinn_front_end_common.utilities.report_functions package

Submodules

spinn_front_end_common.utilities.report_functions.energy_report module

class spinn_front_end_common.utilities.report_functions.energy_report.**EnergyReport**
Bases: object

ENERGY_DETAILED_FILENAME = 'Detailed_energy_report.rpt'

ENERGY_SUMMARY_FILENAME = 'energy_summary_report.rpt'

JULES_PER_MILLISECOND_FOR_BOXED_48_CHIP_FRAME_IDLE_COST = 0.0045833333

JULES_PER_MILLISECOND_FOR_FRAME_IDLE_COST = 0.117

JULES_PER_MILLISECOND_PER_CHIP_ACTIVE_OVERHEAD = 0.0006399999999999999

JULES_PER_MILLISECOND_PER_FPGA = 0.000584635

JULES_PER_MILLISECOND_PER_FRAME_ACTIVE_COST = 0.154163558

JULES_PER_MILLISECOND_PER_IDLE_CHIP = 0.00036

JULES_PER_MILLISECOND_PER_UNBOXED_48_CHIP_FRAME_IDLE_COST = 0.01666667

JULES_PER_SPIKE = 8e-10

JULES_TO_KILLIWATT_HOURS = 3600000

N_MONITORS_ACTIVE_DURING_COMMS = 2

spinn_front_end_common.utilities.report_functions.fixed_route_from_machine_report module

class spinn_front_end_common.utilities.report_functions.fixed_route_from_machine_report.**Fixed**
Bases: object

function for writing the fixed routes from the machine

spinn_front_end_common.utilities.report_functions.memory_map_on_chip_report module

class spinn_front_end_common.utilities.report_functions.memory_map_on_chip_report.**MemoryMapOn**
Bases: object

Report on memory usage

spinn_front_end_common.utilities.report_functions.memory_map_on_host_chip_report module

class spinn_front_end_common.utilities.report_functions.memory_map_on_host_chip_report.**Memory**
Bases: object

Report on memory usage

spinn_front_end_common.utilities.report_functions.memory_map_on_host_report module

class spinn_front_end_common.utilities.report_functions.memory_map_on_host_report.**MemoryMapOn**
Bases: object

Report on memory usage

spinn_front_end_common.utilities.report_functions.routing_table_from_machine_report module

class spinn_front_end_common.utilities.report_functions.routing_table_from_machine_report.**Rou**
Bases: object

Module contents

class spinn_front_end_common.utilities.report_functions.**MemoryMapOnChipReport**
Bases: object

Report on memory usage

class spinn_front_end_common.utilities.report_functions.**MemoryMapOnHostChipReport**
Bases: object

Report on memory usage

class spinn_front_end_common.utilities.report_functions.**MemoryMapOnHostReport**
Bases: object

Report on memory usage

class spinn_front_end_common.utilities.report_functions.**RoutingTableFromMachineReport**
Bases: object

spinn_front_end_common.utilities.scp package

Submodules

spinn_front_end_common.utilities.scp.clear_iobuf_process module

class spinn_front_end_common.utilities.scp.clear_iobuf_process.**ClearIOBUFProcess**(*connection_selec*
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

clear_iobuf(*core_subsets*, *n_cores*)

receive_response(*response*)

spinn_front_end_common.utilities.scp.scp_clear_iobuf_request module

class spinn_front_end_common.utilities.scp.scp_clear_iobuf_request.**SCPClearIOBUFRequest**(*x*, *y*, *p*, *destination_p*, *expect_re*

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

get_scp_response()

spinn_front_end_common.utilities.scp.scp_update_runtime_request module

class spinn_front_end_common.utilities.scp.scp_update_runtime_request.**SCPUpdateRuntimeRequest**

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

get_scp_response()

spinn_front_end_common.utilities.scp.update_runtime_process module

class spinn_front_end_common.utilities.scp.update_runtime_process.**UpdateRuntimeProcess**(*connecti*

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

receive_response(*response*)

update_runtime(*run_time*, *infinite_run*, *core_subsets*, *n_cores*)

Module contents

class spinn_front_end_common.utilities.scp.**ClearIOBUFProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

clear_iobuf(*core_subsets*, *n_cores*)

receive_response(*response*)

class spinn_front_end_common.utilities.scp.**SCPClearIOBUFRequest**(*x*, *y*, *p*, *destination_port*, *expect_response=True*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

get_scp_response()

class spinn_front_end_common.utilities.scp.**SCPUpdateRuntimeRequest**(*x*, *y*, *p*, *run_time*, *infinite_run*, *destination_port*, *expect_response=True*)

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

get_scp_response()

class spinn_front_end_common.utilities.scp.**UpdateRuntimeProcess**(*connection_selector*)

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

receive_response(*response*)

update_runtime(*run_time*, *infinite_run*, *core_subsets*, *n_cores*)

spinn_front_end_common.utilities.utility_objs package

Subpackages

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages package

Submodules

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.get_reinjection_status_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.get_reinjectio

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to get the status of the dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the dropped packet reinjector, between 0 and 17

- **command_code** – the command code used by the extra monitor vertex for getting reinjection status

get_scp_response()

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.get_reinjectio

Bases: *spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse*

An SCP response to a request for the dropped packet reinjection status

read_data_bytestring(*data*, *offset*)

See spinnman.messages.scp.abstract_scp_response.AbstractSCPResponse.read_data_bytestring()

reinjection_functionality_status

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.reset_counters_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.reset_counters

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to reset the statistics counters of the dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the dropped packet reinjector, between 0 and 17
- **command_code** – the command code used by the extra monitor vertex for resetting reinjection counters.

get_scp_response()

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_reinjection_packet_types_messa module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_reinjectio

Bases: *spinnman.messages.scp.abstract_messages.scp_request.*

AbstractSCPRequest

An SCP Request to set the dropped packet reinjected packet types

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the dropped packet reinjector, between 0 and 17
- **point_to_point** – bool stating if point to point should be set
- **multicast** – bool stating if multicast should be set
- **nearest_neighbour** – bool stating if nearest neighbour should be set
- **fixed_route** – bool stating if fixed route should be set
- **command_code** – the code used by the extra monitor vertex for set packet types

get_scp_response()

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_emergency_timeout_mes module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_eme

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to set the router emergency timeout for dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the dropped packet reinjector, between 0 and 17
- **timeout_mantissa** (*int*) – The mantissa of the timeout value, between 0 and 15
- **timeout_exponent** – The exponent of the timeout value, between 0 and 15
- **command_code** – the code used by the extra monitor vertex for setting the emergency timeout value

get_scp_response()

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_timeout_message module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_tim

Bases: *spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest*

An SCP Request to the extra monitor core to set the router timeout for dropped packet reinjection

Parameters

- **x** (*int*) – The x-coordinate of a chip, between 0 and 255
- **y** (*int*) – The y-coordinate of a chip, between 0 and 255
- **p** (*int*) – The processor running the dropped packet reinjector, between 0 and 17
- **timeout_mantissa** (*int*) – The mantissa of the timeout value, between 0 and 15
- **timeout_exponent** – The exponent of the timeout value, between 0 and 15
- **command_code** – the code used by the extra monitor vertex for setting the emergency timeout value

get_scp_response()

Module contents

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes package

Submodules

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.read_status_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.read_status_p

Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

get_reinjection_status(*x*, *y*, *p*, *command_code*)

get_reinjection_status_for_core_subsets(*core_subsets*, *command_code*)

handle_reinjection_status_response(*response*)

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.reset_counters_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.reset_counter
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

reset_counters(*core_subsets*, *command_code*)

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_packet_types_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_packet_ty
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

set_packet_types(*core_subsets*, *point_to_point*, *multicast*, *nearest_neighbour*, *fixed_route*, *command_code*)

Parameters

- **core_subsets** – sets of cores to send command to
- **point_to_point** – bool stating if point to point should be set
- **multicast** – bool stating if multicast should be set
- **nearest_neighbour** – bool stating if nearest neighbour should be set
- **fixed_route** – bool stating if fixed route should be set

Return type None

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_emergency_timeout_proc module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_em
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

set_timeout(*mantissa*, *exponent*, *core_subsets*, *command_code*)

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_timeout_process module

class spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_ti
Bases: *spinnman.processes.abstract_multi_connection_process.AbstractMultiConnectionProcess*

set_timeout(*mantissa*, *exponent*, *core_subsets*, *command_code*)

Module contents

Submodules

spinn_front_end_common.utilities.utility_objs.dpri_flags module

class `spinn_front_end_common.utilities.utility_objs.dpri_flags.`**`DPRIFlags`**(*value*, *doc=''*)

Bases: `enum.Enum`

SCP Dropped Packet Reinjection Packet type flags

`FIXED_ROUTE` = **8**

`MULTICAST` = **1**

`NEAREST_NEIGHBOUR` = **4**

`POINT_TO_POINT` = **2**

spinn_front_end_common.utilities.utility_objs.executable_finder module

class `spinn_front_end_common.utilities.utility_objs.executable_finder.`**`ExecutableFinder`**(*binary_s*
in-
clude_co

Bases: *`spinn_utilities.executable_finder.ExecutableFinder`*

Manages a set of folders in which to search for binaries, and allows for binaries to be discovered within this path. This adds a default location to look to the base class.

Parameters

- **`binary_search_paths`** (*`iterable of str`*) – The initial set of folders to search for binaries.
- **`include_common_binaries_folder`** (*`bool`*) – If True (i.e. the default), the spinn_front_end_common.common_model_binaries folder is searched for binaries. If you are not using the common models, or the model binary names conflict with your own, this parameter can be used to avoid this issue. Note that the folder will be appended to the value of binary_search_paths, so the common binary search path will be looked in last.

spinn_front_end_common.utilities.utility_objs.executable_type module

class `spinn_front_end_common.utilities.utility_objs.executable_type.`**`ExecutableType`**(*value*, *start_state*, *end_state*, *sup-*
ports_auto_pa
doc='')

Bases: `enum.Enum`

supports starting of different types of executables

`NO_APPLICATION` = **3**

`RUNNING` = **0**

`SYNC` = **1**

SYSTEM = 4

USES_SIMULATION_INTERFACE = 2

spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters module

class spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.**LivePacketG**

Bases: object

parameter holder for LPG's so that they can be instantiated at a later date.

board_address

hostname

key_prefix

message_type

number_of_packets_sent_per_time_step

partition_id

payload_as_time_stamps

payload_prefix

payload_right_shift

port

prefix_type

right_shift

strip_sdp

tag

use_payload_prefix

use_prefix

spinn_front_end_common.utilities.utility_objs.provenance_data_item module

class spinn_front_end_common.utilities.utility_objs.provenance_data_item.**ProvenanceDataItem**(*n v r p m s*)

Bases: object

Container for provenance data

Parameters

- **names** – A list of strings representing the hierarchy of naming of this item
- **value** – The value of the item
- **report** – True if the item should be reported to the user
- **message** – The message to send to the end user if report is True

message
The message to report to the end user, or None if no message

names
The hierarchy of names of this bit of provenance data

report
True if this provenance data entry needs reporting to the end user

value
The value of the item

spinn_front_end_common.utilities.utility_objs.re_injection_status module

class spinn_front_end_common.utilities.utility_objs.re_injection_status.**ReInjectionStatus**(*data offset*)

Bases: object

Represents a status information from dropped packet reinjection

Parameters

- **data** (*str*) – The data containing the information
- **offset** (*int*) – The offset in the data where the information starts

is_reinjecting_fixed_route
True if re injection of fixed-route packets is enabled

is_reinjecting_multicast
True if re injection of multicast packets is enabled

is_reinjecting_nearest_neighbour
True if re injection of nearest neighbour packets is enabled

is_reinjecting_point_to_point
True if re injection of point-to-point packets is enabled

n_dropped_packet_overflows
Of the n_dropped_packets received, how many were lost due to not having enough space in the queue of packets to reinject

n_dropped_packets
The number of packets dropped by the router and received by the reinjector (may not fit in the queue though)

n_link_dumps
The number of times that when a dropped packet was caused due to a link failing to take the packet.

Returns int

n_missed_dropped_packets
The number of times that when a dropped packet was read it was found that another one or more packets had also been dropped, but had been missed

n_processor_dumps
The number of times that when a dropped packet was caused due to a processor failing to take the packet.

Returns int

n_reinjected_packets
Of the n_dropped_packets received, how many packets were successfully re injected

router_emergency_timeout
The WAIT2 timeout value of the router in cycles

router_timeout
The WAIT1 timeout value of the router in cycles

Module contents

class spinn_front_end_common.utilities.utility_objs.**ExecutableFinder**(*binary_search_paths=None*, *include_common_binaries_folder=Tr*

Bases: *spinn_utilities.executable_finder.ExecutableFinder*

Manages a set of folders in which to search for binaries, and allows for binaries to be discovered within this path. This adds a default location to look to the base class.

Parameters

- **binary_search_paths** (*iterable of str*) – The initial set of folders to search for binaries.
- **include_common_binaries_folder** (*bool*) – If True (i.e. the default), the spinn_front_end_common.common_model_binaries folder is searched for binaries. If you are not using the common models, or the model binary names conflict with your own, this parameter can be used to avoid this issue. Note that the folder will be appended to the value of binary_search_paths, so the common binary search path will be looked in last.

class spinn_front_end_common.utilities.utility_objs.**ExecutableType**(*value*, *start_state*, *end_state*, *supports_auto_pause_and_resume*, *doc=''*)

Bases: enum.Enum

supports starting of different types of executables

NO_APPLICATION = 3

RUNNING = 0

SYNC = 1

SYSTEM = 4

USES_SIMULATION_INTERFACE = 2

class spinn_front_end_common.utilities.utility_objs.**LivePacketGatherParameters**(*port*, *hostname*, *tag*, *board_address*, *strip_sdp*, *use_prefix*, *key_prefix*, *prefix_type*, *message_type*, *right_shift*, *payload_as_time_stamp*, *use_payload_prefix*, *payload_prefix*, *payload_right_shift*, *number_of_packets_sent*, *partition_id*)

Bases: object

parameter holder for LPG's so that they can be instantiated at a later date.

board_address

hostname

key_prefix

message_type

number_of_packets_sent_per_time_step

partition_id

payload_as_time_stamps

payload_prefix

payload_right_shift

port

prefix_type

right_shift

strip_sdp

tag

use_payload_prefix

use_prefix

class spinn_front_end_common.utilities.utility_objs.**ProvenanceDataItem**(*names, value, report=False, message=None*)

Bases: object

Container for provenance data

Parameters

- **names** – A list of strings representing the hierarchy of naming of this item
- **value** – The value of the item
- **report** – True if the item should be reported to the user
- **message** – The message to send to the end user if report is True

message
The message to report to the end user, or None if no message

names
The hierarchy of names of this bit of provenance data

report
True if this provenance data entry needs reporting to the end user

value
The value of the item

Submodules

spinn_front_end_common.utilities.constants module

class spinn_front_end_common.utilities.constants.**BUFFERING_OPERATIONS**

Bases: enum.Enum

BUFFER_READ = 0

BUFFER_WRITE = 1

class spinn_front_end_common.utilities.constants.**SDP_PORTS**

Bases: enum.Enum

EXTRA_MONITOR_CORE_RE_INJECTION = 4

INPUT_BUFFERING_SDP_PORT = 1

OUTPUT_BUFFERING_SDP_PORT = 2

RUNNING_COMMAND_SDP_PORT = 3

spinn_front_end_common.utilities.constants.**SDP_RUNNING_MESSAGE_CODES**
alias of SDP_RUNNING_MESSAGE_ID_CODES

spinn_front_end_common.utilities.exceptions module

exception spinn_front_end_common.utilities.exceptions.**BufferableRegionTooSmall**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when the SDRAM space of the region for buffered packets is too small to contain any packet at all

exception spinn_front_end_common.utilities.exceptions.**BufferedRegionNotPresent**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when trying to issue buffered packets for a region not managed

exception spinn_front_end_common.utilities.exceptions.**ConfigurationException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when the front end determines a input param is invalid

exception spinn_front_end_common.utilities.exceptions.**ExecutableFailedToStartException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when an executable has not entered the expected state during start up

exception spinn_front_end_common.utilities.exceptions.**ExecutableFailedToStopException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when an executable has not entered the expected state during execution

exception spinn_front_end_common.utilities.exceptions.**ExecutableNotFoundException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when a specified executable could not be found

exception spinn_front_end_common.utilities.exceptions.**RallocException**
Bases: *spinn_front_end_common.utilities.exceptions.SpinnFrontEndException*

Raised when there are not enough routing table entries

exception spinn_front_end_common.utilities.exceptions.**SpinnFrontEndException**
Bases: exceptions.Exception

Raised when the front end detects an error

spinn_front_end_common.utilities.failed_state module

class spinn_front_end_common.utilities.failed_state.**FailedState**
Bases: *spinn_front_end_common.utilities.simulator_interface.SimulatorInterface*

static add_socket_address(*x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run(*run_time*)

stop()

tags

transceiver

use_virtual_board

verify_not_running()

spinn_front_end_common.utilities.function_list module

spinn_front_end_common.utilities.function_list.**get_front_end_common_pacman_xml_paths**()
Get the XML path for the front end common interface functions

spinn_front_end_common.utilities.globals_variables module

spinn_front_end_common.utilities.globals_variables.**get_not_running_simulator**()

spinn_front_end_common.utilities.globals_variables.**get_simulator**()

spinn_front_end_common.utilities.globals_variables.**has_simulator**()

spinn_front_end_common.utilities.globals_variables.**set_failed_state**(*new_failed_state*)

spinn_front_end_common.utilities.globals_variables.**set_simulator**(*new_simulator*)

spinn_front_end_common.utilities.globals_variables.**unset_simulator**()

spinn_front_end_common.utilities.helpful_functions module

spinn_front_end_common.utilities.helpful_functions.**convert_string_into_chip_and_core_subset**
Translate a string list of cores into a core subset

Parameters **cores** (*str or None*) – string representing down cores formatted as x,y,p[:x,y,p]*

spinn_front_end_common.utilities.helpful_functions.**convert_time_diff_to_total_milliseconds**
converts between a time diff and total milliseconds

Returns total milliseconds

`spinn_front_end_common.utilities.helpful_functions.`**`determine_flow_states`**(*executable_types*, *no_sync_changes*)

returns the start and end states for these executable types

Parameters

- **executable_types** – the execute types to locate start and end states from
- **no_sync_changes** – the number of times sync signals been sent

Returns dict of executable type to states.

`spinn_front_end_common.utilities.helpful_functions.`**`generate_unique_folder_name`**(*folder*, *filename*, *extension*)

Generate a unique file name with a given extension in a given folder

Parameters

- **folder** – where to put this unique file
- **filename** – the name of the first part of the file without extension
- **extension** – extension of the file

Returns file path with a unique addition

`spinn_front_end_common.utilities.helpful_functions.`**`get_ethernet_chip`**(*machine*, *board_address*)

locate the chip with the given board IP address

Parameters

- **machine** – the spinnaker machine
- **board_address** – the board address to locate the chip of.

Returns The chip that supports that board address

Raises ***ConfigurationException*** – when that board address has no chip associated with it

`spinn_front_end_common.utilities.helpful_functions.`**`locate_memory_region_for_placement`**(*placem*, *region*, *transc*

Get the address of a region for a placement

Parameters

- **region** (*int*) – the region to locate the base address of
- **placement** (`pacman.model.placements.Placement`) – the placement object to get the region address of
- **transceiver** (*spiNNMan.transciever.Transciever*) – the python interface to the spinnaker machine

`spinn_front_end_common.utilities.helpful_functions.`**`read_config`**(*config*, *section*, *item*)

Get the string value of a config item, returning None if the value is "None"

spinn_front_end_common.utilities.helpful_functions.**read_config_boolean**(*config*, *section*, *item*)

Get the boolean value of a config item, returning None if the value is "None"

spinn_front_end_common.utilities.helpful_functions.**read_config_int**(*config*, *section*, *item*)

Get the integer value of a config item, returning None if the value is "None"

spinn_front_end_common.utilities.helpful_functions.**read_data**(*x*, *y*, *address*, *length*, *data_format*, *transceiver*)

Reads and converts a single data item from memory

Parameters

- **x** – chip x
- **y** – chip y
- **address** – base address of the SDRAM chip to read
- **length** – length to read
- **data_format** – the format to read memory
- **transceiver** – the SpinnMan interface

spinn_front_end_common.utilities.helpful_functions.**set_up_output_application_data_specifics**

Parameters

- **where_to_write_application_data_files** – the location where all app data is by default written to
- **max_application_binaries_kept** – The max number of report folders to keep active at any one time
- **n_calls_to_run** – the counter of how many times run has been called.
- **this_run_time_string** – the time stamp string for this run

Returns the run folder for this simulation to hold app data

spinn_front_end_common.utilities.helpful_functions.**set_up_report_specifics**(*default_report_file_pat*, *max_reports_kept*, *n_calls_to_run*, *this_run_time_string=*

Parameters

- **default_report_file_path** – The location where all reports reside
- **max_reports_kept** – The max number of report folders to keep active at any one time
- **n_calls_to_run** – the counter of how many times run has been called.
- **this_run_time_string** – holder for the timestamp for future runs

Returns The folder for this run, the time_stamp

spinn_front_end_common.utilities.helpful_functions.**sort_out_downed_chips_cores_links**(*downed_*
*downed_*
*downed_*

Translate the down cores and down chips string into a form that spinnman can understand

Parameters

- **downed_cores** (*str or None*) – string representing down cores formatted as x,y,p[:x,y,p]*
- **downed_chips** (*str or None*) – string representing down chips formatted as x,y[:x,y]*
- **downed_links** – string representing down links formatted as x,y,link[:x,y,link]*

Returns a tuple of (set of (x, y) of down chips, set of (x, y, p) of down cores, set of ((x, y), link id) of down links)

Return type ({(int, int,), }, {(int, int, int), }, {((int, int), int), })

spinn_front_end_common.utilities.helpful_functions.**translate_iobuf_extraction_elements**(*hard_*
*hard_*
ex-
e-
cutab
ex-
e-
cutab

Parameters

- **hard_coded_cores** – list of cores to read iobuf from
- **hard_coded_model_binary** – list of binary names to read iobuf from
- **executable_targets** – the targets of cores and executable binaries
- **executable_finder** – where to find binaries paths from binary names

Returns core subsets for the cores to read iobuf from

spinn_front_end_common.utilities.helpful_functions.**write_finished_file**(*app_data_runtime_folder*, *re-*
port_default_directory)

spinn_front_end_common.utilities.math_constants module

random math constants

spinn_front_end_common.utilities.simulator_interface module

class spinn_front_end_common.utilities.simulator_interface.**SimulatorInterface**

Bases: object

add_socket_address(*x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run

stop()

tags

transceiver

use_virtual_board

verify_not_running()

Module contents

class spinn_front_end_common.utilities.**FailedState**

Bases: *spinn_front_end_common.utilities.simulator_interface.SimulatorInterface*

static add_socket_address(*x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run(*run_time*)

stop()

tags

transceiver

use_virtual_board

verify_not_running()

class spinn_front_end_common.utilities.**SimulatorInterface**

Bases: object

add_socket_address(*x*)

buffer_manager

config

graph_mapper

has_ran

increment_none_labelled_vertex_count

machine

machine_time_step

no_machine_time_steps

none_labelled_vertex_count

placements

run

stop()

tags

transceiver

use_virtual_board

verify_not_running()

spinn_front_end_common.utility_models package

Submodules

spinn_front_end_common.utility_models.chip_power_monitor_application_vertex module

class spinn_front_end_common.utility_models.chip_power_monitor_application_vertex.**ChipPowerMo**

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

class for representing idle time recording code in a application graph.

chip power monitor application vertex constructor

Parameters

- **label** – vertex label
- **constraints** – constraints for the vertex
- **n_samples_per_recording** – how many samples to take before / recording to sdram the total

- **sampling_frequency** – how many microseconds between sampling

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex module

class spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.**ChipPowerMonito**

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*,

spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost

machine vertex for c code representing functionality to record / idle times in a machine graph

class CHIP_POWER_MONITOR_REGIONS

Bases: `enum.Enum`

CONFIG = 1

RECORDING = 2

SYSTEM = 0

CONFIG_SIZE_IN_BYTES = 8

DEFAULT_MALLOCS_USED = 3

MAX_BUFFER_SIZE = 1048576

MAX_CORES_PER_CHIP = 18

RECORDING_SIZE_PER_ENTRY = 72

SAMPLE_RECORDING_REGION = 0

static binary_file_name()

returns the string binary file name

Returns basestring

static binary_start_type()

static method to allow app verts to use this

Returns starttype enum

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_minimum_buffer_sdram_usage(*obj*, **args*, ***kwargs*)

Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*obj*, **args*, ***kwargs*)

Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes

- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_data(*placement*, *buffer_manager*)
get data from sdram given placement and buffer manager

Parameters

- **placement** – the location on machien to get data from
- **buffer_manager** – the buffer manager that might have data

Returns results

Return type numpy array with 1 dimension

get_recorded_region_ids()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

static get_resources(*n_machine_time_steps*, *time_step*, *time_scale_factor*, *n_samples_per_recording*, *sampling_frequency*)
get resources used by this vertex

:return:Resource container

n_samples_per_recording

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

sampling_frequency

static sdram_calculation()
calculates the sdram requirements of the vertex

Returns int

spinn_front_end_common.utility_models.command_sender module

class spinn_front_end_common.utility_models.command_sender.**CommandSender**(*label*, *constraints*)

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*,

spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*

A utility for sending commands to a vertex (possibly an external device) at fixed times in the simulation

add_commands(*start_resume_commands*, *pause_stop_commands*, *timed_commands*, *vertex_to_send_to*)

Add commands to be sent down a given edge

Parameters

- **start_resume_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send when the simulation starts or resumes from pause
- **pause_stop_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – the commands to send when the simulation stops or pauses after running
- **timed_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send at specific times
- **vertex_to_send_to** – The vertex these commands are to be sent to

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

dependent_vertices()

Return the vertices which this vertex depends upon

edges_and_partitions()

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str Return a string representation of the models binary

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_outgoing_partition_constraints (*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms (*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

spinn_front_end_common.utility_models.command_sender_machine_vertex module

class spinn_front_end_common.utility_models.command_sender_machine_vertex.**CommandSenderMachin**

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*

BINARY_FILE_NAME = 'command_sender_multicast_source.aplx'

class DATA_REGIONS
Bases: enum.Enum

COMMANDS_AT_START_RESUME = 3

COMMANDS_AT_STOP_PAUSE = 4

COMMANDS_WITH_ARBITRARY_TIMES = 2

PROVENANCE_REGION = 5

SETUP = 1

SYSTEM_REGION = 0

TOTAL_REQUIRED_MALLOCS = 5

generate_data_specification(*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *n_machine_time_steps*)

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

Return a string representation of the models binary

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

static **get_n_command_bytes**(*commands*)

static **get_number_of_mallocs_used_by_dsg**()

static **get_timed_commands_bytes**(*timed_commands*)

resources_required

The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

spinn_front_end_common.utility_models.extra_monitor_support_application_vertex module

class `spinn_front_end_common.utility_models.extra_monitor_support_application_vertex.`**ExtraMor**

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(*spec*, *placement*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex module

class spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.**ExtraMonitor**

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*

constructor

Parameters

- **constraints** – constraints on this vertex
- **reinject_multicast** – if we reinject mc packets
- **reinject_point_to_point** – if we reinject point to point packets
- **reinject_nearest_neighbour** – if we reinject nearest neighbour packets
- **reinject_fixed_route** – if we reinject fixed route packets

generate_data_specification(*spec*, *placement*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_reinjection_status(*placements*, *transceiver*)
gets the reinjection status from this extra monitor vertex

Parameters

- **transceiver** – the spinnMan interface
- **placements** – the placements object

Returns the reinjection status for this vertex

get_reinjection_status_for_vertices(*placements*, *extra_monitor_cores_for_data*, *transceiver*)
gets the reinjection status from a set of extra monitor cores

Parameters

- **placements** – the placements object
- **extra_monitor_cores_for_data** – the extra monitor cores to get status from
- **transceiver** – the spinnMan interface

Return type None

reinject_fixed_route

reinject_multicast

reinject_nearest_neighbour

reinject_point_to_point

reset_reinjection_counters(*transceiver*, *placements*, *extra_monitor_cores_to_set*)
Resets the counters for re injection

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

set_reinjection_packets(*placements*, *transceiver*, *point_to_point=None*, *multicast=None*, *nearest_neighbour=None*, *fixed_route=None*)

Parameters

- **placements** – placements object
- **transceiver** – spinnman instance
- **point_to_point** – bool stating if point to point should be set, or None if left as before

- **multicast** – bool stating if multicast should be set, or None if left as before
- **nearest_neighbour** – bool stating if nearest neighbour should be set, or None if left as before
- **fixed_route** – bool stating if fixed route should be set, or None if left as before.

Return type None

set_reinjection_router_emergency_timeout(*timeout_mantissa*, *timeout_exponent*, *transceiver*, *placements*, *extra_monitor_cores_to_set*)

Sets the timeout of the routers

Parameters

- **timeout_mantissa** (*int*) – The mantissa of the timeout value, between 0 and 15
- **timeout_exponent** (*int*) – The exponent of the timeout value, between 0 and 15
- **transceiver** – the spinnMan instance
- **placements** – the placements object
- **extra_monitor_cores_to_set** – the set of vertices to change the local chip for.

set_router_time_outs(*timeout_mantissa*, *timeout_exponent*, *transceiver*, *placements*, *extra_monitor_cores_to_set*)

supports setting of the router time outs for a set of chips via their extra monitor cores.

Parameters

- **timeout_mantissa** (*int*) – what timeout mantissa to set it to
- **timeout_exponent** (*int*) – what timeout exponent to set it to
- **transceiver** – the spinnman interface
- **placements** – placements object
- **extra_monitor_cores_to_set** – which vertices to use

Return type None

static static_get_binary_file_name()

static static_get_binary_start_type()

static static_resources_required()

spinn_front_end_common.utility_models.live_packet_gather module

class spinn_front_end_common.utility_models.live_packet_gather.**LivePacketGather**(*hostname*, *port*, *board_address=No*, *tag=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEI*, *2>*, *right_shift=0*, *payload_as_time_stan*, *use_payload_prefi*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_se*, *constraints=None*, *label=None*)

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *pacman.model.graphs.application.application_vertex.ApplicationVertex*

A model which stores all the events it receives during a timer tick and then compresses them into Ethernet packets and sends them out of a spinnaker machine.

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(*spec*, *placement*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (`pacman.model.graph.slice.Slice`) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

spinn_front_end_common.utility_models.live_packet_gather_machine_vertex module

class spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.**LivePacketGathe**

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection*

N_ADDITIONAL_PROVENANCE_ITEMS = 2

TRAFFIC_IDENTIFIER = 'LPG_EVENT_STREAM'

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

static **get_cpu_usage**()
Get the CPU used by this vertex

Returns 0

Return type int

static **get_dtcm_usage**()
Get the DTCM used by this vertex

get_provenance_data_from_machine(*transceiver*, *placement*)
Get provenance from the machine

Parameters

- **transceiver** – spinnman interface to the machine
- **placement** – the location of this vertex on the machine

static **get_sdram_usage**()
Get the SDRAM used by this vertex

is_in_injection_mode
Whether this vertex is actually in injection mode.

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

spinn_front_end_common.utility_models.multi_cast_command module

class spinn_front_end_common.utility_models.multi_cast_command.**MultiCastCommand**(*key*, *payload=None*, *time=None*, *repeat=0*, *delay_between_repea*

Bases: `object`

A command to be sent to a vertex

Parameters

- **key** (*int*) – The key of the command
- **payload** (*int*) – The payload of the command
- **time** (*int*) – The time within the simulation at which to send the command, or None if this is not a timed command
- **repeat** (*int*) – The number of times that the command should be repeated after sending it once. This could be used to ensure that the command is sent despite lost packets. Must be between 0 and 65535

- **delay_between_repeats** (*int*) – The amount of time in micro seconds to wait between sending repeats of the same command. Must be between 0 and 65535, and must be 0 if repeat is 0

Raises *SpynnakerException* – If the repeat or delay are out of range

delay_between_repeats

is_payload

Determine if this command has a payload. By default, this returns True if the payload passed in to the constructor is not None, but this can be overridden to indicate that a payload will be generated, despite None being passed to the constructor

Returns True if there is a payload, False otherwise

Return type bool

is_timed

key

payload

Get the payload of the command.

Returns The payload of the command, or None if there is no payload

Return type int

repeat

time

spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source module

class spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.**ReverseIpTagMult**

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons* *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

A model which will allow events to be injected into a spinnaker machine and converted into multicast packets.

Parameters

- **n_keys** – The number of keys to be sent via this multicast source
- **label** – The label of this vertex
- **constraints** – Any initial constraints to this vertex
- **board_address** – The IP address of the board on which to place this vertex if receiving data, either buffered or live (by default, any board is chosen)
- **receive_port** – The port on the board that will listen for incoming event packets (default is to disable this feature; set a value to enable it)

- **receive_sdp_port** – The SDP port to listen on for incoming event packets (defaults to 1)
- **receive_tag** – The IP tag to use for receiving live events (uses any by default)
- **receive_rate** – The estimated rate of packets that will be sent by this source
- **virtual_key** – The base multicast key to send received events with (assigned automatically by default)
- **prefix** – The prefix to "or" with generated multicast keys (default is no prefix)
- **prefix_type** – Whether the prefix should apply to the upper or lower half of the multicast keys (default is upper half)
- **check_keys** – True if the keys of received events should be verified before sending (default False)
- **send_buffer_times** – An array of arrays of times at which keys should be sent (one array for each key, default disabled)
- **send_buffer_partition_id** – The id of the partition containing the edges down which the events are to be sent
- **send_buffer_max_space** – The maximum amount of space to use of the SDRAM on the machine (default is 1MB)
- **send_buffer_space_before_notify** – The amount of space free in the sending buffer before the machine will ask the host for more data (default setting is optimised for most cases)
- **buffer_notification_ip_address** – The IP address of the host that will send new buffers (must be specified if a send buffer is specified or if recording will be used)
- **buffer_notification_port** – The port that the host that will send new buffers is listening on (must be specified if a send buffer is specified, or if recording will be used)
- **buffer_notification_tag** – The IP tag to use to notify the host about space in the buffer (default is to use any tag)

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

enable_recording(*record_buffer_size=1048576*, *buffer_size_before_receive=16384*, *time_between_requests=0*)

generate_data_specification(*spec*, *placement*)

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

send_buffer_times

spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex module

class spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.**Re**

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_host_pre_buffered_impl.SendsBuffersFromHostPreBufferedImpl*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*

A model which allows events to be injected into spinnaker and converted in to multicast packets

enable_recording(*record_buffer_size=1048576*, *buffer_size_before_receive=16384*, *time_between_triggers=0*)
Enable recording of the keys sent

Parameters

- **record_buffer_size** (*int*) – The size of the recording buffer in bytes. Note that when using automatic pause and resume, this will be used as the minimum size of the buffer
- **buffer_size_before_receive** (*int*) – The size that the buffer can grow to before a read request is issued to the host (in bytes)
- **time_between_triggers** (*int*) – The minimum time between the sending of read requests

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

static get_cpu_usage()

static get_dtcm_usage()

get_minimum_buffer_sdram_usage()

Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)

Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_provenance_data_from_machine(*transceiver*, *placement*)

get_recorded_region_ids()

Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

static **get_sdram_usage**(*send_buffer_times*, *send_buffer_max_space*, *recording_enabled*)

is_in_injection_mode
Whether this vertex is actually in injection mode.

is_recording()
Deduce if the recorder is actually recording

mask

static **n_regions_to_allocate**(*send_buffering*, *recording*)
Get the number of regions that will be allocated

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

send_buffer_times

send_buffers

update_buffer(*obj*, *arg*)

virtual_key

Module contents

class spinn_front_end_common.utility_models.**CommandSender**(*label*, *constraints*)
Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*

A utility for sending commands to a vertex (possibly an external device) at fixed times in the simulation

add_commands(*start_resume_commands*, *pause_stop_commands*, *timed_commands*, *vertex_to_send_to*)
Add commands to be sent down a given edge

Parameters

- **start_resume_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send when the simulation starts or resumes from pause

- **pause_stop_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – the commands to send when the simulation stops or pauses after running
- **timed_commands** (iterable of *spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand*) – The commands to send at specific times
- **vertex_to_send_to** – The vertex these commands are to be sent to

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

dependent_vertices()

Return the vertices which this vertex depends upon

edges_and_partitions()

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str Return a string representation of the models binary

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (`pacman.model.graph.slice.Slice`) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises `None` – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

class spinn_front_end_common.utility_models.**CommandSenderMachineVertex**(*constraints*, *resources_required*, *label*, *commands_at_start_resume*, *commands_at_pause_stop*, *timed_commands*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*

BINARY_FILE_NAME = 'command_sender_multicast_source.aplx'

class DATA_REGIONS

Bases: `enum.Enum`

COMMANDS_AT_START_RESUME = 3

COMMANDS_AT_STOP_PAUSE = 4

COMMANDS_WITH_ARBITRARY_TIMES = 2

PROVENANCE_REGION = 5

SETUP = 1

SYSTEM_REGION = 0

TOTAL_REQUIRED_MALLOCS = 5

generate_data_specification(*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *n_machine_time_steps*)

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

Return a string representation of the models binary

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

static get_n_command_bytes(*commands*)

static **get_number_of_mallocs_used_by_dsg**()

static **get_timed_commands_bytes**(*timed_commands*)

resources_required

The resources required by the vertex

Return type pacman.model.resources.ResourceContainer

class spinn_front_end_common.utility_models.**LivePacketGather**(*hostname*, *port*, *board_address=None*, *tag=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOType.KEY_32_BIT: 2>*, *right_shift=0*, *payload_as_time_stamps=True*, *use_payload_prefix=True*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_per_time_step=0*, *constraints=None*, *label=None*)

Bases: spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification, spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary, pacman.model.graphs.application.application_vertex.ApplicationVertex

A model which stores all the events it receives during a timer tick and then compresses them into Ethernet packets and sends them out of a spinnaker machine.

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(*spec*, *placement*)

Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

class spinn_front_end_common.utility_models.**LivePacketGatherMachineVertex**(*label*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOType.KE 2>*, *right_shift=0*, *payload_as_time_stamps=True use_payload_prefix=True*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_per_ti hostname=None*, *port=None*, *strip_sdp=None*, *board_address=None*, *tag=None*, *constraints=None*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*,

spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection

N_ADDITIONAL_PROVENANCE_ITEMS = 2

TRAFFIC_IDENTIFIER = 'LPG_EVENT_STREAM'

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

static **get_cpu_usage**()

Get the CPU used by this vertex

Returns 0

Return type int

static **get_dtcm_usage**()

Get the DTCM used by this vertex

get_provenance_data_from_machine(*transceiver*, *placement*)

Get provenance from the machine

Parameters

- **transceiver** – spinnman interface to the machine
- **placement** – the location of this vertex on the machine

static **get_sdram_usage**()

Get the SDRAM used by this vertex

is_in_injection_mode

Whether this vertex is actually in injection mode.

resources_required

The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

class spinn_front_end_common.utility_models.**MultiCastCommand**(*key*, *payload=None*, *time=None*, *repeat=0*, *delay_between_repeats=0*)

Bases: object

A command to be sent to a vertex

Parameters

- **key** (*int*) – The key of the command
- **payload** (*int*) – The payload of the command
- **time** (*int*) – The time within the simulation at which to send the command, or None if this is not a timed command
- **repeat** (*int*) – The number of times that the command should be repeated after sending it once. This could be used to ensure that the command is sent despite lost packets. Must be between 0 and 65535
- **delay_between_repeats** (*int*) – The amount of time in micro seconds to wait between sending repeats of the same command. Must be between 0 and 65535, and must be 0 if repeat is 0

Raises *SpynnakerException* – If the repeat or delay are out of range

delay_between_repeats

is_payload

Determine if this command has a payload. By default, this returns True if the payload passed in to the constructor is not None, but this can be overridden to indicate that a payload will be generated, despite None being passed to the constructor

Returns True if there is a payload, False otherwise

Return type bool

is_timed

key

payload

Get the payload of the command.

Returns The payload of the command, or None if there is no payload

Return type int

repeat

time

class spinn_front_end_common.utility_models.**ReverseIpTagMultiCastSource**(*n_keys*, *label=None*, *constraints=None*, *max_atoms_per_core=922337*, *board_address=None*, *receive_port=None*, *receive_sdp_port=1*, *receive_tag=None*, *receive_rate=10*, *virtual_key=None*, *prefix=None*, *prefix_type=None*, *check_keys=False*, *send_buffer_times=None*, *send_buffer_partition_id=Non*, *send_buffer_max_space=1048*, *send_buffer_space_before_not*, *buffer_notification_ip_address*, *buffer_notification_port=None*, *buffer_notification_tag=None*, *reserve_reverse_ip_tag=False*)

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

A model which will allow events to be injected into a spinnaker machine and converted into multicast packets.

Parameters

- **n_keys** – The number of keys to be sent via this multicast source
- **label** – The label of this vertex
- **constraints** – Any initial constraints to this vertex
- **board_address** – The IP address of the board on which to place this vertex if receiving data, either buffered or live (by default, any board is chosen)
- **receive_port** – The port on the board that will listen for incoming event packets (default is to disable this feature; set a value to enable it)
- **receive_sdp_port** – The SDP port to listen on for incoming event packets (defaults to 1)

- **receive_tag** – The IP tag to use for receiving live events (uses any by default)
- **receive_rate** – The estimated rate of packets that will be sent by this source
- **virtual_key** – The base multicast key to send received events with (assigned automatically by default)
- **prefix** – The prefix to "or" with generated multicast keys (default is no prefix)
- **prefix_type** – Whether the prefix should apply to the upper or lower half of the multicast keys (default is upper half)
- **check_keys** – True if the keys of received events should be verified before sending (default False)
- **send_buffer_times** – An array of arrays of times at which keys should be sent (one array for each key, default disabled)
- **send_buffer_partition_id** – The id of the partition containing the edges down which the events are to be sent
- **send_buffer_max_space** – The maximum amount of space to use of the SDRAM on the machine (default is 1MB)
- **send_buffer_space_before_notify** – The amount of space free in the sending buffer before the machine will ask the host for more data (default setting is optimised for most cases)
- **buffer_notification_ip_address** – The IP address of the host that will send new buffers (must be specified if a send buffer is specified or if recording will be used)
- **buffer_notification_port** – The port that the host that will send new buffers is listening on (must be specified if a send buffer is specified, or if recording will be used)
- **buffer_notification_tag** – The IP tag to use to notify the host about space in the buffer (default is to use any tag)

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

enable_recording(*record_buffer_size=1048576*, *buffer_size_before_receive=16384*, *time_between_requests=0*)

generate_data_specification(*spec*, *placement*)

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms (*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

send_buffer_times

class spinn_front_end_common.utility_models.**ReverseIPTagMulticastSourceMachineVertex** (**args, **kwargs*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_host_pre_buffered_impl.SendsBuffersFromHostPreBufferedImpl*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*

A model which allows events to be injected into spinnaker and converted in to multicast packets

enable_recording (*record_buffer_size=1048576, buffer_size_before_receive=16384, time_between_triggers=0*)

Enable recording of the keys sent

Parameters

- **record_buffer_size** (*int*) – The size of the recording buffer in bytes. Note that when using automatic pause and resume, this will be used as the minimum size of the buffer
- **buffer_size_before_receive** (*int*) – The size that the buffer can grow to before a read request is issued to the host (in bytes)

- **time_between_triggers** (*int*) – The minimum time between the sending of read requests

generate_data_specification (*obj*, **args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name ()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type ()
Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

static get_cpu_usage ()

static get_dtcm_usage ()

get_minimum_buffer_sdram_usage ()
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space (*buffer_space*, *machine_time_step*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_outgoing_partition_constraints (*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_provenance_data_from_machine (*transceiver*, *placement*)

get_recorded_region_ids ()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address (*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

static get_sdram_usage(*send_buffer_times*, *send_buffer_max_space*, *recording_enabled*)

is_in_injection_mode
Whether this vertex is actually in injection mode.

is_recording()
Deduce if the recorder is actually recording

mask

static n_regions_to_allocate(*send_buffering*, *recording*)
Get the number of regions that will be allocated

resources_required
The resources required by the vertex

Return type `pacman.model.resources.ResourceContainer`

send_buffer_times

send_buffers

update_buffer(*obj*, *arg*)

virtual_key

Module contents

8.1.2 Indices and tables

- genindex
- modindex
- search

SpiNNFrontEndCommon_github

SpiNNFrontEndCommon_individual_docs

CHAPTER 9

SpiNNakerGraphFrontEnd

SpiNNakerGraphFrontEnd_github

SpiNNakerGraphFrontEnd_individual_docs

CHAPTER 10

sPyNNaker

This package provides the shared code for PyNN implementation for SpiNNaker.

10.1 sPyNNaker

10.1.1 Contents

spynnaker

spynnaker package

Subpackages

spynnaker.pyNN package

Subpackages

spynnaker.pyNN.connections package

Submodules

spynnaker.pyNN.connections.ethernet_command_connection module

class spynnaker.pyNN.connections.ethernet_command_connection.**EthernetCommandConnection**(*translato com- mand_co lo- cal_host lo- cal_port*

Bases: *spinn_front_end_common.utilities.database.database_connection.DatabaseConnection*

A connection that can send commands to a device at the start and end of a simulation

Parameters

- **translator** – A translator of multicast commands to device commands
- **command_containers** – A list of instances of AbstractSendMeMulticastCommandsVertex that have commands to be sent at the start and end of simulation
- **local_host** – The optional host to listen on for the start/resume message
- **local_port** – The optional port to listen on for the stop/pause message

add_command_container(*command_container*)

Add a command container

Parameters **command_container** – An instance of AbstractSendMeMulticastCommandsVertex that has commands to be sent at the start and end of simulation

spynnaker.pyNN.connections.ethernet_control_connection module

class spynnaker.pyNN.connections.ethernet_control_connection.**EthernetControlConnection**(*translato lo- cal_host lo- cal_port*

Bases: *spinnman.connections.udp_packet_connections.eieio_connection.EIEIOConnection*, threading.Thread

A connection that can translate Ethernet control messages received from a Population

Parameters

- **translator** – The translator of multicast to control commands
- **local_host** – The optional host to listen on
- **local_port** – The optional port to listen on

close()

Close the connection

run()

spynnaker.pyNN.connections.spynnaker_live_spikes_connection module

class spynnaker.pyNN.connections.spynnaker_live_spikes_connection.**SpynnakerLiveSpikesConnecti**

Bases: *spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection*

A connection for receiving and sending live spikes from and to SpiNNaker

Parameters

- **receive_labels** (*iterable of str*) – Labels of population from which live spikes will be received.
- **send_labels** (*iterable of str*) – Labels of population to which live spikes will be sent
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

send_spike(*label*, *neuron_id*, *send_full_keys=False*)

Send a spike from a single neuron

Parameters

- **label** (*str*) – The label of the population from which the spike will originate
- **neuron_id** (*int*) – The id of the neuron sending a spike

- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each neuron from the database, or whether to send 16-bit neuron ids directly

send_spikes(*label*, *neuron_ids*, *send_full_keys=False*)

Send a number of spikes

Parameters

- **label** (*str*) – The label of the population from which the spikes will originate
- **neuron_ids** (*[int]*) – array-like of neuron ids sending spikes
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each neuron from the database, or whether to send 16-bit neuron ids directly

Module contents

class spynnaker.pyNN.connections.**EthernetCommandConnection**(*translator*, *command_containers=None*, *local_host=None*, *local_port=19999*)

Bases: *spinn_front_end_common.utilities.database.database_connection.DatabaseConnection*

A connection that can send commands to a device at the start and end of a simulation

Parameters

- **translator** – A translator of multicast commands to device commands
- **command_containers** – A list of instances of AbstractSendMeMulticastCommandsVertex that have commands to be sent at the start and end of simulation
- **local_host** – The optional host to listen on for the start/resume message
- **local_port** – The optional port to listen on for the stop/pause message

add_command_container(*command_container*)

Add a command container

Parameters **command_container** – An instance of AbstractSendMeMulticastCommandsVertex that has commands to be sent at the start and end of simulation

class spynnaker.pyNN.connections.**EthernetControlConnection**(*translator*, *local_host=None*, *local_port=None*)

Bases: *spinnman.connections.udp_packet_connections.eieio_connection.EIEIOConnection*, threading.Thread

A connection that can translate Ethernet control messages received from a Population

Parameters

- **translator** – The translator of multicast to control commands
- **local_host** – The optional host to listen on
- **local_port** – The optional port to listen on

close()

Close the connection

run()

class spynnaker.pyNN.connections.**SpynnakerLiveSpikesConnection**(*receive_labels=None*, *send_labels=None*, *local_host=None*, *local_port=19999*, *live_packet_gather_label='LiveSpikeReceiv*

Bases: *spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection*

A connection for receiving and sending live spikes from and to SpiNNaker

Parameters

- **receive_labels** (*iterable of str*) – Labels of population from which live spikes will be received.
- **send_labels** (*iterable of str*) – Labels of population to which live spikes will be sent
- **local_host** (*str*) – Optional specification of the local hostname or ip address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

send_spike(*label*, *neuron_id*, *send_full_keys=False*)

Send a spike from a single neuron

Parameters

- **label** (*str*) – The label of the population from which the spike will originate
- **neuron_id** (*int*) – The id of the neuron sending a spike
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each neuron from the database, or whether to send 16-bit neuron ids directly

send_spikes(*label*, *neuron_ids*, *send_full_keys=False*)

Send a number of spikes

Parameters

- **label** (*str*) – The label of the population from which the spikes will originate
- **neuron_ids** (*[int]*) – array-like of neuron ids sending spikes
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each neuron from the database, or whether to send 16-bit neuron ids directly

spynnaker.pyNN.external_devices_models package

Subpackages

spynnaker.pyNN.external_devices_models.push_bot package

Subpackages

spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules package

Submodules

spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.push_bot_lif_ethernet module

class `spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.push_bot_lif_e`

Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Leaky integrate and fire neuron with an exponentially decaying current input

none_pynn_default_parameters = {'v_init': None}

spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.push_bot_lif_spinnaker_link module

class `spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.push_bot_lif_s`

Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Control module for a pushbot connected to a SpiNNaker Link

none_pynn_default_parameters = {'v_init': None}

Module contents

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.**PushBotLifEthe**

Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Leaky integrate and fire neuron with an exponentially decaying current input

none_pynn_default_parameters = {'v_init': None}

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.**PushBotLifSpir**

Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Control module for a pushbot connected to a SpiNNaker Link

none_pynn_default_parameters = {'v_init': None}

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet package

Submodules

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_dev

Bases: *spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice*

An arbitrary PushBot device

Parameters

- **protocol** – The protocol instance to get commands from
- **device** – The Enum instance of the device to control
- **uses_payload** – True if the device uses a payload for control

device_control_key
The key that must be sent to the device to control it

Return type int

device_control_max_value
The maximum value to send to the device

Return type float

device_control_min_value
The minimum value to send to the device

Return type float

device_control_partition_id
A partition ID to give to an outgoing edge partition that will control this device

Return type str

device_control_timesteps_between_sending
The number of timesteps between sending commands to the device. This defines the "sampling interval" for the device.

Return type int

device_control_uses_payload
True if the control of the device accepts an arbitrary valued payload, the value of which will change the devices behaviour

Return type bool

protocol
The protocol instance, for use in the subclass

set_command_protocol(*command_protocol*)
Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_laser_device module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_las

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl. ProvidesKeyToAtomMappingImpl*, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The Laser of a PushBot

Parameters

- **laser** – The PushBotLaser value to control

- **protocol** – The protocol instance to get commands from
- **start_active_time** – The "active time" value to send at the start
- **start_total_period** – The "total period" value to send at the start
- **start_frequency** – The "frequency" to send at the start
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_led_device module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_led

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The LED of a PushBot

Parameters

- **led** – The PushBotLED parameter to control
- **protocol** – The protocol instance to get commands from
- **start_active_time_front** – The "active time" to set for the front LED at the start
- **start_active_time_back** – The "active time" to set for the back LED at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_motor_device module

class `spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_mot`

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl. ProvidesKeyToAtomMappingImpl*, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The motor of a PushBot

Parameters

- **motor** – a PushBotMotor value to indicate the motor to control
- **protocol** – The protocol used to control the device
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_retina_device module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_ret

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice*, *spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor*

get_database_connection()
Get a Database Connection instance that this device uses to inject packets

get_injector_label()
Get the label to give to the Spike Injector

get_injector_parameters()
Get the parameters of the Spike Injector to use with this device

get_n_neurons()
Get the number of neurons that will be sent out by the device

get_translator()
Get a translator of multicast commands to Ethernet commands

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_speaker_device module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_spe

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.*

ProvidesKeyToAtomMappingImpl, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The Speaker of a PushBot

Parameters

- **speaker** – The PushBotSpeaker value to control
- **protocol** – The protocol instance to get commands from
- **start_active_time** – The "active time" to set at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start
- **start_melody** – The "melody" to set at the start
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_retina_connection module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_retina_conne

Bases: *spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpynnakerLiveSpikesConnection*

A connection that sends spikes from the PushBot retina to a spike injector in SpiNNaker. Note that this assumes a packet format of 16-bits per retina event.

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_translator module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_translator.P

Bases: *spynnaker.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator*

Translates packets between PushBot Multicast packets and PushBot WiFi Commands

Parameters

- **protocol** – The instance of the PushBot protocol to get keys from
- **pushbot_wifi_connection** – A WiFi connection to the PushBot

translate_control_packet(*multicast_packet*)

Translate a multicast packet received over Ethernet and send appropriate messages to the external device

Parameters **multicast_packet** (spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement) – A received multicast packet

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_wifi_connection module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_wifi_connect

Bases: *spinnman.connections.abstract_classes.connection.Connection*, *spinnman.connections.abstract_classes.listenable.Listenable*

A connection to a pushbot via WiFi

Parameters

- **remote_host** (*str*) – The IP address of the PushBot
- **remote_port** (*int*) – The port number of the PushBot (default 56000)

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error setting up the communication channel

close()

See spinnman.connections.Connection.close()

get_receive_method()

is_connected()

See spinnman.connections.Connection.is_connected()

is_ready_to_receive(*timeout=0*)

local_ip_address

The local IP address to which the connection is bound.

Returns The local ip address as a dotted string e.g. 0.0.0.0

Return type str

Raises **None** – No known exceptions are thrown

local_port
The local port to which the connection is bound.

Returns The local port number

Return type int

Raises **None** – No known exceptions are thrown

receive(*timeout=None*)
Receive data from the connection

Parameters **timeout** (*None*) – The timeout, or None to wait forever

Returns The data received

Return type *bytestring*

Raises

- ***SpinnmanTimeoutException*** – If a timeout occurs before any data is received
- ***SpinnmanIOException*** – If an error occurs receiving the data

remote_ip_address
The remote ip address to which the connection is connected.

Returns The remote ip address as a dotted string, or None if not connected remotely

Return type str

remote_port
The remote port to which the connection is connected.

Returns The remote port, or None if not connected remotely

Return type int

send(*data*)
Send data down this connection

Parameters **data** (bytestring) – The data to be sent

Raises ***SpinnmanIOException*** – If there is an error sending the data

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_wifi_connection

Get an existing connection to a PushBot, or make a new one

Parameters

- **remote_host** (*str*) – The IP address of the PushBot
- **remote_port** (*int*) – The port number of the PushBot (default 56000)

Module contents

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetDevice**

Bases: *spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice*

An arbitrary PushBot device

Parameters

- **protocol** – The protocol instance to get commands from
- **device** – The Enum instance of the device to control
- **uses_payload** – True if the device uses a payload for control

device_control_key
The key that must be sent to the device to control it

Return type int

device_control_max_value
The maximum value to send to the device

Return type float

device_control_min_value
The minimum value to send to the device

Return type float

device_control_partition_id
A partition ID to give to an outgoing edge partition that will control this device

Return type str

device_control_timesteps_between_sending
The number of timesteps between sending commands to the device. This defines the "sampling interval" for the device.

Return type int

device_control_uses_payload
True if the control of the device accepts an arbitrary valued payload, the value of which will change the devices behaviour

Return type bool

protocol
The protocol instance, for use in the subclass

set_command_protocol(*command_protocol*)
Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetLaserD**

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The Laser of a PushBot

Parameters

- **laser** – The PushBotLaser value to control
- **protocol** – The protocol instance to get commands from
- **start_active_time** – The "active time" value to send at the start
- **start_total_period** – The "total period" value to send at the start
- **start_frequency** – The "frequency" to send at the start
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetLEDDev**

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl. ProvidesKeyToAtomMappingImpl*, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The LED of a PushBot

Parameters

- **led** – The PushBotLED parameter to control
- **protocol** – The protocol instance to get commands from
- **start_active_time_front** – The "active time" to set for the front LED at the start
- **start_active_time_back** – The "active time" to set for the back LED at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetMotorD**

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The motor of a PushBot

Parameters

- **motor** – a PushBotMotor value to indicate the motor to control
- **protocol** – The protocol used to control the device
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetRetina**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice*, *spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor*

get_database_connection()
Get a Database Connection instance that this device uses to inject packets

get_injector_label()
Get the label to give to the Spike Injector

get_injector_parameters()
Get the parameters of the Spike Injector to use with this device

get_n_neurons()
Get the number of neurons that will be sent out by the device

get_translator()
Get a translator of multicast commands to Ethernet commands

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetSpeake**

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl. ProvidesKeyToAtomMappingImpl*, *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*

The Speaker of a PushBot

Parameters

- **speaker** – The PushBotSpeaker value to control
- **protocol** – The protocol instance to get commands from
- **start_active_time** – The "active time" to set at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start
- **start_melody** – The "melody" to set at the start
- **timesteps_between_send** – The number of timesteps between sending commands to the device, or None to use the default

pause_stop_commands
The commands needed when pausing or stopping simulation

set_command_protocol(*command_protocol*)

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotRetinaConnecti**

Bases: *spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpynnakerLiveSpikesConnection*

A connection that sends spikes from the PushBot retina to a spike injector in SpiNNaker. Note that this assumes a packet format of 16-bits per retina event.

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotTranslator**(*prot*
push
*bot_*

Bases: *spynnaker.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator*

Translates packets between PushBot Multicast packets and PushBot WiFi Commands

Parameters

- **protocol** – The instance of the PushBot protocol to get keys from
- **pushbot_wifi_connection** – A WiFi connection to the PushBot

translate_control_packet(*multicast_packet*)

Translate a multicast packet received over Ethernet and send appropriate messages to the external device

Parameters **multicast_packet** (spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement) – A received multicast packet

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**get_pushbot_wifi_connecti**

Get an existing connection to a PushBot, or make a new one

Parameters

- **remote_host** (*str*) – The IP address of the PushBot
- **remote_port** (*int*) – The port number of the PushBot (default 56000)

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotWIFIConnection**

Bases: *spinnman.connections.abstract_classes.connection.Connection*, *spinnman.connections.abstract_classes.listenable.Listenable*

A connection to a pushbot via WiFi

Parameters

- **remote_host** (*str*) – The IP address of the PushBot

- **remote_port** (*int*) – The port number of the PushBot (default 56000)

Raises ***spinnman.exceptions.SpinnmanIOException*** – If there is an error setting up the communication channel

close()
See `spinnman.connections.Connection.close()`

get_receive_method()

is_connected()
See `spinnman.connections.Connection.is_connected()`

is_ready_to_receive(*timeout=0*)

local_ip_address
The local IP address to which the connection is bound.

Returns The local ip address as a dotted string e.g. 0.0.0.0

Return type str

Raises **None** – No known exceptions are thrown

local_port
The local port to which the connection is bound.

Returns The local port number

Return type int

Raises **None** – No known exceptions are thrown

receive(*timeout=None*)
Receive data from the connection

Parameters **timeout** (*None*) – The timeout, or None to wait forever

Returns The data received

Return type *bytestring*

Raises

- ***SpinnmanTimeoutException*** – If a timeout occurs before any data is received
- ***SpinnmanIOException*** – If an error occurs receiving the data

remote_ip_address
The remote ip address to which the connection is connected.

Returns The remote ip address as a dotted string, or None if not connected remotely

Return type str

remote_port
The remote port to which the connection is connected.

Returns The remote port, or None if not connected remotely

Return type int

send(*data*)
Send data down this connection

Parameters **data** (`bytestring`) – The data to be sent

Raises ***SpinnmanIOException*** – If there is an error sending the data

spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters package

Submodules

spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_laser module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_laser.**Push**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

LASER_ACTIVE_TIME = 1

LASER_FREQUENCY = 2

LASER_TOTAL_PERIOD = 0

spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_led module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_led.**PushBo**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

LED_BACK_ACTIVE_TIME = 2

LED_FREQUENCY = 3

LED_FRONT_ACTIVE_TIME = 1

LED_TOTAL_PERIOD = 0

spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_motor module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_motor.**Push**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

MOTOR_0_LEAKY = 1

MOTOR_0_PERMANENT = 0

MOTOR_1_LEAKY = 3

MOTOR_1_PERMANENT = 2

spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_resolution module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_res

Bases: enum.Enum

DOWNSAMPLE_16_X_16 = <RetinaKey.DOWNSAMPLE_16_X_16: 268435456>

DOWNSAMPLE_32_X_32 = <RetinaKey.DOWNSAMPLE_32_X_32: 201326592>

DOWNSAMPLE_64_X_64 = <RetinaKey.DOWNSAMPLE_64_X_64: 134217728>

NATIVE_128_X_128 = <RetinaKey.NATIVE_128_X_128: 67108864>

spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_viewer module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_vie

Bases: threading.Thread

local_host

local_port

run()

spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_speaker module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_speaker.**Pu**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

SPEAKER_ACTIVE_TIME = 1

SPEAKER_MELODY = 3

SPEAKER_TONE = 2

SPEAKER_TOTAL_PERIOD = 0

Module contents

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.**PushBotLaser**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

LASER_ACTIVE_TIME = 1

LASER_FREQUENCY = 2

LASER_TOTAL_PERIOD = 0

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.**PushBotLED**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

LED_BACK_ACTIVE_TIME = 2

LED_FREQUENCY = 3

LED_FRONT_ACTIVE_TIME = 1

LED_TOTAL_PERIOD = 0

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.**PushBotMotor**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

MOTOR_0_LEAKY = 1

MOTOR_0_PERMANENT = 0

MOTOR_1_LEAKY = 3

MOTOR_1_PERMANENT = 2

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.**PushBotSpeaker**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

SPEAKER_ACTIVE_TIME = 1

SPEAKER_MELODY = 3

SPEAKER_TONE = 2

SPEAKER_TOTAL_PERIOD = 0

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.**PushBotRetinaResolu**

Bases: enum.Enum

DOWNSAMPLE_16_X_16 = <RetinaKey.DOWNSAMPLE_16_X_16: 268435456>

DOWNSAMPLE_32_X_32 = <RetinaKey.DOWNSAMPLE_32_X_32: 201326592>

DOWNSAMPLE_64_X_64 = <RetinaKey.DOWNSAMPLE_64_X_64: 134217728>

NATIVE_128_X_128 = <RetinaKey.NATIVE_128_X_128: 67108864>

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.**PushBotRetinaViewer**

Bases: threading.Thread

local_host

local_port

run()

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link package

Submodules

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_laser_de module

class `spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinna`

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_laser_device.PushBotEthernetLaserDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

The Laser of a PushBot

Parameters

- **laser** – The PushBotLaser value to control
- **protocol** – The protocol instance to get commands from
- **spinnaker_link_id** – The SpiNNakerLink that the PushBot is connected to
- **n_neurons** – The number of neurons in the device
- **label** – A label for the device
- **board_address** – The IP address of the board that the device is connected to
- **start_active_time** – The "active time" value to send at the start
- **start_total_period** – The "total period" value to send at the start
- **start_frequency** – The "frequency" to send at the start

default_parameters = {'start_total_period': 0, 'n_neurons': 1, 'label': None, 'board_address': None, 'start_frequen

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_led_devi module

class `spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinna`

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_led_device.PushBotEthernetLEDDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

The LED of a PushBot

Parameters

- **led** – The PushBotLED parameter to control
- **protocol** – The protocol instance to get commands from
- **spinnaker_link_id** – The SpiNNakerLink connected to
- **n_neurons** – The number of neurons in the device
- **label** – The label of the device
- **board_address** – The IP address of the board that the device is connected to
- **start_active_time_front** – The "active time" to set for the front LED at the start
- **start_active_time_back** – The "active time" to set for the back LED at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start

default_parameters = {'start_active_time_back': None, 'start_total_period': None, 'n_neurons': 1, 'label': None, 'b

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_motor_d module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinna

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

The motor of a PushBot

Parameters

- **motor** – a PushBotMotor value to indicate the motor to control
- **protocol** – The protocol used to control the device
- **spinnaker_link_id** – The SpiNNakerLink connected to
- **n_neurons** – The number of neurons in the device
- **label** – The label of the device
- **board_address** – The IP address of the board that the device is connected to

default_parameters = {'n_neurons': 1, 'board_address': None, 'label': None}

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_retina_d module

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinna

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

default_parameters = {'board_address': None, 'label': None}

graph_mapper(*obj*, *arg*)

routing_info(*obj*, *arg*)

start_resume_commands

The commands needed when starting or resuming simulation

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_speaker_ module

class `spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinna`

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_speaker_device.PushBotEthernetSpeakerDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

The speaker of a PushBot

Parameters

- **speaker** – The PushBotSpeaker value to control
- **protocol** – The protocol instance to get commands from
- **spinnaker_link_id** – The SpiNNakerLink connected to
- **n_neurons** – The number of neurons in the device
- **label** – The label of the device
- **board_address** – The IP address of the board that the device is connected to
- **start_active_time** – The "active time" to set at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start
- **start_melody** – The "melody" to set at the start

default_parameters = {'start_total_period': 100, 'n_neurons': 1, 'label': None, 'board_address': None, 'start_frequ

Module contents

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.**PushBotSpiNNake**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_laser_device.PushBotEthernetLaserDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

The Laser of a PushBot

Parameters

- **laser** – The PushBotLaser value to control
- **protocol** – The protocol instance to get commands from
- **spinnaker_link_id** – The SpiNNakerLink that the PushBot is connected to
- **n_neurons** – The number of neurons in the device
- **label** – A label for the device
- **board_address** – The IP address of the board that the device is connected to
- **start_active_time** – The "active time" value to send at the start
- **start_total_period** – The "total period" value to send at the start
- **start_frequency** – The "frequency" to send at the start

default_parameters = {'start_total_period': 0, 'n_neurons': 1, 'label': None, 'board_address': None, 'start_frequen

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.**PushBotSpiNNake**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_led_device.PushBotEthernetLEDDevice*, *pacman.*

model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex

The LED of a PushBot

Parameters

- **led** – The PushBotLED parameter to control
- **protocol** – The protocol instance to get commands from
- **spinnaker_link_id** – The SpiNNakerLink connected to
- **n_neurons** – The number of neurons in the device
- **label** – The label of the device
- **board_address** – The IP address of the board that the device is connected to
- **start_active_time_front** – The "active time" to set for the front LED at the start
- **start_active_time_back** – The "active time" to set for the back LED at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start

default_parameters = {'start_active_time_back': None, 'start_total_period': None, 'n_neurons': 1, 'label': None, 'b

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.**PushBotSpiNNake**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

The motor of a PushBot

Parameters

- **motor** – a PushBotMotor value to indicate the motor to control
- **protocol** – The protocol used to control the device
- **spinnaker_link_id** – The SpiNNakerLink connected to
- **n_neurons** – The number of neurons in the device
- **label** – The label of the device
- **board_address** – The IP address of the board that the device is connected to

default_parameters = {'n_neurons': 1, 'board_address': None, 'label': None}

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.**PushBotSpiNNake**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice*,

pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex

default_parameters = {'board_address': None, 'label': None}

graph_mapper(*obj*, *arg*)

routing_info(*obj*, *arg*)

start_resume_commands

The commands needed when starting or resuming simulation

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.**PushBotSpiNNake**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_speaker_device.PushBotEthernetSpeakerDevice*, *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*

The speaker of a PushBot

Parameters

- **speaker** – The PushBotSpeaker value to control
- **protocol** – The protocol instance to get commands from
- **spinnaker_link_id** – The SpiNNakerLink connected to
- **n_neurons** – The number of neurons in the device
- **label** – The label of the device
- **board_address** – The IP address of the board that the device is connected to
- **start_active_time** – The "active time" to set at the start
- **start_total_period** – The "total period" to set at the start
- **start_frequency** – The "frequency" to set at the start
- **start_melody** – The "melody" to set at the start

default_parameters = {'start_total_period': 100, 'n_neurons': 1, 'label': None, 'board_address': None, 'start_frequ

Submodules

spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device module

class spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.**Abstrac**

Bases: enum.Enum

max_value

min_value

protocol_property

time_between_send

spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device module

class spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.**Abstrac**

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

Module contents

class spynnaker.pyNN.external_devices_models.push_bot.**AbstractPushBotOutputDevice**

Bases: enum.Enum

max_value

min_value

protocol_property

time_between_send

class spynnaker.pyNN.external_devices_models.push_bot.**AbstractPushBotRetinaDevice**(*protocol*, *resolution*)

Bases: *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

Submodules

spynnaker.pyNN.external_devices_models.abstract_ethernet_controller module

class spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.**AbstractEthernetCor**
Bases: object

A controller that can send multicast packets which can be received over Ethernet and translated to control an external device

get_external_devices()
Get the external devices that are to be controlled by the controller

get_message_translator()
Get the translator of messages

Return type spynnaker_external_devices_plugin.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator

get_outgoing_partition_ids()
Get the partition ids of messages coming out of the controller

Return type list of str

spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor module

class spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor.**AbstractEthernetSensor**
Bases: object

get_database_connection()
Get a Database Connection instance that this device uses to inject packets

get_injector_label()
Get the label to give to the Spike Injector

get_injector_parameters()
Get the parameters of the Spike Injector to use with this device

get_n_neurons()
Get the number of neurons that will be sent out by the device

get_translator()
Get a translator of multicast commands to Ethernet commands

spynnaker.pyNN.external_devices_models.abstract_ethernet_translator module

class spynnaker.pyNN.external_devices_models.abstract_ethernet_translator.**AbstractEthernetTra**

Bases: object

A module that can translate packets received over Ethernet into control of an external device

translate_control_packet(*multicast_packet*)

Translate a multicast packet received over Ethernet and send appropriate messages to the external device

Parameters **multicast_packet** (spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement) – A received multicast packet

spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device module

class spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.**AbstractM**

Bases: object

A device that can be controlled by sending Multicast packets to it, either directly, or via Ethernet using an AbstractEthernetTranslator

device_control_key

The key that must be sent to the device to control it

Return type int

device_control_max_value

The maximum value to send to the device

Return type float

device_control_min_value

The minimum value to send to the device

Return type float

device_control_partition_id

A partition ID to give to an outgoing edge partition that will control this device

Return type str

device_control_timesteps_between_sending()

The number of timesteps between sending commands to the device. This defines the "sampling interval" for the device.

Return type int

device_control_uses_payload

True if the control of the device accepts an arbitrary valued payload, the value of which will change the devices behaviour

Return type bool

spynnaker.pyNN.external_devices_models.arbitrary_fpga_device module

class spynnaker.pyNN.external_devices_models.arbitrary_fpga_device.**ArbitraryFPGADevice**(*n_neuro*
fpga_lin
fpga_id,
board_a
la-
bel='Ar

Bases: *pacman.model.graphs.application.application_fpga_vertex.ApplicationFPGAVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

default_parameters = {'board_address': None, 'label': 'ArbitraryFPGADevice'}

spynnaker.pyNN.external_devices_models.external_device_lif_control module

class spynnaker.pyNN.external_devices_models.external_device_lif_control.**ExternalDeviceLifCon**

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*
spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController, *spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.AbstractVertexWithEdgeToDependentVertices*

Abstract control module for the pushbot, based on the LIF neuron, but without spikes, and using the voltage as the output to the various devices

Parameters

- **n_neurons** – The number of neurons in the population
- **devices** – The AbstractMulticastControllableDevice instances to be controlled by the population
- **create_edges** – True if edges to the devices should be added by this dev (set to False if using the dev over Ethernet using a translator)
- **translator** – Translator to be used when used for Ethernet communication. Must be provided if the dev is to be controlled over Ethernet.

default_parameters = {'tau_refrac': 0.1, 'isyn_inh': 0.0, 'tau_m': 20.0, 'tau_syn_E': 5.0, 'v_rest': 0.0, 'cm': 1.0, 'i_o

dependent_vertices()
Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)
Return the dependent edge identifiers for this vertex

get_external_devices()
Get the external devices that are to be controlled by the controller

static **get_max_atoms_per_core**()

get_message_translator()
Get the translator of messages

Return type spynnaker_external_devices_plugin.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_outgoing_partition_ids()
Get the partition ids of messages coming out of the controller

Return type list of str

none_pynn_default_parameters = {'v_init': None}

routing_key_partition_atom_mapping(*routing_info*, *partition*)

spynnaker.pyNN.external_devices_models.external_spinnaker_link_cochlea_device module

class spynnaker.pyNN.external_devices_models.external_spinnaker_link_cochlea_device.**ExternalC**

Bases: *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

default_parameters = {'board_address': None, 'label': 'ExternalCochleaDevice'}

spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device module

class spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.**Exter**

Bases: *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*, *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

Parameters

- **mode** – The retina "mode"
- **retina_key** – The value of the top 16-bits of the key
- **spinnaker_link_id** – The spinnaker link to which the retina is connected
- **polarity** – The "polarity" of the retina data
- **label** –
- **n_neurons** – The number of neurons in the population
- **board_address** –

DOWN_POLARITY = 'DOWN'

MERGED_POLARITY = 'MERGED'

MODE_128 = '128'

MODE_16 = '16'

MODE_32 = '32'

MODE_64 = '64'

UP_POLARITY = 'UP'

default_parameters = {'board_address': None, 'label': 'ExternalFPGARetinaDevice'}

static get_n_neurons(*mode*, *polarity*)

get_outgoing_partition_constraints(*partition*)

pause_stop_commands

The commands needed when pausing or stopping simulation

start_resume_commands

The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.**get_spike**

spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.**get_x_fro**

spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.**get_y_fro**

spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device module

class spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.**MunichMotorDe**

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.AbstractVertexWithEdgeToDependentVertices*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

An Omnibot motor control device - has a real vertex and an external device vertex

PARAMS_REGION = 1

PARAMS_SIZE = 28

SYSTEM_REGION = 0

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)
Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

default_parameters = {'update_time': 512, 'board_address': None, 'delay_time': 5, 'sample_time': 4096, 'speed': 3

dependent_vertices()

Return the vertices which this vertex depends upon Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)

Return the dependent edge identifiers for this vertex Return the dependent edge identifier

generate_application_data_specification(*spec*, *placement*, *graph_mapper*, *application_graph*, *machine_graph*, *routing_info*, *iptags*, *reverse_iptags*, *machine_time_step*, *time_scale_factor*)

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

reserve_memory_regions(*spec*)
Reserve SDRAM space for memory areas: 1) Area for information on what data to record 2) area for start commands 3) area for end commands

spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device module

class spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.**MunichRetina**

Bases: *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*, *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

DOWN_POLARITY = 'DOWN'

LEFT_RETINA = 'LEFT'

LEFT_RETINA_DISABLE = 69

LEFT_RETINA_ENABLE = 69

LEFT_RETINA_KEY_SET = 67

MANAGEMENT_BIT = 1024

MANAGEMENT_MASK = 4294965248

MERGED_POLARITY = 'MERGED'

RIGHT_RETINA = 'RIGHT'

RIGHT_RETINA_DISABLE = 70

RIGHT_RETINA_ENABLE = 70

RIGHT_RETINA_KEY_SET = 68

UP_POLARITY = 'UP'

default_parameters = {'polarity': None, 'board_address': None, 'label': 'MunichRetinaDevice'}

get_outgoing_partition_constraints(*partition*)

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.**get_spike_value_**

spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.**get_x_from_robot**

spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.**get_y_from_robot**

spynnaker.pyNN.external_devices_models.threshold_type_multicast_device_control module

class spynnaker.pyNN.external_devices_models.threshold_type_multicast_device_control.**Threshol**
Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*

A threshold type that can send multicast keys with the value of membrane voltage as the payload

get_n_cpu_cycles_per_neuron()

get_n_threshold_parameters()

get_threshold_parameter_types()

get_threshold_parameters()

Module contents

class spynnaker.pyNN.external_devices_models.**AbstractEthernetController**
Bases: object

A controller that can send multicast packets which can be received over Ethernet and translated to control an external device

get_external_devices()
Get the external devices that are to be controlled by the controller

get_message_translator()
Get the translator of messages

Return type spynnaker_external_devices_plugin.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator

get_outgoing_partition_ids()
Get the partition ids of messages coming out of the controller

Return type list of str

class spynnaker.pyNN.external_devices_models.**AbstractEthernetSensor**
Bases: object

get_database_connection()
Get a Database Connection instance that this device uses to inject packets

get_injector_label()
Get the label to give to the Spike Injector

get_injector_parameters()
Get the parameters of the Spike Injector to use with this device

get_n_neurons()
Get the number of neurons that will be sent out by the device

get_translator()
Get a translator of multicast commands to Ethernet commands

class spynnaker.pyNN.external_devices_models.**AbstractEthernetTranslator**
Bases: object

A module that can translate packets received over Ethernet into control of an external device

translate_control_packet(*multicast_packet*)
Translate a multicast packet received over Ethernet and send appropriate messages to the external device

Parameters **multicast_packet** (spinnman.messages.eieio.data_messages.abstract_eieio_data_element.AbstractEIEIODataElement) – A received multicast packet

class spynnaker.pyNN.external_devices_models.**ArbitraryFPGADevice**(*n_neurons*, *fpga_link_id*, *fpga_id*, *board_address=None*, *label='ArbitraryFPGADevice'*)
Bases: *pacman.model.graphs.application.application_fpga_vertex.ApplicationFPGAVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

default_parameters = {'board_address': None, 'label': 'ArbitraryFPGADevice'}

class spynnaker.pyNN.external_devices_models.**AbstractMulticastControllableDevice**
Bases: object

A device that can be controlled by sending Multicast packets to it, either directly, or via Ethernet using an AbstractEthernetTranslator

device_control_key
The key that must be sent to the device to control it

Return type int

device_control_max_value
The maximum value to send to the device

Return type float

device_control_min_value
The minimum value to send to the device

Return type float

device_control_partition_id
A partition ID to give to an outgoing edge partition that will control this device

Return type str

device_control_timesteps_between_sending()
The number of timesteps between sending commands to the device. This defines the "sampling interval" for the device.

Return type int

device_control_uses_payload
True if the control of the device accepts an arbitrary valued payload, the value of which will change the devices behaviour

Return type bool

class spynnaker.pyNN.external_devices_models.**ExternalDeviceLifControl**(*n_neurons*, *devices*, *create_edges*, *translator=None*, *spikes_per_second=None*, *label=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=0.0*, *v_reset=0.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*, *isyn_inh=0.0*, *isyn_exc=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*
spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController, *spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.AbstractVertexWithEdgeToDependentVertices*

Abstract control module for the pushbot, based on the LIF neuron, but without spikes, and using the voltage as the output to the various devices

Parameters

- **n_neurons** – The number of neurons in the population
- **devices** – The AbstractMulticastControllableDevice instances to be controlled by the population
- **create_edges** – True if edges to the devices should be added by this dev (set to False if using the dev over Ethernet using a translator)
- **translator** – Translator to be used when used for Ethernet communication. Must be provided if the dev is to be controlled over Ethernet.

default_parameters = {'tau_refrac': 0.1, 'isyn_inh': 0.0, 'tau_m': 20.0, 'tau_syn_E': 5.0, 'v_rest': 0.0, 'cm': 1.0, 'i_o

dependent_vertices()
Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)
Return the dependent edge identifiers for this vertex

get_external_devices()
Get the external devices that are to be controlled by the controller

static get_max_atoms_per_core()

get_message_translator()
Get the translator of messages

Return type spynnaker_external_devices_plugin.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_outgoing_partition_ids()
Get the partition ids of messages coming out of the controller

Return type list of str

none_pynn_default_parameters = {'v_init': None}

routing_key_partition_atom_mapping(*routing_info*, *partition*)

class spynnaker.pyNN.external_devices_models.**ExternalCochleaDevice**(*n_neurons*, *spinnaker_link*, *label='ExternalCochleaDevice'*, *board_address=None*)

Bases: *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

default_parameters = {'board_address': None, 'label': 'ExternalCochleaDevice'}

class spynnaker.pyNN.external_devices_models.**ExternalFPGARetinaDevice**(*mode*, *retina_key*, *spinnaker_link_id*, *polarity*, *label='ExternalFPGARetinaDevice'*, *n_neurons=None*, *board_address=None*)

Bases: *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*, *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.*

abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl

Parameters

- **mode** – The retina "mode"
- **retina_key** – The value of the top 16-bits of the key
- **spinnaker_link_id** – The spinnaker link to which the retina is connected
- **polarity** – The "polarity" of the retina data
- **label** –
- **n_neurons** – The number of neurons in the population
- **board_address** –

DOWN_POLARITY = 'DOWN'

MERGED_POLARITY = 'MERGED'

MODE_128 = '128'

MODE_16 = '16'

MODE_32 = '32'

MODE_64 = '64'

UP_POLARITY = 'UP'

default_parameters = {'board_address': None, 'label': 'ExternalFPGARetinaDevice'}

static **get_n_neurons**(*mode*, *polarity*)

get_outgoing_partition_constraints(*partition*)

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spynnaker.pyNN.external_devices_models.**MunichMotorDevice**(*n_neurons*, *spinnaker_link_id*, *board_address=None*, *speed=30*, *sample_time=4096*, *update_time=512*, *delay_time=5*, *delta_threshold=23*, *continue_if_not_different=True*, *label='RobotMotorControl'*)

Bases: *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *pacman.model.graphs.application.application_vertex.ApplicationVertex*,

spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.AbstractVertexWithEdgeToDependentVertices, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

An Omnibot motor control device - has a real vertex and an external device vertex

PARAMS_REGION = 1

PARAMS_SIZE = 28

SYSTEM_REGION = 0

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

default_parameters = {'update_time': 512, 'board_address': None, 'delay_time': 5, 'sample_time': 4096, 'speed': 3

dependent_vertices()

Return the vertices which this vertex depends upon Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex(*vertex*)

Return the dependent edge identifiers for this vertex Return the dependent edge identifier

generate_application_data_specification(*spec*, *placement*, *graph_mapper*, *application_graph*, *machine_graph*, *routing_info*, *iptags*, *reverse_iptags*, *machine_time_step*, *time_scale_factor*)

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

reserve_memory_regions(*spec*)

Reserve SDRAM space for memory areas: 1) Area for information on what data to record 2) area for start commands 3) area for end commands

class spynnaker.pyNN.external_devices_models.**MunichRetinaDevice**(*retina_key*, *spinnaker_link_id*, *position*, *label='MunichRetinaDevice'*, *n_neurons=None*, *polarity=None*, *board_address=None*)

Bases: *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*, *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

DOWN_POLARITY = 'DOWN'

LEFT_RETINA = 'LEFT'

LEFT_RETINA_DISABLE = 69

LEFT_RETINA_ENABLE = 69

LEFT_RETINA_KEY_SET = 67

MANAGEMENT_BIT = 1024

MANAGEMENT_MASK = 4294965248

MERGED_POLARITY = 'MERGED'

RIGHT_RETINA = 'RIGHT'

RIGHT_RETINA_DISABLE = 70

RIGHT_RETINA_ENABLE = 70

RIGHT_RETINA_KEY_SET = 68

UP_POLARITY = 'UP'

default_parameters = {'polarity': None, 'board_address': None, 'label': 'MunichRetinaDevice'}

get_outgoing_partition_constraints(*partition*)

pause_stop_commands
The commands needed when pausing or stopping simulation

start_resume_commands
The commands needed when starting or resuming simulation

timed_commands
The commands to be sent at given times in the simulation

class spynnaker.pyNN.external_devices_models.**ThresholdTypeMulticastDeviceControl**(*devices*)
Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*

A threshold type that can send multicast keys with the value of membrane voltage as the payload

get_n_cpu_cycles_per_neuron()

get_n_threshold_parameters()

get_threshold_parameter_types()

get_threshold_parameters()

spynnaker.pyNN.model_binaries package

Module contents

This module contains no python code.

spynnaker.pyNN.models package

Subpackages

spynnaker.pyNN.models.abstract_models package

Submodules

spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses module

class spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.**AbstractAccept**
Bases: object

Indicates an object that can be a post-vertex in a PyNN projection

add_pre_run_connection_holder(*connection_holder*, *projection_edge*, *synapse_information*)
Add a connection holder to the vertex to be filled in when the connections are actually generated

clear_connection_cache()
clears the connection data stored in the vertex so far.

get_connections_from_machine(*transceiver*, *placement*, *edge*, *graph_mapper*, *routing_infos*, *synapse_information*, *machine_time_step*)
Get the connections from the machine post-run

get_maximum_delay_supported_in_ms(*machine_time_step*)
Get the maximum delay supported by this vertex

set_synapse_dynamics(*synapse_dynamics*)
Set the synapse dynamics of this vertex

synapse_type
The synapse type of the vertex

Return type *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*

spynnaker.pyNN.models.abstract_models.abstract_contains_units module

class spynnaker.pyNN.models.abstract_models.abstract_contains_units.**AbstractContainsUnits**
Bases: object

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

spynnaker.pyNN.models.abstract_models.abstract_filterable_edge module

class spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.**AbstractFilterableEdge**
Bases: object

An edge that can be filtered

filter_edge(*graph_mapper*)
method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

spynnaker.pyNN.models.abstract_models.abstract_population_initializable module

class spynnaker.pyNN.models.abstract_models.abstract_population_initializable.**AbstractPopulat**
Bases: object

Indicates that this object has properties that can be initialised by a PyNN Population

initialize(*variable*, *value*)
Set the initial value of one of the state variables of the neurons in this population.

spynnaker.pyNN.models.abstract_models.abstract_population_settable module

class spynnaker.pyNN.models.abstract_models.abstract_population_settable.**AbstractPopulationSe**
Bases: object

Indicates that some properties of this object can be accessed from the PyNN population set and get methods

get_value(*key*)
Get a property

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set module

class spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set.**AbstractReadP**
Bases: object

A vertex whose parameters must be read before any can be set

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)
Read the parameters from the machine before any are changed

Parameters

- **transceiver** – the SpinnMan interface
- **placement** – the placement of a vertex
- **vertex_slice** – the slice of atoms for this vertex

spynnaker.pyNN.models.abstract_models.abstract_weight_updatable module

class spynnaker.pyNN.models.abstract_models.abstract_weight_updatable.**AbstractWeightUpdatable**
Bases: object

An object the weight of which can be updated

update_weight(*graph_mapper*)
Update the weight

Module contents

class spynnaker.pyNN.models.abstract_models.**AbstractAcceptsIncomingSynapses**
Bases: object

Indicates an object that can be a post-vertex in a PyNN projection

add_pre_run_connection_holder(*connection_holder*, *projection_edge*, *synapse_information*)
Add a connection holder to the vertex to be filled in when the connections are actually generated

clear_connection_cache()
clears the connection data stored in the vertex so far.

get_connections_from_machine(*transceiver*, *placement*, *edge*, *graph_mapper*, *routing_infos*, *synapse_information*, *machine_time_step*)
Get the connections from the machine post-run

get_maximum_delay_supported_in_ms(*machine_time_step*)
Get the maximum delay supported by this vertex

set_synapse_dynamics(*synapse_dynamics*)
Set the synapse dynamics of this vertex

synapse_type
The synapse type of the vertex

Return type *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*

class spynnaker.pyNN.models.abstract_models.**AbstractFilterableEdge**
Bases: object

An edge that can be filtered

filter_edge(*graph_mapper*)
method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

class spynnaker.pyNN.models.abstract_models.**AbstractPopulationInitializable**
Bases: object

Indicates that this object has properties that can be initialised by a PyNN Population

initialize(*variable*, *value*)
Set the initial value of one of the state variables of the neurons in this population.

class spynnaker.pyNN.models.abstract_models.**AbstractPopulationSettable**
Bases: object

Indicates that some properties of this object can be accessed from the PyNN population set and get methods

get_value(*key*)
Get a property

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

class spynnaker.pyNN.models.abstract_models.**AbstractReadParametersBeforeSet**
Bases: object

A vertex whose parameters must be read before any can be set

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)
Read the parameters from the machine before any are changed

Parameters

- **transceiver** – the SpinnMan interface
- **placement** – the placement of a vertex

- **vertex_slice** – the slice of atoms for this vertex

class spynnaker.pyNN.models.abstract_models.**AbstractWeightUpdatable**
Bases: object

An object the weight of which can be updated

update_weight(*graph_mapper*)
Update the weight

class spynnaker.pyNN.models.abstract_models.**AbstractContainsUnits**
Bases: object

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

spynnaker.pyNN.models.common package

Submodules

spynnaker.pyNN.models.common.abstract_neuron_recordable module

class spynnaker.pyNN.models.common.abstract_neuron_recordable.**AbstractNeuronRecordable**
Bases: object

Indicates that membrane voltage can be recorded from this object

clear_recording(*variable*, *buffer_manager*, *placements*, *graph_mapper*)
clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

get_data(*variable*, *n_machine_time_steps*, *placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)

Parameters

- **variable** –
- **n_machine_time_steps** –
- **placements** –
- **graph_mapper** –
- **buffer_manager** –
- **machine_time_step** –

Returns

get_recordable_variables()
Returns a list of the variables this models is expected to collect

is_recording(*variable*)
Determines if variable is being recorded

Returns True if vavriable are being recorded, False otherwise

Return type bool

set_recording(*variable*, *new_state=True*)
Sets v to being recorded

spynnaker.pyNN.models.common.abstract_spike_recordable module

class spynnaker.pyNN.models.common.abstract_spike_recordable.**AbstractSpikeRecordable**
Bases: object

Indicates that spikes can be recorded from this object

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)
clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes()
Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

set_recording_spikes(*new_state=True*)
Sets spikes to being recorded

spynnaker.pyNN.models.common.abstract_uint32_recorder module

class spynnaker.pyNN.models.common.abstract_uint32_recorder.**AbstractUInt32Recorder**
Bases: object

N_BYTES_PER_NEURON = 4

N_CPU_CYCLES_PER_NEURON = 4

static **get_data**(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *machine_time_step*, *variable*, *n_machine_time_steps*)

method for reading a uint32 mapped to time and neuron ids from the SpiNNaker machine

Parameters

- **label** – vertex label
- **buffer_manager** – the manager for buffered data
- **region** – the dsg region id used for this data
- **placements** – the placements object
- **graph_mapper** – the mapping between application and machine vertices
- **application_vertex** –
- **machine_time_step** –
- **variable** –
- **n_machine_time_steps** –

Returns

spynnaker.pyNN.models.common.eieio_spike_recorder module

class spynnaker.pyNN.models.common.eieio_spike_recorder.**EIEIOSpikeRecorder**

Bases: object

Records spikes using EIEIO format

get_dtcm_usage_in_bytes()

get_n_cpu_cycles(*n_neurons*)

get_spikes(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *base_key_function*, *machine_time_step*)

record

spynnaker.pyNN.models.common.gsyn_inhibitory_recorder module

class spynnaker.pyNN.models.common.gsyn_inhibitory_recorder.**GsynInhibitoryRecorder**

Bases: *spynnaker.pyNN.models.common.abstract_uint32_recorder.AbstractUInt32Recorder*

get_dtcm_usage_in_bytes()

get_gsyn_inhibitory(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *machine_time_step*)

get_n_cpu_cycles(*n_neurons*)

get_sdram_usage_in_bytes(*n_neurons*, *n_machine_time_steps*)

record_gsyn_inhibitory

spynnaker.pyNN.models.common.multi_spike_recorder module

class spynnaker.pyNN.models.common.multi_spike_recorder.**MultiSpikeRecorder**

Bases: object

get_dtcm_usage_in_bytes()

get_n_cpu_cycles(*n_neurons*)

get_sdram_usage_in_bytes(*n_neurons*, *spikes_per_timestep*, *n_machine_time_steps*)

get_spikes(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *machine_time_step*)

record

spynnaker.pyNN.models.common.neuron_recorder module

class spynnaker.pyNN.models.common.neuron_recorder.**NeuronRecorder**(*allowed_variables*)

Bases: *spynnaker.pyNN.models.common.abstract_uint32_recorder.AbstractUInt32Recorder*

get_dtcm_usage_in_bytes()

get_n_cpu_cycles(*n_neurons*)

get_recordable_variables()

get_sdram_usage_in_bytes(*variable*, *n_neurons*, *n_machine_time_steps*)

is_recording(*variable*)

recording_variables

set_recording(*variable*, *new_state*)

spynnaker.pyNN.models.common.recording_utils module

spynnaker.pyNN.models.common.recording_utils.**get_buffer_sizes**(*buffer_max*, *space_needed*, *enable_buffered_recording*)

spynnaker.pyNN.models.common.recording_utils.**get_data**(*transceiver*, *placement*, *region*, *region_size*)

Get the recorded data from a region

spynnaker.pyNN.models.common.recording_utils.**get_recording_region_size_in_bytes**(*n_machine_tim*
bytes_per_time

Get the size of a recording region in bytes

spynnaker.pyNN.models.common.recording_utils.**needs_buffering**(*buffer_max*, *space_needed*, *enable_buffered_recording*)

spynnaker.pyNN.models.common.recording_utils.**pull_off_cached_lists**(*no_loads*, *cache_file*)

Extracts numpy based data from a file

Parameters

- **no_loads** – the number of numpy elements in the file

- **cache_file** – the file to extract from

Returns The extracted data

spynnaker.pyNN.models.common.simple_population_settable module

class spynnaker.pyNN.models.common.simple_population_settable.**SimplePopulationSettable**
Bases: *spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable*

An object all of whose properties can be accessed from a PyNN Population i.e. no properties are hidden

get_value(*key*)
Get a property

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

spynnaker.pyNN.models.common.spike_recorder module

class spynnaker.pyNN.models.common.spike_recorder.**SpikeRecorder**
Bases: object

get_dtcm_usage_in_bytes()

get_n_cpu_cycles(*n_neurons*)

get_sdram_usage_in_bytes(*n_neurons*, *n_machine_time_steps*)

get_spikes(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *machine_time_step*)

record

Module contents

class spynnaker.pyNN.models.common.**AbstractNeuronRecordable**
Bases: object

Indicates that membrane voltage can be recorded from this object

clear_recording(*variable*, *buffer_manager*, *placements*, *graph_mapper*)
clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

get_data(*variable*, *n_machine_time_steps*, *placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)

Parameters

- **variable** –
- **n_machine_time_steps** –
- **placements** –
- **graph_mapper** –
- **buffer_manager** –
- **machine_time_step** –

Returns

get_recordable_variables()

Returns a list of the variables this models is expected to collect

is_recording(*variable*)

Determines if variable is being recorded

Returns True if vavriable are being recorded, False otherwise

Return type bool

set_recording(*variable*, *new_state=True*)

Sets v to being recorded

class spynnaker.pyNN.models.common.**AbstractUInt32Recorder**

Bases: object

N_BYTES_PER_NEURON = 4

N_CPU_CYCLES_PER_NEURON = 4

static get_data(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *machine_time_step*, *variable*, *n_machine_time_steps*)

method for reading a uint32 mapped to time and neuron ids from the SpiNNaker machine

Parameters

- **label** – vertex label
- **buffer_manager** – the manager for buffered data
- **region** – the dsg region id used for this data
- **placements** – the placements object
- **graph_mapper** – the mapping between application and machine vertices
- **application_vertex** –
- **machine_time_step** –
- **variable** –
- **n_machine_time_steps** –

Returns

class spynnaker.pyNN.models.common.**AbstractSpikeRecordable**

Bases: object

Indicates that spikes can be recorded from this object

clear_spike_recording (*buffer_manager*, *placements*, *graph_mapper*)
clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

get_spikes (*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes ()
Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

set_recording_spikes (*new_state=True*)
Sets spikes to being recorded

class spynnaker.pyNN.models.common.**EIEIOSpikeRecorder**
Bases: object

Records spikes using EIEIO format

get_dtcm_usage_in_bytes ()

get_n_cpu_cycles (*n_neurons*)

get_spikes (*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *base_key_function*, *machine_time_step*)

record

class spynnaker.pyNN.models.common.**NeuronRecorder** (*allowed_variables*)
Bases: *spynnaker.pyNN.models.common.abstract_uint32_recorder.AbstractUInt32Recorder*

get_dtcm_usage_in_bytes ()

get_n_cpu_cycles (*n_neurons*)

get_recordable_variables ()

get_sdram_usage_in_bytes (*variable*, *n_neurons*, *n_machine_time_steps*)

is_recording (*variable*)

recording_variables

set_recording (*variable*, *new_state*)

class spynnaker.pyNN.models.common.**MultiSpikeRecorder**
Bases: object

get_dtcm_usage_in_bytes()

get_n_cpu_cycles(*n_neurons*)

get_sdram_usage_in_bytes(*n_neurons*, *spikes_per_timestep*, *n_machine_time_steps*)

get_spikes(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *machine_time_step*)

record

class spynnaker.pyNN.models.common.**SimplePopulationSettable**
Bases: *spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable*

An object all of whose properties can be accessed from a PyNN Population i.e. no properties are hidden

get_value(*key*)
Get a property

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

class spynnaker.pyNN.models.common.**SpikeRecorder**
Bases: object

get_dtcm_usage_in_bytes()

get_n_cpu_cycles(*n_neurons*)

get_sdram_usage_in_bytes(*n_neurons*, *n_machine_time_steps*)

get_spikes(*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *machine_time_step*)

record

spynnaker.pyNN.models.common.**get_buffer_sizes**(*buffer_max*, *space_needed*, *enable_buffered_recording*)

spynnaker.pyNN.models.common.**get_data**(*transceiver*, *placement*, *region*, *region_size*)
Get the recorded data from a region

spynnaker.pyNN.models.common.**needs_buffering**(*buffer_max*, *space_needed*, *enable_buffered_recording*)

spynnaker.pyNN.models.common.**get_recording_region_size_in_bytes**(*n_machine_time_steps*, *bytes_per_timestep*)
Get the size of a recording region in bytes

spynnaker.pyNN.models.common.**pull_off_cached_lists**(*no_loads*, *cache_file*)
Extracts numpy based data from a file

Parameters

- **no_loads** – the number of numpy elements in the file
- **cache_file** – the file to extract from

Returns The extracted data

spynnaker.pyNN.models.neural_projections package

Subpackages

spynnaker.pyNN.models.neural_projections.connectors package

Submodules

spynnaker.pyNN.models.neural_projections.connectors.abstract_connector module

class `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.`**`AbstractConnecto`**

Bases: `object`

Abstract class which PyNN Connectors extend

`NUMPY_SYNAPSES_DTYPE` = **[('source', 'uint32'), ('target', 'uint16'), ('weight', 'float64'), ('delay', 'float64'), ('synapse_t**

`create_synaptic_block`(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

Create a synaptic block from the data

`generate_on_machine`()

Determines if the connector generation is supported on the machine or if the connector must be generated on the host

`get_delay_maximum`()

Get the maximum delay specified by the user in ms, or None if unbounded

`get_delay_variance`(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the variance of the delays for this connection

`get_n_connections_from_pre_vertex_maximum`(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

Get the maximum number of connections between those from each of the neurons in the pre_vertex_slice to neurons in the post_vertex_slice, for connections with a delay between min_delay and max_delay (inclusive) if both specified (otherwise all connections)

`get_n_connections_to_post_vertex_maximum`(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the maximum number of connections between those to each of the neurons in the post_vertex_slice from neurons in the pre_vertex_slice

`get_provenance_data`()

`get_weight_maximum`(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the maximum of the weights for this connection

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the mean of the weights for this connection

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the variance of the weights for this connection

safe

set_projection_information(*pre_population*, *post_population*, *rng*, *machine_time_step*)

set_space(*space*)
allows setting of the space object after instantiation

Parameters **space** –

Returns

set_weights_and_delays(*weights*, *delays*)
sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

space

verbose

spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector module

class spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.**AllToAllConnec**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Connects all cells in the presynaptic population to all cells in the postsynaptic population

Parameters **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.

allow_self_connections

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

set_weights_and_delays(*weights*, *delays*)

sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector module

class spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_conn

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Make connections using a distribution which varies with distance.

Parameters

- **d_expression** (*string*) – the right-hand side of a valid python expression for probability, involving 'd', e.g. "exp(-abs(d))", or "d<3", that can be parsed by eval(), that computes the distance dependent distribution

- **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.
- **space** (*pyNN.Space*) – a Space object, needed if you wish to specify distance- dependent weights or delays
- **n_connections** (*int*) – The number of efferent synaptic connections per neuron.

allow_self_connections

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

d_expression

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector module

class spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.**FixedNu**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

allow_self_connections

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector module

class spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.**FixedNum**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Connects a fixed number of pre-synaptic neurons selected at random, to all post-synaptic neurons

Parameters

- **n** (*int*) – number of random pre-synaptic neurons connected to output
- **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.
- **space** (*pyNN.Space*) – a Space object, needed if you wish to specify distance- dependent weights or delays - not implemented

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector module

class spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.**FixedPr**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

For each pair of pre-post cells, the connection probability is constant.

Parameters

- **p_connect** (*float*) – a float between zero and one. Each potential connection is created with this probability.
- **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.
- **space** (*pyNN.Space*) – a Space object, needed if you wish to specify distance- dependent weights or delays - not implemented

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

spynnaker.pyNN.models.neural_projections.connectors.from_file_connector module

class spynnaker.pyNN.models.neural_projections.connectors.from_file_connector.**FromFileConnect**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector*

get_reader(*file*)

get a filereader object probably using the pynn methods

For example calling from pyNN.recording import files return files.StandardTextFile(file, mode="r") :return: A pynn StandardTextFile or similar

spynnaker.pyNN.models.neural_projections.connectors.from_list_connector module

class spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.**FromListConnect**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Make connections according to a list.

Param conn_list: a list of tuples, one tuple for each connection. Each tuple should contain:

```
(pre_idx, post_idx, weight, delay)
```

where pre_idx is the index (i.e. order in the Population, not the ID) of the presynaptic neuron, and post_idx is the index of the postsynaptic neuron.

Creates a new FromListConnector.

CONN_LIST_DTYPE = dtype([('source', '<u4'), ('target', '<u4'), ('weight', '<f8'), ('delay', '<f8')])

conn_list

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

set_weights_and_delays(*weights*, *delays*)

allows setting of the weights and delays at seperate times to the init, also sets the dtypes correctly.....

Parameters

- **weights** –
- **delays** –

Returns

spynnaker.pyNN.models.neural_projections.connectors.multapse_connector module

class spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.**MultapseConnecto**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Create a multapse connector. The size of the source and destination populations are obtained when the projection is connected. The number of synapses is specified. when instantiated, the required number of synapses is created by selecting at random from the source and target populations with replacement. Uniform selection probability is assumed.

Parameters **num_synapses** – Integer. This is the total number of synapses in the connection.

Creates a new connector.

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

set_weights_and_delays(*weights*, *delays*)
sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector module

class spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.**OneToOneConnec**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Where the pre- and postsynaptic populations have the same size, connect cell i in the presynaptic pynn_population.py to cell i in the postsynaptic pynn_population.py for all i.

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

set_weights_and_delays(*weights*, *delays*)
sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

spynnaker.pyNN.models.neural_projections.connectors.small_world_connector module

class spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.**SmallWorldCon**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Module contents

class spynnaker.pyNN.models.neural_projections.connectors.**AbstractConnector**(*safe=True*, *verbose=False*)

Bases: object

Abstract class which PyNN Connectors extend

NUMPY_SYNAPSES_DTYPE = [('source', 'uint32'), ('target', 'uint16'), ('weight', 'float64'), ('delay', 'float64'), ('synapse_t

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

Create a synaptic block from the data

generate_on_machine()

Determines if the connector generation is supported on the machine or if the connector must be generated on the host

get_delay_maximum()

Get the maximum delay specified by the user in ms, or None if unbounded

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the variance of the delays for this connection

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

Get the maximum number of connections between those from each of the neurons in the pre_vertex_slice to neurons in the post_vertex_slice, for connections with a delay between min_delay and max_delay (inclusive) if both specified (otherwise all connections)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the maximum number of connections between those to each of the neurons in the post_vertex_slice from neurons in the pre_vertex_slice

get_provenance_data()

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the maximum of the weights for this connection

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the mean of the weights for this connection

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the variance of the weights for this connection

safe

set_projection_information(*pre_population*, *post_population*, *rng*, *machine_time_step*)

set_space(*space*)

allows setting of the space object after instantiation

Parameters **space** –

Returns

set_weights_and_delays(*weights*, *delays*)

sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

space

verbose

class spynnaker.pyNN.models.neural_projections.connectors.**AllToAllConnector**(*allow_self_connections=safe=True, verbose=None*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Connects all cells in the presynaptic population to all cells in the postsynaptic population

Parameters **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.

allow_self_connections

create_synaptic_block(*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, min_delay=None, max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice*)

get_weight_maximum(*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice*)

get_weight_mean(*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice*)

get_weight_variance(*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice*)

set_weights_and_delays(*weights, delays*)

sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

class spynnaker.pyNN.models.neural_projections.connectors.**DistanceDependentProbabilityConnect**

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Make connections using a distribution which varies with distance.

Parameters

- **d_expression** (*string*) – the right-hand side of a valid python expression for probability, involving 'd', e.g. "exp(-abs(d))", or "d<3", that can be parsed by eval(), that computes the distance dependent distribution
- **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.
- **space** (*pyNN.Space*) – a Space object, needed if you wish to specify distance- dependent weights or delays
- **n_connections** (*int*) – The number of efferent synaptic connections per neuron.

allow_self_connections

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

d_expression

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

class spynnaker.pyNN.models.neural_projections.connectors.**FixedNumberPostConnector**(*n*, *allow_self_conn safe=True*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

allow_self_connections

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

class spynnaker.pyNN.models.neural_projections.connectors.**FixedNumberPreConnector**(*n*, *allow_self_connec safe=True*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Connects a fixed number of pre-synaptic neurons selected at random, to all post-synaptic neurons

Parameters

- **n** (*int*) – number of random pre-synaptic neurons connected to output
- **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.

- **space** (*pyNN.Space*) – a Space object, needed if you wish to specify distance- dependent weights or delays - not implemented

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

class spynnaker.pyNN.models.neural_projections.connectors.**FixedProbabilityConnector**(*p_connect*, *allow_self_con safe=True*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

For each pair of pre-post cells, the connection probability is constant.

Parameters

- **p_connect** (*float*) – a float between zero and one. Each potential connection is created with this probability.
- **allow_self_connections** (*bool*) – if the connector is used to connect a Population to itself, this flag determines whether a neuron is allowed to connect to itself, or only to other neurons in the Population.
- **space** (*pyNN.Space*) – a Space object, needed if you wish to specify distance- dependent weights or delays - not implemented

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

class spynnaker.pyNN.models.neural_projections.connectors.**FromFileConnector**(*file*, *distributed=False*, *safe=True*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector*

get_reader(*file*)

get a filereader object probably using the pynn methods

For example calling from pyNN.recording import files return files.StandardTextFile(file, mode="r") :return: A pynn StandardTextFile or similar

class spynnaker.pyNN.models.neural_projections.connectors.**FromListConnector**(*conn_list*, *safe=True*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Make connections according to a list.

Param conn_list: a list of tuples, one tuple for each connection. Each tuple should contain:

```
(pre_idx, post_idx, weight, delay)
```

where pre_idx is the index (i.e. order in the Population, not the ID) of the presynaptic neuron, and post_idx is the index of the postsynaptic neuron.

Creates a new FromListConnector.

CONN_LIST_DTYPE = dtype([('source', '<u4'), ('target', '<u4'), ('weight', '<f8'), ('delay', '<f8')])

conn_list

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

set_weights_and_delays(*weights*, *delays*)

allows setting of the weights and delays at seperate times to the init, also sets the dtypes correctly.....

Parameters

- **weights** –
- **delays** –

Returns

class spynnaker.pyNN.models.neural_projections.connectors.**MultapseConnector**(*num_synapses*, *safe=True*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Create a multapse connector. The size of the source and destination populations are obtained when the projection is connected. The number of synapses is specified. when instantiated, the required number of synapses is created by selecting at random from the source and target populations with replacement. Uniform selection probability is assumed.

Parameters **num_synapses** – Integer. This is the total number of synapses in the connection.

Creates a new connector.

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

set_weights_and_delays(*weights*, *delays*)

sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

class spynnaker.pyNN.models.neural_projections.connectors.**OneToOneConnector**(*random_number_class*, *safe=True*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Where the pre- and postsynaptic populations have the same size, connect cell i in the presynaptic pynn_population.py to cell i in the postsynaptic pynn_population.py for all i.

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

set_weights_and_delays(*weights*, *delays*)

sets the weights and delays as needed

Parameters

- **weights** (*float*) – may either be a float, a !RandomDistribution object, a list 1D array with at least as many items as connections to be created, or a distance dependence as per a d_expression. Units nA.
- **delays** (*float*) – – as *weights*. If *None*, all synaptic delays will be set to the global minimum delay.

Raises

- **Exception** – when not a standard interface of list, scaler, or random number generator
- **NotImplementedError** – when lists are not supported and entered

class spynnaker.pyNN.models.neural_projections.connectors.**SmallWorldConnector**(*degree*, *rewiring*, *allow_self_connections=False*, *safe=True*, *verbose=False*, *n_connections=None*)

Bases: spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector

create_synaptic_block(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

generate_on_machine()

get_delay_maximum()

get_delay_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_n_connections_from_pre_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *min_delay=None*, *max_delay=None*)

get_n_connections_to_post_vertex_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_maximum(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Submodules

spynnaker.pyNN.models.neural_projections.delay_afferent_application_edge module

class spynnaker.pyNN.models.neural_projections.delay_afferent_application_edge.**DelayAfferentA**

Bases: *pacman.model.graphs.application.application_edge.ApplicationEdge*

create_machine_edge(*pre_vertex*, *post_vertex*, *label*)

spynnaker.pyNN.models.neural_projections.delay_afferent_machine_edge module

class spynnaker.pyNN.models.neural_projections.delay_afferent_machine_edge.**DelayAfferentMachi**

Bases: *pacman.model.graphs.machine.machine_edge.MachineEdge*, *spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge*, *spynnaker.pyNN.models.abstract_models.abstract_weight_updatable.AbstractWeightUpdatable*

filter_edge(*graph_mapper*)
method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

update_weight(*graph_mapper*)
Update the weight

spynnaker.pyNN.models.neural_projections.delayed_application_edge module

class spynnaker.pyNN.models.neural_projections.delayed_application_edge.**DelayedApplicationEdg**

Bases: *pacman.model.graphs.application.application_edge.ApplicationEdge*

add_synapse_information(*synapse_information*)

create_machine_edge(*pre_vertex*, *post_vertex*, *label*)
Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the start of the edge

- **post_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the end of the edge
- **label** (*str*) – label of the edge

Returns The created machine edge

Return type pacman.model.graph.machine.machine_edge.MachineEdge

spynnaker.pyNN.models.neural_projections.delayed_machine_edge module

class spynnaker.pyNN.models.neural_projections.delayed_machine_edge.**DelayedMachineEdge**(*synapse_ pre_vert post_ver la- bel=Nor weight=*

Bases: *pacman.model.graphs.machine.machine_edge.MachineEdge*, *spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge*

filter_edge(*graph_mapper*)

method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

spynnaker.pyNN.models.neural_projections.projection_application_edge module

class spynnaker.pyNN.models.neural_projections.projection_application_edge.**ProjectionApplicat**

Bases: *pacman.model.graphs.application.application_edge.ApplicationEdge*

An edge which terminates on an AbstractPopulationVertex

add_synapse_information(*synapse_information*)

create_machine_edge(*pre_vertex*, *post_vertex*, *label*)

Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the start of the edge
- **post_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the end of the edge
- **label** (*str*) – label of the edge

Returns The created machine edge

Return type pacman.model.graph.machine.machine_edge.MachineEdge

delay_edge

n_delay_stages

synapse_information

spynnaker.pyNN.models.neural_projections.projection_machine_edge module

class spynnaker.pyNN.models.neural_projections.projection_machine_edge.**ProjectionMachineEdge**(

Bases: *pacman.model.graphs.machine.machine_edge.MachineEdge*, *spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge*, *spynnaker.pyNN.models.abstract_models.abstract_weight_updatable.AbstractWeightUpdatable*, *spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data.AbstractProvidesLocalProvenanceData*

filter_edge(*graph_mapper*)

method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

get_local_provenance_data()

Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

synapse_information

update_weight(*graph_mapper*)

Update the weight

spynnaker.pyNN.models.neural_projections.synapse_information module

class spynnaker.pyNN.models.neural_projections.synapse_information.**SynapseInformation**(*connector*
synapse_d
synapse_t

Bases: object

Contains the synapse information including the connector, synapse type and synapse dynamics

connector

index

synapse_dynamics

synapse_type

Module contents

class spynnaker.pyNN.models.neural_projections.**DelayAfferentApplicationEdge**(*prevertex*, *delayvertex*, *label=None*)

Bases: *pacman.model.graphs.application.application_edge.ApplicationEdge*

create_machine_edge(*pre_vertex*, *post_vertex*, *label*)

class spynnaker.pyNN.models.neural_projections.**DelayAfferentMachineEdge**(*pre_vertex*, *post_vertex*, *label*, *weight=1*)

Bases: *pacman.model.graphs.machine.machine_edge.MachineEdge*, *spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge*, *spynnaker.pyNN.models.abstract_models.abstract_weight_updatable.AbstractWeightUpdatable*

filter_edge(*graph_mapper*)

method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

update_weight(*graph_mapper*)

Update the weight

class spynnaker.pyNN.models.neural_projections.**DelayedApplicationEdge**(*pre_vertex*, *post_vertex*, *synapse_information*, *label=None*)

Bases: *pacman.model.graphs.application.application_edge.ApplicationEdge*

add_synapse_information(*synapse_information*)

create_machine_edge(*pre_vertex*, *post_vertex*, *label*)

Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the start of the edge
- **post_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the end of the edge
- **label** (*str*) – label of the edge

Returns The created machine edge

Return type pacman.model.graph.machine.machine_edge.MachineEdge

class spynnaker.pyNN.models.neural_projections.**DelayedMachineEdge**(*synapse_information*, *pre_vertex*, *post_vertex*, *label=None*, *weight=1*)

Bases: pacman.model.graphs.machine.machine_edge.MachineEdge, spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge

filter_edge(*graph_mapper*)

method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

class spynnaker.pyNN.models.neural_projections.**ProjectionApplicationEdge**(*pre_vertex*, *post_vertex*, *synapse_information*, *label=None*)

Bases: pacman.model.graphs.application.application_edge.ApplicationEdge

An edge which terminates on an AbstractPopulationVertex

add_synapse_information(*synapse_information*)

create_machine_edge(*pre_vertex*, *post_vertex*, *label*)

Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the start of the edge
- **post_vertex** (pacman.model.graph.machine.abstract_machine_vertex.impl.MachineVertex) – The machine vertex at the end of the edge
- **label** (*str*) – label of the edge

Returns The created machine edge

Return type pacman.model.graph.machine.machine_edge.MachineEdge

delay_edge

n_delay_stages

synapse_information

class spynnaker.pyNN.models.neural_projections.**ProjectionMachineEdge**(*synapse_information*, *pre_vertex*, *post_vertex*, *label=None*, *traffic_weight=1*)

Bases: pacman.model.graphs.machine.machine_edge.MachineEdge, spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge, spynnaker.pyNN.models.abstract_models.abstract_weight_updatable.AbstractWeightUpdatable, spinn_front_end_common.

interface.provenance.abstract_provides_local_provenance_data.AbstractProvidesLocalProvenanceData

filter_edge(*graph_mapper*)
method to allow edges to determine if a edge is filter-able

Parameters **graph_mapper** – the mapper between graphs

Return type bool

get_local_provenance_data()
Get an iterable of provenance data items

Returns iterable of ProvenanceDataItem

synapse_information

update_weight(*graph_mapper*)
Update the weight

class spynnaker.pyNN.models.neural_projections.**SynapseInformation**(*connector*, *synapse_dynamics*, *synapse_type*)

Bases: object

Contains the synapse information including the connector, synapse type and synapse dynamics

connector

index

synapse_dynamics

synapse_type

spynnaker.pyNN.models.neural_properties package

Submodules

spynnaker.pyNN.models.neural_properties.neural_parameter module

class spynnaker.pyNN.models.neural_properties.neural_parameter.**NeuronParameter**(*value*, *datatype*)

Bases: object

get_dataspec_datatype()

get_value()

Module contents

class spynnaker.pyNN.models.neural_properties.**NeuronParameter**(*value*, *datatype*)
Bases: object

get_dataspec_datatype()

get_value()

spynnaker.pyNN.models.neuron package

Subpackages

spynnaker.pyNN.models.neuron.additional_inputs package

Submodules

spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input module

class spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.**AbstractAdditi**

Bases: object

Represents a possible additional independent input for a model

get_dtcm_usage_per_neuron_in_bytes()

Get the DTCM usage of this additional input in bytes

Returns The DTCM usage

Return type int

get_n_cpu_cycles_per_neuron()

Get the number of CPU cycles executed by additional_input_get_input_value_as_current and additional_input_has_spiked

get_n_parameters()

Get the number of parameters for the additional input

Returns The number of parameters

Return type int

get_parameter_types()

Get the types of the parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_parameters()

Get the parameters for the additional input

Returns An array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_sdram_usage_per_neuron_in_bytes()

Get the SDRAM usage of this additional input in bytes

Returns The SDRAM usage

Return type int

set_parameters(*parameters*, *vertex_slice*)

Set the parameters for a given subset of neurons.

To be overridden only when there is a changing variable to extract

Parameters

- **parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive module

class spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.**AdditionalI**

Bases: *spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput*

get_dtcm_usage_per_neuron_in_bytes()

get_n_cpu_cycles_per_neuron()

get_n_parameters()

get_parameter_types()

get_parameters(*obj*, **args*, ***kwargs*)

get_sdram_usage_per_neuron_in_bytes()

i_alpha

i_ca2

set_parameters(*parameters*, *vertex_slice*)

tau_ca2

Module contents

class spynnaker.pyNN.models.neuron.additional_inputs.**AbstractAdditionalInput**

Bases: object

Represents a possible additional independent input for a model

get_dtcm_usage_per_neuron_in_bytes()

Get the DTCM usage of this additional input in bytes

Returns The DTCM usage

Return type int

get_n_cpu_cycles_per_neuron()

Get the number of CPU cycles executed by additional_input_get_input_value_as_current and additional_input_has_spiked

get_n_parameters()

Get the number of parameters for the additional input

Returns The number of parameters

Return type int

get_parameter_types()

Get the types of the parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_parameters()

Get the parameters for the additional input

Returns An array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_sdram_usage_per_neuron_in_bytes()

Get the SDRAM usage of this additional input in bytes

Returns The SDRAM usage

Return type int

set_parameters(*parameters*, *vertex_slice*)

Set the parameters for a given subset of neurons.

To be overridden only when there is a changing variable to extract

Parameters

- **parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

class spynnaker.pyNN.models.neuron.additional_inputs.**AdditionalInputCa2Adaptive**(*n_neurons*, *tau_ca2*, *i_ca2*, *i_alpha*)

Bases: *spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput*

get_dtcm_usage_per_neuron_in_bytes()

get_n_cpu_cycles_per_neuron()

get_n_parameters()

get_parameter_types()

get_parameters(*obj*, **args*, ***kwargs*)

get_sdram_usage_per_neuron_in_bytes()

i_alpha

i_ca2

set_parameters(*parameters*, *vertex_slice*)

tau_ca2

spynnaker.pyNN.models.neuron.builds package

Submodules

spynnaker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_ista module

class spynnaker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_ista.**EIFConductanceAlphaPopulat**

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Exponential integrate and fire neuron with spike triggered and sub-threshold adaptation currents (isfa, ista reps.)

static set_model_max_atoms_per_core(*new_value*)

spynnaker.pyNN.models.neuron.builds.hh_cond_exp module

class `spynnaker.pyNN.models.neuron.builds.hh_cond_exp.`**`HHCondExp`**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *gbar_K=6.0*, *cm=0.2*, *e_rev_Na=50.0*, *tau_syn_E=0.2*, *tau_syn_I=2.0*, *i_offset=0.0*, *g_leak=0.01*, *e_rev_E=0.0*, *gbar_Na=20.0*, *e_rev_leak=-65.0*, *e_rev_I=-80*, *e_rev_K=-90.0*, *v_offset=-63*, *v_init=None*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Single-compartment Hodgkin-Huxley model with exponentially decaying current input

static **`set_model_max_atoms_per_core`**(*new_value*)

spynnaker.pyNN.models.neuron.builds.if_cond_alpha module

class `spynnaker.pyNN.models.neuron.builds.if_cond_alpha.`**`IFCondAlpha`**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20*, *cm=1.0*, *e_rev_E=0.0*, *e_rev_I=-70.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=0.3*, *tau_syn_I=0.5*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*)

Bases: `spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`

Leaky integrate and fire neuron with an alpha-shaped current input

static **`set_model_max_atoms_per_core`**(*new_value*)

spynnaker.pyNN.models.neuron.builds.if_cond_exp_base module

class spynnaker.pyNN.models.neuron.builds.if_cond_exp_base.**IFCondExpBase**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *tau_refrac=0.1*, *i_offset=0*, *e_rev_E=0.0*, *e_rev_I=-70.0*, *v_init=None*, *isyn_exc=0.0*, *isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an exponentially decaying conductance input

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0, 'tau_m': 20.0,

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_model_max_atoms_per_core(*new_value=255*)

spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc module

class spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc.**IFCondExpStoc**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *tau_refrac=0.1*, *i_offset=0*, *e_rev_E=0.0*, *e_rev_I=-70.0*, *du_th=0.5*, *tau_th=20.0*, *v_init=None*, *isyn_exc=0.0*, *isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'tau_th': 20.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0,

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_max_atoms_per_core(*new_value*)

spynnaker.pyNN.models.neuron.builds.if_curr_alpha module

class spynnaker.pyNN.models.neuron.builds.if_curr_alpha.**IFCurrAlpha**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=0.5*, *tau_syn_I=0.5*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an alpha-shaped conductance input

static set_model_max_atoms_per_core(*new_value*)

spynnaker.pyNN.models.neuron.builds.if_curr_delta module

class spynnaker.pyNN.models.neuron.builds.if_curr_delta.**IFCurrDelta**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*, *isyn_exc=0.0*, *isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an instantaneous current input

default_parameters = {'tau_refrac': 0.1, 'v_thresh': -50.0, 'tau_m': 20.0, 'isyn_exc': 0.0, 'v_rest': -65.0, 'cm': 1.0, 'i

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_max_atoms_per_core(*new_value*)

spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base module

class spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base.**IFCurrDualExpBase**(*n_neurons, spikes_per_seco ring_buffer_sig in-com-ing_spike_buffe con-straints=None, la-bel=None, tau_m=20.0, cm=1.0, v_rest=-65.0, v_reset=-65.0, v_thresh=-50.0, tau_syn_E=5.0, tau_syn_E2=5.0 tau_syn_I=5.0, tau_refrac=0.1, i_offset=0, v_init=None, isyn_exc=0.0, isyn_inh=0.0, isyn_exc2=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with two exponentially decaying excitatory current inputs, and one exponentially decaying inhibitory current input

default_parameters = {'tau_refrac': 0.1, 'isyn_exc2': 0.0, 'cm': 1.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_model_max_atoms_per_core(*new_value=255*)

spynnaker.pyNN.models.neuron.builds.if_curr_exp_base module

class spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.**IFCurrExpBase**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*, *isyn_exc=0.0*, *isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an exponentially decaying current input

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0, 'tau_m': 20.0,

static get_max_atoms_per_core()

isyn_exc

isyn_inh

none_pynn_default_parameters = {'v_init': None}

static set_model_max_atoms_per_core(*new_value=255*)

spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive module

class spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive.**IFCurrExpCa2Adaptive**(*n_neur spikes_ ring_b in- com- ing_sp con- straint la- bel=N tau_m= cm=1. v_rest= 65.0, v_rese 65.0, v_thres 50.0, tau_sy tau_sy tau_re i_offse tau_ca i_ca2= i_alph v_init= isyn_e isyn_in*

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Model from Liu, Y. H., & Wang, X. J. (2001). Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron. Journal of Computational Neuroscience, 10(1), 25-45. doi:10.1023/A:1008916026143

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'i_ca2': 0.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0, 'ta

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_max_atoms_per_core(*new_value*)

spynnaker.pyNN.models.neuron.builds.if_facets_hardware1 module

class spynnaker.pyNN.models.neuron.builds.if_facets_hardware1.**IFFacetsConductancePopulation**(*n*
s
r
i
c
i
c
s
l
b
g
t
t
v
5
v
6
e
8
v
8
v

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with conductance-based synapses and fixed threshold as it is resembled by the FACETS Hardware Stage 1

static set_model_max_atoms_per_core(*new_value*)

spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base module

class spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base.**IzkCondExpBase**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=No*, *constraints=None*, *label=None*, *a=0.02*, *b=0.2*, *c=-65.0*, *d=2.0*, *i_offset=0*, *u_init=-14.0*, *v_init=-70.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *e_rev_E=0.0*, *e_rev_I=-70.0*, *isyn_exc=0*, *isyn_inh=0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

default_parameters = {'tau_syn_E': 5.0, 'tau_syn_I': 5.0, 'i_offset': 0, 'a': 0.02, 'c': -65.0, 'b': 0.2, 'e_rev_E': 0.0, 'd

static get_max_atoms_per_core()

static set_model_max_atoms_per_core(*new_value=255*)

spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base module

class spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base.**IzkCurrExpBase**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=No*, *constraints=None*, *label=None*, *a=0.02*, *b=0.2*, *c=-65.0*, *d=2.0*, *i_offset=0*, *u_init=-14.0*, *v_init=-70.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *isyn_exc=0*, *isyn_inh=0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

default_parameters = {'tau_syn_E': 5.0, 'tau_syn_I': 5.0, 'i_offset': 0, 'a': 0.02, 'c': -65.0, 'b': 0.2, 'd': 2.0, 'isyn_inh

static get_max_atoms_per_core()

static set_model_max_atoms_per_core(*new_value=255*)

Module contents

class spynnaker.pyNN.models.neuron.builds.**EIFConductanceAlphaPopulation**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=9.3667*, *cm=0.281*, *v_rest=-70.6*, *v_reset=-70.6*, *v_thresh=-50.4*, *tau_syn_E=5.0*, *tau_syn_I=0.5*, *tau_refrac=0.1*, *i_offset=0.0*, *a=4.0*, *b=0.0805*, *v_spike=-40.0*, *tau_w=144.0*, *e_rev_E=0.0*, *e_rev_I=-80.0*, *delta_T=2.0*, *v_init=None*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Exponential integrate and fire neuron with spike triggered and sub-threshold adaptation currents (isfa, ista reps.)

static set_model_max_atoms_per_core(*new_value*)

class spynnaker.pyNN.models.neuron.builds.**HHCondExp**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *gbar_K=6.0*, *cm=0.2*, *e_rev_Na=50.0*, *tau_syn_E=0.2*, *tau_syn_I=2.0*, *i_offset=0.0*, *g_leak=0.01*, *e_rev_E=0.0*, *gbar_Na=20.0*, *e_rev_leak=-65.0*, *e_rev_I=-80*, *e_rev_K=-90.0*, *v_offset=-63*, *v_init=None*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Single-compartment Hodgkin-Huxley model with exponentially decaying current input

static set_model_max_atoms_per_core(*new_value*)

class spynnaker.pyNN.models.neuron.builds.**IFCondAlpha**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20*, *cm=1.0*, *e_rev_E=0.0*, *e_rev_I=-70.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=0.3*, *tau_syn_I=0.5*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an alpha-shaped current input

static set_model_max_atoms_per_core(*new_value*)

class spynnaker.pyNN.models.neuron.builds.**IFCondExpBase**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *tau_refrac=0.1*, *i_offset=0*, *e_rev_E=0.0*, *e_rev_I=-70.0*, *v_init=None*, *isyn_exc=0.0*, *isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an exponentially decaying conductance input

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0, 'tau_m': 20.0,

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_model_max_atoms_per_core(*new_value=255*)

class spynnaker.pyNN.models.neuron.builds.**IFCurrAlpha**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=0.5*, *tau_syn_I=0.5*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an alpha-shaped conductance input

static set_model_max_atoms_per_core(*new_value*)

class spynnaker.pyNN.models.neuron.builds.**IFCurrDualExpBase**(*n_neurons, spikes_per_second=None, ring_buffer_sigma=None, incoming_spike_buffer_size=None, constraints=None, label=None, tau_m=20.0, cm=1.0, v_rest=-65.0, v_reset=-65.0, v_thresh=-50.0, tau_syn_E=5.0, tau_syn_E2=5.0, tau_syn_I=5.0, tau_refrac=0.1, i_offset=0, v_init=None, isyn_exc=0.0, isyn_inh=0.0, isyn_exc2=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with two exponentially decaying excitatory current inputs, and one exponentially decaying inhibitory current input

default_parameters = {'tau_refrac': 0.1, 'isyn_exc2': 0.0, 'cm': 1.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_model_max_atoms_per_core(*new_value=255*)

class spynnaker.pyNN.models.neuron.builds.**IFCurrExpBase**(*n_neurons, spikes_per_second=None, ring_buffer_sigma=None, incoming_spike_buffer_size=None, constraints=None, label=None, tau_m=20.0, cm=1.0, v_rest=-65.0, v_reset=-65.0, v_thresh=-50.0, tau_syn_E=5.0, tau_syn_I=5.0, tau_refrac=0.1, i_offset=0, v_init=None, isyn_exc=0.0, isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an exponentially decaying current input

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0, 'tau_m': 20.0,

static get_max_atoms_per_core()

isyn_exc

isyn_inh

none_pynn_default_parameters = {'v_init': None}

static set_model_max_atoms_per_core(*new_value=255*)

class spynnaker.pyNN.models.neuron.builds.**IFFacetsConductancePopulation**(*n_neurons, spikes_per_second=None, ring_buffer_sigma=None, incoming_spike_buffer_size=None, constraints=None, label=None, g_leak=40.0, tau_syn_E=30.0, tau_syn_I=30.0, v_thresh=-55.0, v_rest=-65.0, e_rev_I=-80, v_reset=-80.0, v_init=None*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with conductance-based synapses and fixed threshold as it is resembled by the FACETS Hardware Stage 1

static set_model_max_atoms_per_core(*new_value*)

class spynnaker.pyNN.models.neuron.builds.**IzkCondExpBase**(*n_neurons, spikes_per_second=None, ring_buffer_sigma=None, incoming_spike_buffer_size=None, constraints=None, label=None, a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, u_init=-14.0, v_init=-70.0, tau_syn_E=5.0, tau_syn_I=5.0, e_rev_E=0.0, e_rev_I=-70.0, isyn_exc=0, isyn_inh=0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

default_parameters = {'tau_syn_E': 5.0, 'tau_syn_I': 5.0, 'i_offset': 0, 'a': 0.02, 'c': -65.0, 'b': 0.2, 'e_rev_E': 0.0, 'd

static **get_max_atoms_per_core**()

static **set_model_max_atoms_per_core**(*new_value=255*)

class spynnaker.pyNN.models.neuron.builds.**IzkCurrExpBase**(*n_neurons, spikes_per_second=None, ring_buffer_sigma=None, incoming_spike_buffer_size=None, constraints=None, label=None, a=0.02, b=0.2, c=-65.0, d=2.0, i_offset=0, u_init=-14.0, v_init=-70.0, tau_syn_E=5.0, tau_syn_I=5.0, isyn_exc=0, isyn_inh=0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

default_parameters = {'tau_syn_E': 5.0, 'tau_syn_I': 5.0, 'i_offset': 0, 'a': 0.02, 'c': -65.0, 'b': 0.2, 'd': 2.0, 'isyn_inh

static **get_max_atoms_per_core**()

static **set_model_max_atoms_per_core**(*new_value=255*)

class spynnaker.pyNN.models.neuron.builds.**IFCondExpStoc**(*n_neurons, spikes_per_second=None, ring_buffer_sigma=None, incoming_spike_buffer_size=None, constraints=None, label=None, tau_m=20.0, cm=1.0, v_rest=-65.0, v_reset=-65.0, v_thresh=-50.0, tau_syn_E=5.0, tau_syn_I=5.0, tau_refrac=0.1, i_offset=0, e_rev_E=0.0, e_rev_I=-70.0, du_th=0.5, tau_th=20.0, v_init=None, isyn_exc=0.0, isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'tau_th': 20.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0,

static **get_max_atoms_per_core**()

none_pynn_default_parameters = {'v_init': None}

static **set_max_atoms_per_core**(*new_value*)

class spynnaker.pyNN.models.neuron.builds.**IFCurrDelta**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_refrac=0.1*, *i_offset=0*, *v_init=None*, *isyn_exc=0.0*, *isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Leaky integrate and fire neuron with an instantaneous current input

default_parameters = {'tau_refrac': 0.1, 'v_thresh': -50.0, 'tau_m': 20.0, 'isyn_exc': 0.0, 'v_rest': -65.0, 'cm': 1.0, 'i

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_max_atoms_per_core(*new_value*)

class spynnaker.pyNN.models.neuron.builds.**IFCurrExpCa2Adaptive**(*n_neurons*, *spikes_per_second=None*, *ring_buffer_sigma=None*, *incoming_spike_buffer_size=None*, *constraints=None*, *label=None*, *tau_m=20.0*, *cm=1.0*, *v_rest=-65.0*, *v_reset=-65.0*, *v_thresh=-50.0*, *tau_syn_E=5.0*, *tau_syn_I=5.0*, *tau_refrac=0.1*, *i_offset=0*, *tau_ca2=50.0*, *i_ca2=0.0*, *i_alpha=0.1*, *v_init=None*, *isyn_exc=0.0*, *isyn_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex*

Model from Liu, Y. H., & Wang, X. J. (2001). Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron. Journal of Computational Neuroscience, 10(1), 25-45. doi:10.1023/A:1008916026143

default_parameters = {'tau_refrac': 0.1, 'cm': 1.0, 'i_ca2': 0.0, 'tau_syn_E': 5.0, 'v_rest': -65.0, 'tau_syn_I': 5.0, 'ta

static get_max_atoms_per_core()

none_pynn_default_parameters = {'v_init': None}

static set_max_atoms_per_core(*new_value*)

spynnaker.pyNN.models.neuron.input_types package

Submodules

spynnaker.pyNN.models.neuron.input_types.abstract_input_type module

class spynnaker.pyNN.models.neuron.input_types.abstract_input_type.**AbstractInputType**

Bases: object

Represents a possible input type for a neuron model (e.g. current)

get_dtcm_usage_per_neuron_in_bytes()

Get the DTCM usage of this input type in bytes

Returns The DTCM usage

Return type int

get_global_weight_scale()

Get the global weight scaling value

Returns The global weight scaling value

Return type float

get_input_type_parameter_types()

Get the types of the input type parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_input_type_parameters()

Get the parameters for the input type

Returns An array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_n_cpu_cycles_per_neuron(*n_synapse_types*)

Get the number of CPU cycles executed by input_type_get_input_value once per synapse, input_type_convert_excitatory_input_to_current and input_type_convert_inhibitory_input_to_current, per neuron

get_n_input_type_parameters()

Get the number of parameters for the input type

Returns The number of parameters

Return type int

get_sdram_usage_per_neuron_in_bytes()

Get the SDRAM usage of this input type in bytes

Returns The SDRAM usage

Return type int

set_input_type_parameters(*parameters*, *vertex_slice*)

Sets the input type parameters. Override if there are any variables that change.

Parameters

- **parameters** – the parameter values in a list of numpy arrays, ordered the same as get_input_type_parameters
- **vertex_slice** – The neurons to which the parameters apply

spynnaker.pyNN.models.neuron.input_types.input_type_conductance module

class spynnaker.pyNN.models.neuron.input_types.input_type_conductance.**InputTypeConductance**(*n_*
*e_*
*e_*

Bases: *spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

The conductance input type

e_rev_E

e_rev_I

get_global_weight_scale()

get_input_type_parameter_types()

get_input_type_parameters()

get_n_cpu_cycles_per_neuron(*n_synapse_types*)

get_n_input_type_parameters()

get_units(*variable*)

get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

spynnaker.pyNN.models.neuron.input_types.input_type_current module

class spynnaker.pyNN.models.neuron.input_types.input_type_current.**InputTypeCurrent**

Bases: *spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

The current input type

get_global_weight_scale()

get_input_type_parameter_types()

get_input_type_parameters()

get_n_cpu_cycles_per_neuron(*n_synapse_types*)

get_n_input_type_parameters()

get_units(*variable*)

get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

Module contents

class `spynnaker.pyNN.models.neuron.input_types.`**`AbstractInputType`**

Bases: `object`

Represents a possible input type for a neuron model (e.g. current)

`get_dtcm_usage_per_neuron_in_bytes`()

Get the DTCM usage of this input type in bytes

Returns The DTCM usage

Return type int

`get_global_weight_scale`()

Get the global weight scaling value

Returns The global weight scaling value

Return type float

`get_input_type_parameter_types`()

Get the types of the input type parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *`data_specification.enums.DataType`*

`get_input_type_parameters`()

Get the parameters for the input type

Returns An array of parameters

Return type array of *`spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter`*

`get_n_cpu_cycles_per_neuron`(*n_synapse_types*)

Get the number of CPU cycles executed by input_type_get_input_value once per synapse, input_type_convert_excitatory_input_to_current and input_type_convert_inhibitory_input_to_current, per neuron

`get_n_input_type_parameters`()

Get the number of parameters for the input type

Returns The number of parameters

Return type int

`get_sdram_usage_per_neuron_in_bytes`()

Get the SDRAM usage of this input type in bytes

Returns The SDRAM usage

Return type int

`set_input_type_parameters`(*parameters*, *vertex_slice*)

Sets the input type parameters. Override if there are any variables that change.

Parameters

- **`parameters`** – the parameter values in a list of numpy arrays, ordered the same as get_input_type_parameters
- **`vertex_slice`** – The neurons to which the parameters apply

class spynnaker.pyNN.models.neuron.input_types.**InputTypeConductance**(*n_neurons*, *e_rev_E*, *e_rev_I*)

Bases: spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType, spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits

The conductance input type

e_rev_E

e_rev_I

get_global_weight_scale()

get_input_type_parameter_types()

get_input_type_parameters()

get_n_cpu_cycles_per_neuron(*n_synapse_types*)

get_n_input_type_parameters()

get_units(*variable*)

get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

class spynnaker.pyNN.models.neuron.input_types.**InputTypeCurrent**

Bases: spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType, spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits

The current input type

get_global_weight_scale()

get_input_type_parameter_types()

get_input_type_parameters()

get_n_cpu_cycles_per_neuron(*n_synapse_types*)

get_n_input_type_parameters()

get_units(*variable*)

get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

spynnaker.pyNN.models.neuron.master_pop_table_generators package

Submodules

spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory module

class spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_fact

Bases: object

extract_synaptic_matrix_data_location(*incoming_key*, *master_pop_base_mem_address*, *txrx*, *chip_x*, *chip_y*)

Parameters

- **incoming_key** (*int*) – the source key which the synaptic matrix needs to be mapped to
- **master_pop_base_mem_address** (*int*) – the base address of the master pop
- **txrx** (*spinnman.transceiver.Transceiver object*) – the transceiver object from spinnman
- **chip_y** (*int*) – the y coordinate of the chip of this master pop
- **chip_x** (*int*) – the x coordinate of the chip of this master pop

Returns a synaptic matrix memory position.

finish_master_pop_table(*spec*, *master_pop_table_region*)

completes the master pop table in the spec

Parameters

- **spec** – the spec to write the master pop entry to
- **master_pop_table_region** – the region to which the master pop table is being stored

get_edge_constraints()

Gets the constraints for this table on edges coming in to a vertex that uses

Returns a list of constraints

Return type list of *pacman.model.constraints.AbstractConstraint*

get_master_population_table_size(*vertex_slice*, *in_edges*)

Get the size of the master population table in SDRAM

update_master_population_table(*spec*, *block_start_addr*, *row_length*, *keys_and_masks*, *master_pop_table_region*, *is_single=False*)

updates a spec with a master pop entry in some form

Parameters

- **spec** – the spec to write the master pop entry to
- **block_start_addr** – the start address of the row in the region
- **row_length** – the row length of this entry
- **keys_and_masks** (list of `pacman.model.routing_info.KeyAndMask`) – list of key_and_mask objects containing the keys and masks for a given edge that will require being received to be stored in the master pop table
- **master_pop_table_region** – the region to which the master pop table is being stored
- **is_single** – True if this is a single synapse, False otherwise

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array module

class spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.M

Bases: *spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory.AbstractMasterPopTableFactory*

extract_synaptic_matrix_data_location(*incoming_key*, *master_pop_base_mem_address*, *txrx*, *chip_x*, *chip_y*)

finish_master_pop_table(*spec*, *master_pop_table_region*)

get_allowed_row_length(*row_length*)

get_edge_constraints()

get_master_population_table_size(*vertex_slice*, *in_edges*)

get_next_allowed_address(*next_address*)

initialise_table(*spec*, *master_population_table_region*)

update_master_population_table(*spec*, *block_start_addr*, *row_length*, *key_and_mask*, *master_pop_table_region*, *is_single=False*)

Writes an entry in the Master Population Table for the newly created synaptic block.

An entry in the table is a 16-bit value, with the following structure: * Bits [2:0] Row length information. This value (from 0->7) indicates the maximum number of synapses in this block. It is translated in the row length translation table by the executing code each time the table is accessed, to calculate offsets. * Bits [15:3] Address within the synaptic matrix region of the start of the block. This is 1K bytes aligned, so the true value is found by shifting left by 7 bits then adding the start address of the memory region.

Parameters

- **spec** –
- **block_start_addr** –
- **row_length** –
- **key_and_mask** –
- **master_pop_table_region** –
- **is_single** – True if this is a single synapse, False otherwise

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.**get_l**

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.**get_r**

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.**get_r**

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.**get_p**

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.**get_x**

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.**get_y**

spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search module

class spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_sea

Bases: *spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory.AbstractMasterPopTableFactory*

binary search master pop class.

ADDRESS_LIST_DTYPE = '<u4'

ADDRESS_MASK = 2147483392

MASTER_POP_ENTRY_DTYPE = [('key', '<u4'), ('mask', '<u4'), ('start', '<u2'), ('count', '<u2')]

ROW_LENGTH_MASK = 255

SINGLE_BIT_FLAG_BIT = 2147483648

extract_synaptic_matrix_data_location(*incoming_key_combo*, *master_pop_base_mem_address*, *txrx*, *chip_x*, *chip_y*)

finish_master_pop_table(*spec*, *master_pop_table_region*)

Completes any operations required after all entries have been added

Parameters

- **spec** – the writer for the dsg
- **master_pop_table_region** – the region to which the master pop resides in

Return type None

get_allowed_row_length(*row_length*)

Parameters **row_length** – the row length being considered

Returns the row length available

get_edge_constraints()

Returns any constraints placed on the edges because of having this master pop table implemented in the cores.

get_exact_master_population_table_size(*vertex*, *machine_graph*, *graph_mapper*)

Returns the size the master pop table will take in SDRAM (in bytes)

get_master_population_table_size(*vertex_slice*, *in_edges*)

Parameters

- **vertex_slice** – the slice of the vertex
- **in_edges** – the in coming edges

Returns the size the master pop table will take in SDRAM (in bytes)

get_next_allowed_address(*next_address*)

Parameters **next_address** – The next address that would be used

Returns The next address that can be used following next_address

initialise_table(*spec*, *master_population_table_region*)

Initialises the master pop data structure

Parameters

- **spec** – the dsg writer
- **master_population_table_region** – the region in memory that the master pop table will be written in

Return type None

update_master_population_table(*spec*, *block_start_addr*, *row_length*, *key_and_mask*, *master_pop_table_region*, *is_single=False*)

Adds a entry in the binary search to deal with the synaptic matrix

Parameters

- **spec** – the writer for dsg
- **block_start_addr** – where the synaptic matrix block starts
- **row_length** – how long in bytes each synaptic entry is
- **key_and_mask** – the key and mask for this master pop entry
- **master_pop_table_region** – the region id for the master pop
- **is_single** – flag that states if the entry is a direct entry for a single row.

Return type None

Module contents

class spynnaker.pyNN.models.neuron.master_pop_table_generators.**MasterPopTableAs2dArray**

Bases: *spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory.AbstractMasterPopTableFactory*

extract_synaptic_matrix_data_location(*incoming_key*, *master_pop_base_mem_address*, *txrx*, *chip_x*, *chip_y*)

finish_master_pop_table(*spec*, *master_pop_table_region*)

get_allowed_row_length(*row_length*)

get_edge_constraints()

get_master_population_table_size(*vertex_slice*, *in_edges*)

get_next_allowed_address(*next_address*)

initialise_table(*spec*, *master_population_table_region*)

update_master_population_table(*spec*, *block_start_addr*, *row_length*, *key_and_mask*, *master_pop_table_region*, *is_single=False*)

Writes an entry in the Master Population Table for the newly created synaptic block.

An entry in the table is a 16-bit value, with the following structure: * Bits [2:0] Row length information. This value (from 0->7) indicates the maximum number of synapses in this block. It is translated in the row length translation table by the executing code each time the table is accessed, to calculate offsets. * Bits [15:3] Address within the synaptic matrix region of the start of the block. This is 1K bytes aligned, so the true value is found by shifting left by 7 bits then adding the start address of the memory region.

Parameters

- **spec** –
- **block_start_addr** –
- **row_length** –

- **key_and_mask** –
- **master_pop_table_region** –
- **is_single** – True if this is a single synapse, False otherwise

class spynnaker.pyNN.models.neuron.master_pop_table_generators.**MasterPopTableAsBinarySearch**

Bases: *spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory.AbstractMasterPopTableFactory*

binary search master pop class.

ADDRESS_LIST_DTYPE = '<u4'

ADDRESS_MASK = 2147483392

MASTER_POP_ENTRY_DTYPE = [('key', '<u4'), ('mask', '<u4'), ('start', '<u2'), ('count', '<u2')]

ROW_LENGTH_MASK = 255

SINGLE_BIT_FLAG_BIT = 2147483648

extract_synaptic_matrix_data_location(*incoming_key_combo*, *master_pop_base_mem_address*, *txrx*, *chip_x*, *chip_y*)

finish_master_pop_table(*spec*, *master_pop_table_region*)

Completes any operations required after all entries have been added

Parameters

- **spec** – the writer for the dsg
- **master_pop_table_region** – the region to which the master pop resides in

Return type None

get_allowed_row_length(*row_length*)

Parameters **row_length** – the row length being considered

Returns the row length available

get_edge_constraints()

Returns any constraints placed on the edges because of having this master pop table implemented in the cores.

get_exact_master_population_table_size(*vertex*, *machine_graph*, *graph_mapper*)

Returns the size the master pop table will take in SDRAM (in bytes)

get_master_population_table_size(*vertex_slice*, *in_edges*)

Parameters

- **vertex_slice** – the slice of the vertex
- **in_edges** – the in coming edges

Returns the size the master pop table will take in SDRAM (in bytes)

get_next_allowed_address(*next_address*)

Parameters **next_address** – The next address that would be used

Returns The next address that can be used following next_address

initialise_table(*spec*, *master_population_table_region*)

Initialises the master pop data structure

Parameters

- **spec** – the dsg writer
- **master_population_table_region** – the region in memory that the master pop table will be written in

Return type None

update_master_population_table(*spec*, *block_start_addr*, *row_length*, *key_and_mask*, *master_pop_table_region*, *is_single=False*)

Adds a entry in the binary search to deal with the synaptic matrix

Parameters

- **spec** – the writer for dsg
- **block_start_addr** – where the synaptic matrix block starts
- **row_length** – how long in bytes each synaptic entry is
- **key_and_mask** – the key and mask for this master pop entry
- **master_pop_table_region** – the region id for the master pop
- **is_single** – flag that states if the entry is a direct entry for a single row.

Return type None

spynnaker.pyNN.models.neuron.neuron_models package

Submodules

spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model module

class spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.**AbstractNeuronModel**

Bases: object

Represents a neuron model

get_dtcm_usage_per_neuron_in_bytes()

Get the DTCM usage of this neuron model in bytes

Returns The DTCM usage

Return type int

get_global_parameter_types()

Get the types of the global parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_global_parameters()

Get the global parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_n_cpu_cycles_per_neuron()
Get the total number of CPU cycles executed by neuron_model_state_update and neuron_model_has_spiked, per neuron

Returns The number of CPU cycles executed

Return type int

get_n_global_parameters()
Get the number of global parameters

Returns The number of global parameters

Return type int

get_n_neural_parameters()
Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()
Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_neural_parameters()
Get the neural parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_sdram_usage_for_global_parameters_in_bytes()
Get the SDRAM usage of the global parameters in bytes

Returns The SDRAM usage

Return type int

get_sdram_usage_per_neuron_in_bytes()
Get the total sdram usage in bytes

Returns The SDRAM usage

Return type int

set_global_parameters(*parameters*)
Sets any global parameters. Override if there are changing variables in the global parameters

Parameters **parameters** – the parameter values as a list, ordered the same as get_global_parameters

set_neural_parameters(*neural_parameters*, *vertex_slice*)
Sets any neural parameters. Override if there are changing variables in the neural parameters

Parameters

- **neural_parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh module

class spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.**NeuronModelIzh**(*n_neurons*, *a*, *b*, *c*, *d*, *v_init*, *u_init*, *i_offset*)

Bases: *spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

a

b

c

d

get_global_parameter_types()

Get the types of the global parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_global_parameters(*obj*, **args*, ***kwargs*)

Get the global parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_n_cpu_cycles_per_neuron()

get_n_global_parameters()

Get the number of global parameters

Returns The number of global parameters

Return type int

get_n_neural_parameters()

Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()

Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_neural_parameters(*obj*, **args*, ***kwargs*)

Get the neural parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

i_offset

initialize_u(*u_init*)

initialize_v(*v_init*)

set_neural_parameters(*neural_parameters*, *vertex_slice*)
Sets any neural parameters. Override if there are changing variables in the neural parameters

Parameters

- **neural_parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

u_init

v_init

spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate module

class spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.**NeuronModelLeak**

Bases: *spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

cm

get_global_parameter_types()
Get the types of the global parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_global_parameters()
Get the global parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_n_cpu_cycles_per_neuron()

get_n_global_parameters()
Get the number of global parameters

Returns The number of global parameters

Return type int

get_n_neural_parameters()
Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()
Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_neural_parameters(*obj*, **args*, ***kwargs*)
Get the neural parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

i_offset

initialize_v(*v_init*)

set_neural_parameters(*neural_parameters*, *vertex_slice*)
Sets any neural parameters. Override if there are changing variables in the neural parameters

Parameters

- **neural_parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

tau_m

v_init

v_rest

spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire module

class spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.**Neuron**

Bases: *spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.*

NeuronModelLeakyIntegrate

get_n_cpu_cycles_per_neuron()

get_n_neural_parameters()
Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()
Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_neural_parameters(*obj*, **args*, ***kwargs*)

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

tau_refrac

v_reset

Module contents

class spynnaker.pyNN.models.neuron.neuron_models.**AbstractNeuronModel**
Bases: object

Represents a neuron model

get_dtcm_usage_per_neuron_in_bytes()
Get the DTCM usage of this neuron model in bytes

Returns The DTCM usage

Return type int

get_global_parameter_types()
Get the types of the global parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_global_parameters()
Get the global parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_n_cpu_cycles_per_neuron()
Get the total number of CPU cycles executed by neuron_model_state_update and neuron_model_has_spiked, per neuron

Returns The number of CPU cycles executed

Return type int

get_n_global_parameters()
Get the number of global parameters

Returns The number of global parameters

Return type int

get_n_neural_parameters()
Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()
Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_neural_parameters()
Get the neural parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_sdram_usage_for_global_parameters_in_bytes()
Get the SDRAM usage of the global parameters in bytes

Returns The SDRAM usage

Return type int

get_sdram_usage_per_neuron_in_bytes()
Get the total sdram usage in bytes

Returns The SDRAM usage

Return type int

set_global_parameters(*parameters*)
Sets any global parameters. Override if there are changing variables in the global parameters

Parameters **parameters** – the parameter values as a list, ordered the same as get_global_parameters

set_neural_parameters(*neural_parameters*, *vertex_slice*)
Sets any neural parameters. Override if there are changing variables in the neural parameters

Parameters

- **neural_parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

class spynnaker.pyNN.models.neuron.neuron_models.**NeuronModelIzh**(*n_neurons*, *a*, *b*, *c*, *d*, *v_init*, *u_init*, *i_offset*)

Bases: *spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

a

b

c

d

get_global_parameter_types()
Get the types of the global parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of `data_specification.enums.DataType`

get_global_parameters(*obj*, **args*, ***kwargs*)
Get the global parameters

Returns an array of parameters

Return type array of `spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter`

get_n_cpu_cycles_per_neuron()

get_n_global_parameters()
Get the number of global parameters

Returns The number of global parameters

Return type int

get_n_neural_parameters()
Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()
Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of `data_specification.enums.DataType`

get_neural_parameters(*obj*, **args*, ***kwargs*)
Get the neural parameters

Returns an array of parameters

Return type array of `spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter`

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

i_offset

initialize_u(*u_init*)

initialize_v(*v_init*)

set_neural_parameters(*neural_parameters*, *vertex_slice*)
Sets any neural parameters. Override if there are changing variables in the neural parameters

Parameters

- **neural_parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

u_init

v_init

class spynnaker.pyNN.models.neuron.neuron_models.**NeuronModelLeakyIntegrate**(*n_neurons*, *v_init*, *v_rest*, *tau_m*, *cm*, *i_offset*)

Bases: *spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

cm

get_global_parameter_types()
Get the types of the global parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_global_parameters()
Get the global parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_n_cpu_cycles_per_neuron()

get_n_global_parameters()
Get the number of global parameters

Returns The number of global parameters

Return type int

get_n_neural_parameters()
Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()
Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_neural_parameters(*obj*, **args*, ***kwargs*)
Get the neural parameters

Returns an array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

i_offset

initialize_v(*v_init*)

set_neural_parameters(*neural_parameters*, *vertex_slice*)
Sets any neural parameters. Override if there are changing variables in the neural parameters

Parameters

- **neural_parameters** – the parameter values in a list of numpy arrays, ordered the same as get_neural_parameters
- **vertex_slice** – The neurons to which the parameters apply

tau_m

v_init

v_rest

class spynnaker.pyNN.models.neuron.neuron_models.**NeuronModelLeakyIntegrateAndFire**(*n_neurons*, *v_init*, *v_rest*, *tau_m*, *cm*, *i_offset*, *v_reset*, *tau_refrac*)

Bases: *spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate*

get_n_cpu_cycles_per_neuron()

get_n_neural_parameters()
Get the number of neural parameters

Returns The number of parameters

Return type int

get_neural_parameter_types()
Get the types of the neural parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_neural_parameters(*obj*, **args*, ***kwargs*)

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

tau_refrac

v_reset

spynnaker.pyNN.models.neuron.plasticity package

Subpackages

spynnaker.pyNN.models.neuron.plasticity.stdp package

Subpackages

spynnaker.pyNN.models.neuron.plasticity.stdp.common package

Submodules

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers module

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers.**float_to_fixed**(*value*, *fixed_p*

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers.**get_lut_provenance**(*p*
n
e
t
p
l

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers.**write_exp_lut**(*spec*, *time_co*
size,
shift,
fixed_po

Module contents

spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure package

Submodules

spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure module

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structu

Bases: object

get_n_bytes_per_connection()

Get the number of bytes for each connection

get_synaptic_data(*connections*)

Get the plastic synaptic data for this connection

read_synaptic_data(*fp_size*, *pp_data*)
Read the plastic synaptic data for this connection from the data

spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_weight_accumulator module

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_weight
Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_bytes_per_connection()

get_synaptic_data(*connections*)

read_synaptic_data(*fp_size*, *pp_data*)

spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_weight_only module

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_weight
Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_bytes_per_connection()

get_synaptic_data(*connections*)

read_synaptic_data(*fp_size*, *pp_data*)

Module contents

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.**AbstractSynapseStructure**
Bases: object

get_n_bytes_per_connection()
Get the number of bytes for each connection

get_synaptic_data(*connections*)
Get the plastic synaptic data for this connection

read_synaptic_data(*fp_size*, *pp_data*)
Read the plastic synaptic data for this connection from the data

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.**SynapseStructureWeightOr**
Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_bytes_per_connection()

get_synaptic_data(*connections*)

read_synaptic_data(*fp_size*, *pp_data*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.**SynapseStructureWeightAc**
Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_bytes_per_connection()

get_synaptic_data(*connections*)

read_synaptic_data(*fp_size*, *pp_data*)

spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence package

Submodules

spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_depender

Bases: object

get_parameter_names()

get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()

Get the amount of SDRAM used by the parameters of this rule

get_provenance_data(*pre_population_label*, *post_population_label*)

Get any provenance data

is_same_as(*timing_dependence*)

Determine if this timing dependence is the same as another

n_weight_terms

The number of weight terms expected by this timing rule

pre_trace_n_bytes

The number of bytes used by the pre-trace of the rule per neuron

synaptic_structure

Get the synaptic structure of the plastic part of the rows

vertex_executable_suffix

The suffix to be appended to the vertex executable for this rule

write_parameters(*spec*, *machine_time_step*, *weight_scales*)

Write the parameters of the rule to the spec

spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_pfister_spike_triplet module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_pfiste

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

get_parameter_names()

get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()

get_provenance_data(*pre_population_label*, *post_population_label*)

is_same_as(*timing_dependence*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau_minus

tau_plus

tau_x

tau_y

vertex_executable_suffix

write_parameters(*spec*, *machine_time_step*, *weight_scales*)

spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurr

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

default_parameters = {'accumulator_potentiation': 6, 'dual_fsm': True, 'mean_pre_window': 35.0, 'mean_post_wi

get_parameter_names()

get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()

is_same_as(*other*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

vertex_executable_suffix

write_parameters(*spec*, *machine_time_step*, *weight_scales*)

spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

default_parameters = {'tau_minus': 20.0, 'tau_plus': 20.0}

get_parameter_names()
get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()

get_provenance_data(*pre_population_label*, *post_population_label*)

is_same_as(*timing_dependence*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau_minus

tau_plus

vertex_executable_suffix

write_parameters(*spec*, *machine_time_step*, *weight_scales*)

spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

get_parameter_names()
get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()
Get the amount of SDRAM used by the parameters of this rule

get_provenance_data(*pre_population_label*, *post_population_label*)

is_same_as(*timing_dependence*)
Determine if this timing dependence is the same as another

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau_minus

tau_plus

vertex_executable_suffix

write_parameters (*spec*, *machine_time_step*, *weight_scales*)

spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels_2011 module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

default_parameters = {'tau': 20.0}

get_parameter_names ()
get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes ()

is_same_as (*other*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau

vertex_executable_suffix

write_parameters (*spec*, *machine_time_step*, *weight_scales*)

Module contents

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.**AbstractTimingDependence**
Bases: object

get_parameter_names ()
get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes ()
Get the amount of SDRAM used by the parameters of this rule

get_provenance_data (*pre_population_label*, *post_population_label*)
Get any provenance data

is_same_as (*timing_dependence*)
Determine if this timing dependence is the same as another

n_weight_terms
The number of weight terms expected by this timing rule

pre_trace_n_bytes
The number of bytes used by the pre-trace of the rule per neuron

synaptic_structure
Get the synaptic structure of the plastic part of the rows

vertex_executable_suffix
The suffix to be appended to the vertex executable for this rule

write_parameters(*spec*, *machine_time_step*, *weight_scales*)
Write the parameters of the rule to the spec

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.**TimingDependenceSpikePai**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

get_parameter_names()
get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()
Get the amount of SDRAM used by the parameters of this rule

get_provenance_data(*pre_population_label*, *post_population_label*)

is_same_as(*timing_dependence*)
Determine if this timing dependence is the same as another

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau_minus

tau_plus

vertex_executable_suffix

write_parameters(*spec*, *machine_time_step*, *weight_scales*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.**TimingDependencePfisterS**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

get_parameter_names()
get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()

get_provenance_data(*pre_population_label*, *post_population_label*)

is_same_as(*timing_dependence*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau_minus

tau_plus

tau_x

tau_y

vertex_executable_suffix

write_parameters(*spec*, *machine_time_step*, *weight_scales*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.**TimingDependenceRecurrer**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

default_parameters = {'accumulator_potentiation': 6, 'dual_fsm': True, 'mean_pre_window': 35.0, 'mean_post_wi

get_parameter_names()

get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()

is_same_as(*other*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

vertex_executable_suffix

write_parameters(*spec*, *machine_time_step*, *weight_scales*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.**TimingDependenceSpikeNea**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

default_parameters = {'tau_minus': 20.0, 'tau_plus': 20.0}

get_parameter_names()

get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes()

get_provenance_data(*pre_population_label*, *post_population_label*)

is_same_as(*timing_dependence*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau_minus

tau_plus

vertex_executable_suffix

write_parameters (*spec*, *machine_time_step*, *weight_scales*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.**TimingDependenceVogels20**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence*

default_parameters = {'tau': 20.0}

get_parameter_names ()

get the params from the timing dependencies

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes ()

is_same_as (*other*)

n_weight_terms

pre_trace_n_bytes

synaptic_structure

tau

vertex_executable_suffix

write_parameters (*spec*, *machine_time_step*, *weight_scales*)

spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence package

Submodules

spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus module

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_mi

Bases: object

A_minus

A_plus

set_a_plus_a_minus (*a_plus*, *a_minus*)

spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence module

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_depender

Bases: object

get_parameter_names ()

returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes (*n_synapse_types*, *n_weight_terms*)

Get the amount of SDRAM used by the parameters of this rule

get_provenance_data (*pre_population_label*, *post_population_label*)
Get any provenance data

Parameters **pre_population_label** – label of pre.

:param post_population_label:label of post. :return the provenance data of the weight dependency

is_same_as (*weight_dependence*)
Determine if this weight dependence is the same as another

vertex_executable_suffix
The suffix to be appended to the vertex executable for this rule

weight_maximum
The maximum weight that will ever be set in a synapse as a result of this rule

write_parameters (*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)
Write the parameters of the rule to the spec

spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive module

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additi

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*, *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus*

get_parameter_names ()
returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes (*n_synapse_types*, *n_weight_terms*)

is_same_as (*weight_dependence*)

vertex_executable_suffix

w_max

w_min

weight_maximum

write_parameters (*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)

spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet module

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additi

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*, *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus*

A3_minus

A3_plus

default_parameters = {'A3_plus': 0.01, 'w_min': 0.0, 'w_max': 1.0, 'A3_minus': 0.01}

get_parameter_names()
returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes(*n_synapse_types*, *n_weight_terms*)

is_same_as(*weight_dependence*)

vertex_executable_suffix

w_max

w_min

weight_maximum

write_parameters(*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)

spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multiplicative module

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multip

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*, *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus*

get_parameter_names()
returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes(*n_synapse_types*, *n_weight_terms*)

is_same_as(*weight_dependence*)

vertex_executable_suffix

w_max

w_min

weight_maximum

write_parameters(*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)

Module contents

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.**AbstractHasAPlusAMinus**
Bases: object

A_minus

A_plus

set_a_plus_a_minus(*a_plus*, *a_minus*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.**AbstractWeightDependence**

Bases: object

get_parameter_names()

returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes(*n_synapse_types*, *n_weight_terms*)

Get the amount of SDRAM used by the parameters of this rule

get_provenance_data(*pre_population_label*, *post_population_label*)

Get any provenance data

Parameters **pre_population_label** – label of pre.

:param post_population_label:label of post. :return the provenance data of the weight dependency

is_same_as(*weight_dependence*)

Determine if this weight dependence is the same as another

vertex_executable_suffix

The suffix to be appended to the vertex executable for this rule

weight_maximum

The maximum weight that will ever be set in a synapse as a result of this rule

write_parameters(*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)

Write the parameters of the rule to the spec

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.**WeightDependenceAdditive**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*, *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus*

get_parameter_names()

returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes(*n_synapse_types*, *n_weight_terms*)

is_same_as(*weight_dependence*)

vertex_executable_suffix

w_max

w_min

weight_maximum

write_parameters(*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.**WeightDependenceMultipli**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*, *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus*

get_parameter_names()

returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes(*n_synapse_types*, *n_weight_terms*)

is_same_as(*weight_dependence*)

vertex_executable_suffix

w_max

w_min

weight_maximum

write_parameters(*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.**WeightDependenceAdditive**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*, *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus*

A3_minus

A3_plus

default_parameters = {'A3_plus': 0.01, 'w_min': 0.0, 'w_max': 1.0, 'A3_minus': 0.01}

get_parameter_names()

returns iterable of parameter names as basestrings

Returns iterable of basestring

get_parameters_sdram_usage_in_bytes(*n_synapse_types*, *n_weight_terms*)

is_same_as(*weight_dependence*)

vertex_executable_suffix

w_max

w_min

weight_maximum

write_parameters(*spec*, *machine_time_step*, *weight_scales*, *n_weight_terms*)

Module contents

Module contents

spynnaker.pyNN.models.neuron.synapse_dynamics package

Submodules

spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics module

class spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.**Abstrac**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.*

abstract_synapse_dynamics.AbstractSynapseDynamics

AbstractPlasticSynapseDynamics : synapses which change over time

get_n_fixed_plastic_words_per_row(*fp_size*)
Get the number of fixed plastic words to be read from each row

get_n_plastic_plastic_words_per_row(*pp_size*)
Get the number of plastic plastic words to be read from each row

get_n_synapses_in_rows(*pp_size*, *fp_size*)
Get the number of synapses in each of the rows with plastic sizes pp_size and fp_size

get_n_words_for_plastic_connections(*n_connections*)
Get the number of 32-bit words for n_connections in a single row

get_plastic_synaptic_data(*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)
Get the fixed-plastic data, and plastic-plastic data for each row, and lengths for the fixed_plastic and plastic-plastic parts of each row. Data is returned as an array made up of an array of 32-bit words for each row, for each of the fixed-plastic and plastic-plastic data regions. The row into which connection should go is given by connection_row_indices, and the total number of rows is given by n_rows. Lengths are returned as an array made up of an integer for each row, for each of the fixed-plastic and plastic-plastic regions.

read_plastic_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *pp_size*, *pp_data*, *fp_size*, *fp_data*)
Read the connections indicated in the connection indices from the data in pp_data and fp_data

spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics module

class spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.**Abstract**
Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics*

AbstractStaticSynapseDynamics: dynamics which don't change over time.

get_n_static_words_per_row(*ff_size*)
Get the number of bytes to be read per row for the static data given the size that was written to each row

get_n_synapses_in_rows(*ff_size*)
Get the number of synapses in the rows with sizes ff_size

get_n_words_for_static_connections(*n_connections*)
Get the number of 32-bit words for n_connections in a single row

get_static_synaptic_data(*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)
Get the fixed-fixed data for each row, and lengths for the fixed-fixed parts of each row. Data is returned as an array made up of an array of 32-bit words for each row for the fixed-fixed region. The row into which connection should go is given by connection_row_indices, and the total number of rows is given by n_rows. Lengths are returned as an array made up of an integer for each row, for the fixed-fixed region.

read_static_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *ff_size*, *ff_data*)
Read the connections from the words of data in ff_data

spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics module

class spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.**AbstractSynapse**
Bases: object

NUMPY_CONNECTORS_DTYPE = [('source', 'uint32'), ('target', 'uint32'), ('weight', 'float64'), ('delay', 'float64')]

are_weights_signed()
Determines if the weights are signed values

convert_per_connection_data_to_rows(*connection_row_indices*, *n_rows*, *data*)
Converts per-connection data generated from connections into row-based data to be returned from get_synaptic_data

get_delay_maximum(*connector*)
Get the maximum delay for the synapses

get_delay_variance(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the variance in delay for the synapses

get_max_synapses(*n_words*)
Get the maximum number of synapses that can be held in the given number of words

Parameters **n_words** – The number of words the synapses must fit in

Return type int

get_n_items(*rows*, *item_size*)
Get the number of items in each row as 4-byte values, given the item size

get_parameter_names()
return the parameter names available from the synapse dynamics components

Returns iterable list of basestring

get_parameters_sdram_usage_in_bytes(*n_neurons*, *n_synapse_types*)
Get the SDRAM usage of the synapse dynamics parameters in bytes

get_provenance_data(*pre_population_label*, *post_population_label*)
Get the provenance data from this synapse dynamics object

get_vertex_executable_suffix()
Get the executable suffix for a vertex for this dynamics

get_weight_maximum(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the maximum weight for the synapses

get_weight_mean(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the mean weight for the synapses

get_weight_variance(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the variance in weight for the synapses

get_words(*rows*)
Convert the row data to words

is_same_as(*synapse_dynamics*)
Determines if this synapse dynamics is the same as another

write_parameters(*spec*, *region*, *machine_time_step*, *weight_scales*)
Write the synapse parameters to the spec

spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_synapse_dynamics module

class spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_synapse_dynamics.**PyNNSynapseDynamics**

Bases: object

slow

spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static module

class spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.**SynapseDynamicsSt**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamics*, *spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable*, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*

are_weights_signed()

get_max_synapses(*n_words*)

Get the maximum number of synapses that can be held in the given number of words

Parameters **n_words** – The number of words the synapses must fit in

Return type int

get_n_static_words_per_row(*ff_size*)

get_n_synapses_in_rows(*ff_size*)

get_n_words_for_static_connections(*n_connections*)

get_parameter_names()

return the parameter names available from the synapse dynamics components

Returns iterable list of basestring

get_parameters_sdram_usage_in_bytes(*n_neurons*, *n_synapse_types*)

get_static_synaptic_data(*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)

get_value(*key*)

Get a property Get a property

get_vertex_executable_suffix()

is_same_as(*synapse_dynamics*)

mark_no_changes()

Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False. Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

read_static_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *ff_size*, *ff_data*)

requires_mapping()

True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created! True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

set_value (*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

write_parameters (*spec*, *region*, *machine_time_step*, *weight_scales*)

spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp module

class spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.**SynapseDynamicsSTDP**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics*, *spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable*, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*

are_weights_signed ()

dendritic_delay_fraction

get_max_synapses (*n_words*)
Get the maximum number of synapses that can be held in the given number of words

Parameters **n_words** – The number of words the synapses must fit in

Return type int

get_n_fixed_plastic_words_per_row (*fp_size*)

get_n_plastic_plastic_words_per_row (*pp_size*)

get_n_synapses_in_rows (*pp_size*, *fp_size*)

get_n_words_for_plastic_connections (*n_connections*)

get_parameter_names ()
return the parameter names available from the synapse dynamics components

Returns iterable list of basestring

get_parameters_sdram_usage_in_bytes (*n_neurons*, *n_synapse_types*)

get_plastic_synaptic_data (*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)

get_provenance_data (*pre_population_label*, *post_population_label*)

get_value (*key*)
Get a property Get a property

get_vertex_executable_suffix ()

get_weight_maximum(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

is_same_as(*synapse_dynamics*)

mark_no_changes()

Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False. Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

read_plastic_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *pp_size*, *pp_data*, *fp_size*, *fp_data*)

requires_mapping()

True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created! True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

set_value(*key*, *value*)

Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

timing_dependence

weight_dependence

write_parameters(*spec*, *region*, *machine_time_step*, *weight_scales*)

Module contents

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractSynapseDynamics**

Bases: object

NUMPY_CONNECTORS_DTYPE = [('source', 'uint32'), ('target', 'uint32'), ('weight', 'float64'), ('delay', 'float64')]

are_weights_signed()

Determines if the weights are signed values

convert_per_connection_data_to_rows(*connection_row_indices*, *n_rows*, *data*)

Converts per-connection data generated from connections into row-based data to be returned from get_synaptic_data

get_delay_maximum(*connector*)

Get the maximum delay for the synapses

get_delay_variance(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

Get the variance in delay for the synapses

get_max_synapses(*n_words*)
Get the maximum number of synapses that can be held in the given number of words

Parameters **n_words** – The number of words the synapses must fit in

Return type int

get_n_items(*rows*, *item_size*)
Get the number of items in each row as 4-byte values, given the item size

get_parameter_names()
return the parameter names available from the synapse dynamics components

Returns iterable list of basestring

get_parameters_sdram_usage_in_bytes(*n_neurons*, *n_synapse_types*)
Get the SDRAM usage of the synapse dynamics parameters in bytes

get_provenance_data(*pre_population_label*, *post_population_label*)
Get the provenance data from this synapse dynamics object

get_vertex_executable_suffix()
Get the executable suffix for a vertex for this dynamics

get_weight_maximum(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the maximum weight for the synapses

get_weight_mean(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the mean weight for the synapses

get_weight_variance(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)
Get the variance in weight for the synapses

get_words(*rows*)
Convert the row data to words

is_same_as(*synapse_dynamics*)
Determines if this synapse dynamics is the same as another

write_parameters(*spec*, *region*, *machine_time_step*, *weight_scales*)
Write the synapse parameters to the spec

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractStaticSynapseDynamics**
Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics*

AbstractStaticSynapseDynamics: dynamics which don't change over time.

get_n_static_words_per_row(*ff_size*)
Get the number of bytes to be read per row for the static data given the size that was written to each row

get_n_synapses_in_rows(*ff_size*)
Get the number of synapses in the rows with sizes ff_size

get_n_words_for_static_connections(*n_connections*)
Get the number of 32-bit words for n_connections in a single row

get_static_synaptic_data(*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)
Get the fixed-fixed data for each row, and lengths for the fixed-fixed parts of each row. Data is returned as an array made up of an array of 32-bit words for each row for the fixed-fixed region. The row into

which connection should go is given by connection_row_indices, and the total number of rows is given by n_rows. Lengths are returned as an array made up of an integer for each row, for the fixed-fixed region.

read_static_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *ff_size*, *ff_data*)
Read the connections from the words of data in ff_data

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractPlasticSynapseDynamics**
Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics*

AbstractPlasticSynapseDynamics : synapses which change over time

get_n_fixed_plastic_words_per_row(*fp_size*)
Get the number of fixed plastic words to be read from each row

get_n_plastic_plastic_words_per_row(*pp_size*)
Get the number of plastic plastic words to be read from each row

get_n_synapses_in_rows(*pp_size*, *fp_size*)
Get the number of synapses in each of the rows with plastic sizes pp_size and fp_size

get_n_words_for_plastic_connections(*n_connections*)
Get the number of 32-bit words for n_connections in a single row

get_plastic_synaptic_data(*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)
Get the fixed-plastic data, and plastic-plastic data for each row, and lengths for the fixed_plastic and plastic-plastic parts of each row. Data is returned as an array made up of an array of 32-bit words for each row, for each of the fixed-plastic and plastic-plastic data regions. The row into which connection should go is given by connection_row_indices, and the total number of rows is given by n_rows. Lengths are returned as an array made up of an integer for each row, for each of the fixed-plastic and plastic-plastic regions.

read_plastic_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *pp_size*, *pp_data*, *fp_size*, *fp_data*)
Read the connections indicated in the connection indices from the data in pp_data and fp_data

class spynnaker.pyNN.models.neuron.synapse_dynamics.**PyNNSynapseDynamics**(*slow=None*, *fast=None*)
Bases: `object`

slow

class spynnaker.pyNN.models.neuron.synapse_dynamics.**SynapseDynamicsStatic**
Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamics*, *spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable*, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*

are_weights_signed()

get_max_synapses(*n_words*)
Get the maximum number of synapses that can be held in the given number of words

Parameters **n_words** – The number of words the synapses must fit in

Return type int

get_n_static_words_per_row(*ff_size*)

get_n_synapses_in_rows(*ff_size*)

get_n_words_for_static_connections(*n_connections*)

get_parameter_names()
return the parameter names available from the synapse dynamics components

Returns iterable list of basestring

get_parameters_sdram_usage_in_bytes(*n_neurons*, *n_synapse_types*)

get_static_synaptic_data(*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)

get_value(*key*)
Get a property Get a property

get_vertex_executable_suffix()

is_same_as(*synapse_dynamics*)

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False. Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

read_static_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *ff_size*, *ff_data*)

requires_mapping()
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created! True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

write_parameters(*spec*, *region*, *machine_time_step*, *weight_scales*)

class spynnaker.pyNN.models.neuron.synapse_dynamics.**SynapseDynamicsSTDP**(*timing_dependence=None*, *weight_dependence=None*, *voltage_dependence=None*, *dendritic_delay_fraction=1.0*)

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics*, *spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable*, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*

are_weights_signed()

dendritic_delay_fraction

get_max_synapses(*n_words*)
Get the maximum number of synapses that can be held in the given number of words

Parameters **n_words** – The number of words the synapses must fit in

Return type int

get_n_fixed_plastic_words_per_row(*fp_size*)

get_n_plastic_plastic_words_per_row(*pp_size*)

get_n_synapses_in_rows(*pp_size*, *fp_size*)

get_n_words_for_plastic_connections(*n_connections*)

get_parameter_names()
return the parameter names available from the synapse dynamics components

Returns iterable list of basestring

get_parameters_sdram_usage_in_bytes(*n_neurons*, *n_synapse_types*)

get_plastic_synaptic_data(*connections*, *connection_row_indices*, *n_rows*, *post_vertex_slice*, *n_synapse_types*)

get_provenance_data(*pre_population_label*, *post_population_label*)

get_value(*key*)
Get a property Get a property

get_vertex_executable_suffix()

get_weight_maximum(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_mean(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

get_weight_variance(*connector*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*)

is_same_as(*synapse_dynamics*)

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False. Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

read_plastic_synaptic_data(*post_vertex_slice*, *n_synapse_types*, *pp_size*, *pp_data*, *fp_size*, *fp_data*)

requires_mapping()
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created! True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

timing_dependence

weight_dependence

write_parameters (*spec*, *region*, *machine_time_step*, *weight_scales*)

spynnaker.pyNN.models.neuron.synapse_io package

Submodules

spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io module

class spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.**AbstractSynapseIO**

Bases: object

get_block_n_bytes (*max_row_length*, *n_rows*)

Get the number of bytes in a block given the max row length and number of rows

get_maximum_delay_supported_in_ms (*machine_time_step*)

Get the maximum delay supported by the synapse representation before extensions are required, or None if any delay is supported

get_sdram_usage_in_bytes (*edge*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*)

Get the SDRAM usage of a list of synapse information objects for the given slices, and given number of delay stages (each stage representing a multiple of the maximum delay supported), returning the size for the non-delayed synapse information and the size for the delayed information

get_synapses (*edge*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*, *n_synapse_types*, *weight_scales*, *machine_time_step*)

Get the synapses as an array of words for non-delayed synapses and an array of words for delayed synapses

read_synapses (*edge*, *synapse_info*, *pre_vertex_slice*, *post_vertex_slice*, *max_row_length*, *delayed_max_row_length*, *n_synapse_types*, *weight_scales*, *data*, *delayed_data*, *machine_time_step*)

Read the synapses for a given projection synapse information object out of the given data

spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based module

class spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.**SynapseIORowBased**

Bases: *spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO*

A SynapseRowIO implementation that uses a row for each source neuron, where each row consists of a fixed region, a plastic region, and a fixed-plastic region (this is the bits of the plastic row that don't actually change). The plastic region structure is determined by the synapse dynamics of the connector.

get_block_n_bytes (*max_row_length*, *n_rows*)

get_maximum_delay_supported_in_ms (*machine_time_step*)

get_sdram_usage_in_bytes (*synapse_info*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*, *machine_time_step*, *in_edge*)

get_synapses (*synapse_info*, *pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*, *n_synapse_types*, *weight_scales*, *machine_time_step*)

read_synapses(*synapse_info*, *pre_vertex_slice*, *post_vertex_slice*, *max_row_length*, *delayed_max_row_length*, *n_synapse_types*, *weight_scales*, *data*, *delayed_data*, *n_delay_stages*, *machine_time_step*)

Module contents

class spynnaker.pyNN.models.neuron.synapse_io.**AbstractSynapseIO**

Bases: object

get_block_n_bytes(*max_row_length*, *n_rows*)

Get the number of bytes in a block given the max row length and number of rows

get_maximum_delay_supported_in_ms(*machine_time_step*)

Get the maximum delay supported by the synapse representation before extensions are required, or None if any delay is supported

get_sdram_usage_in_bytes(*edge*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*)

Get the SDRAM usage of a list of synapse information objects for the given slices, and given number of delay stages (each stage representing a multiple of the maximum delay supported), returning the size for the non-delayed synapse information and the size for the delayed information

get_synapses(*edge*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*, *n_synapse_types*, *weight_scales*, *machine_time_step*)

Get the synapses as an array of words for non-delayed synapses and an array of words for delayed synapses

read_synapses(*edge*, *synapse_info*, *pre_vertex_slice*, *post_vertex_slice*, *max_row_length*, *delayed_max_row_length*, *n_synapse_types*, *weight_scales*, *data*, *delayed_data*, *machine_time_step*)

Read the synapses for a given projection synapse information object out of the given data

class spynnaker.pyNN.models.neuron.synapse_io.**SynapseIORowBased**

Bases: *spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO*

A SynapseRowIO implementation that uses a row for each source neuron, where each row consists of a fixed region, a plastic region, and a fixed-plastic region (this is the bits of the plastic row that don't actually change). The plastic region structure is determined by the synapse dynamics of the connector.

get_block_n_bytes(*max_row_length*, *n_rows*)

get_maximum_delay_supported_in_ms(*machine_time_step*)

get_sdram_usage_in_bytes(*synapse_info*, *n_pre_slices*, *pre_slice_index*, *n_post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*, *machine_time_step*, *in_edge*)

get_synapses(*synapse_info*, *pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *n_delay_stages*, *population_table*, *n_synapse_types*, *weight_scales*, *machine_time_step*)

read_synapses(*synapse_info*, *pre_vertex_slice*, *post_vertex_slice*, *max_row_length*, *delayed_max_row_length*, *n_synapse_types*, *weight_scales*, *data*, *delayed_data*, *n_delay_stages*, *machine_time_step*)

spynnaker.pyNN.models.neuron.synapse_types package

Submodules

spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type module

class spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.**AbstractSynapseType**

Bases: object

Represents the synapse types supported

get_dtcm_usage_per_neuron_in_bytes()

Get the DTCM usage of the synapse type per neuron in bytes

Returns the number of bytes

Return type int

get_n_cpu_cycles_per_neuron()

Get the total number of CPU cycles executed by synapse_types_shape_input, synapse_types_add_neuron_input, synapse_types_get_excitatory_input and synapse_types_get_inhibitory_input

Returns The number of CPU cycles

Return type int

get_n_synapse_type_bits()

Get the number of bits required to represent the synapse types

Returns the number of bits

Return type int

get_n_synapse_type_parameters()

Get the number of synapse type parameters

Returns the number of parameters

Return type int

get_n_synapse_types()

Get the number of synapse types supported

Returns The number of synapse types supported

Return type int

get_sdram_usage_per_neuron_in_bytes()

Get the SDRAM usage of the synapse type per neuron in bytes

Returns the number of bytes

Return type int

get_synapse_id_by_target(*target*)

Get the id of a synapse given the name

Returns The id of the synapse

Return type int

get_synapse_targets()

Get the target names of the synapse type

Returns an array of strings

Return type array of str

get_synapse_type_parameter_types()

Get the types of the synapse parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_synapse_type_parameters()

Get the synapse type parameters

Returns The parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

set_synapse_type_parameters(*parameters*, *vertex_slice*)

Sets any synapse type parameters. Override if there are changing variables in the synapse type parameters

Parameters

- **parameters** – the parameter values in a list of numpy arrays, ordered the same as get_synapse_type_parameters
- **vertex_slice** – The neurons to which the parameters apply

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.**SynapseTypeDelta**(*n_neurons*, *initial_input_e*, *initial_input_i*)

Bases: *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*

This represents a synapse type with two delta synapses

get_n_cpu_cycles_per_neuron()

get_n_synapse_type_parameters()

get_n_synapse_types()

get_synapse_id_by_target(*target*)

get_synapse_targets()

get_synapse_type_parameter_types()

get_synapse_type_parameters()

isyn_exc

isyn_inh

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.**SynapseTypeDua**

Bases: *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

get_n_cpu_cycles_per_neuron()

get_n_synapse_type_parameters()

get_n_synapse_types()

get_synapse_id_by_target(*target*)

get_synapse_targets()

get_synapse_type_parameter_types()

get_synapse_type_parameters(*obj*, **args*, ***kwargs*)

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

isyn_exc

isyn_exc2

isyn_inh

tau_syn_E

tau_syn_E2

tau_syn_I

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.**SynapseTypeExponent**

Bases: *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

get_n_cpu_cycles_per_neuron()

get_n_synapse_type_parameters()

get_n_synapse_types()

get_synapse_id_by_target(*target*)

get_synapse_targets()

get_synapse_type_parameter_types()

get_synapse_type_parameters(*obj*, **args*, ***kwargs*)

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

isyn_exc

isyn_inh

tau_syn_E

tau_syn_I

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.**get_exponential_decay_a**

Module contents

class spynnaker.pyNN.models.neuron.synapse_types.**AbstractSynapseType**
Bases: object

Represents the synapse types supported

get_dtcm_usage_per_neuron_in_bytes()
Get the DTCM usage of the synapse type per neuron in bytes

Returns the number of bytes

Return type int

get_n_cpu_cycles_per_neuron()
Get the total number of CPU cycles executed by synapse_types_shape_input, synapse_types_add_neuron_input, synapse_types_get_excitatory_input and synapse_types_get_inhibitory_input

Returns The number of CPU cycles

Return type int

get_n_synapse_type_bits()
Get the number of bits required to represent the synapse types

Returns the number of bits

Return type int

get_n_synapse_type_parameters()
Get the number of synapse type parameters

Returns the number of parameters

Return type int

get_n_synapse_types()

Get the number of synapse types supported

Returns The number of synapse types supported

Return type int

get_sdram_usage_per_neuron_in_bytes()

Get the SDRAM usage of the synapse type per neuron in bytes

Returns the number of bytes

Return type int

get_synapse_id_by_target(*target*)

Get the id of a synapse given the name

Returns The id of the synapse

Return type int

get_synapse_targets()

Get the target names of the synapse type

Returns an array of strings

Return type array of str

get_synapse_type_parameter_types()

Get the types of the synapse parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of `data_specification.enums.DataType`

get_synapse_type_parameters()

Get the synapse type parameters

Returns The parameters

Return type array of `spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter`

set_synapse_type_parameters(*parameters*, *vertex_slice*)

Sets any synapse type parameters. Override if there are changing variables in the synapse type parameters

Parameters

- **parameters** – the parameter values in a list of numpy arrays, ordered the same as get_synapse_type_parameters
- **vertex_slice** – The neurons to which the parameters apply

class spynnaker.pyNN.models.neuron.synapse_types.**SynapseTypeDualExponential**(*n_neurons*, *tau_syn_E*, *tau_syn_E2*, *tau_syn_I*, *initial_input_exc*, *initial_input_exc2*, *initial_input_inh*)

Bases: *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

get_n_cpu_cycles_per_neuron()

get_n_synapse_type_parameters()

get_n_synapse_types()

get_synapse_id_by_target(*target*)

get_synapse_targets()

get_synapse_type_parameter_types()

get_synapse_type_parameters(*obj*, **args*, ***kwargs*)

get_units(*variable*)

get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

isyn_exc

isyn_exc2

isyn_inh

tau_syn_E

tau_syn_E2

tau_syn_I

class spynnaker.pyNN.models.neuron.synapse_types.**SynapseTypeExponential**(*n_neurons*, *tau_syn_E*, *tau_syn_I*, *initial_input_exc=0.0*, *initial_input_inh=0.0*)

Bases: *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

get_n_cpu_cycles_per_neuron()

get_n_synapse_type_parameters()

get_n_synapse_types()

get_synapse_id_by_target(*target*)

get_synapse_targets()

get_synapse_type_parameter_types()

get_synapse_type_parameters(*obj*, **args*, ***kwargs*)

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

isyn_exc

isyn_inh

tau_syn_E

tau_syn_I

class spynnaker.pyNN.models.neuron.synapse_types.**SynapseTypeDelta**(*n_neurons*, *initial_input_exc*, *initial_input_inh*)

Bases: *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*

This represents a synapse type with two delta synapses

get_n_cpu_cycles_per_neuron()

get_n_synapse_type_parameters()

get_n_synapse_types()

get_synapse_id_by_target(*target*)

get_synapse_targets()

get_synapse_type_parameter_types()

get_synapse_type_parameters()

isyn_exc

isyn_inh

spynnaker.pyNN.models.neuron.threshold_types package

Submodules

spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type module

class spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.**AbstractThresholdT**

Bases: object

Represents types of threshold for a neuron (e.g. stochastic)

get_dtcm_usage_per_neuron_in_bytes()
Get the amount of DTCM used per neuron in bytes

Returns The number of bytes

Return type int

get_n_cpu_cycles_per_neuron()

Get the number of CPU cycles executed by threshold_type_is_above_threshold, per neuron

Returns The number of CPU cycles

Return type int

get_n_threshold_parameters()

Get the number of threshold parameters

Returns The number of threshold parameters

Return type int

get_sdram_usage_per_neuron_in_bytes()

Get the amount of SDRAM used per neuron in bytes

Returns The number of bytes

Return type int

get_threshold_parameter_types()

Get the types of the threshold parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_threshold_parameters()

Get the threshold parameters

Returns An array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

set_threshold_parameters(*parameters*, *vertex_slice*)

Sets the threshold type parameters. Override if there are any variables that change.

Parameters

- **parameters** – the parameter values in a list of numpy arrays, ordered the same as get_threshold_type_parameters
- **vertex_slice** – The neurons to which the parameters apply

spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic module

class spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic.**ThresholdT**

Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*

A stochastic threshold

du_th

get_n_cpu_cycles_per_neuron()

get_n_threshold_parameters()

get_threshold_parameter_types()

get_threshold_parameters()

tau_th

v_thresh

spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static module

class spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static.**ThresholdTypeStatic**(

Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

A threshold that is a static value

get_n_cpu_cycles_per_neuron()

get_n_threshold_parameters()
Get the number of threshold parameters

Returns The number of threshold parameters

Return type int

get_threshold_parameter_types()
Get the types of the threshold parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_threshold_parameters()
Get the threshold parameters

Returns An array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

v_thresh

Module contents

class spynnaker.pyNN.models.neuron.threshold_types.**AbstractThresholdType**
Bases: object

Represents types of threshold for a neuron (e.g. stochastic)

get_dtcm_usage_per_neuron_in_bytes()
Get the amount of DTCM used per neuron in bytes

Returns The number of bytes

Return type int

get_n_cpu_cycles_per_neuron()
Get the number of CPU cycles executed by threshold_type_is_above_threshold, per neuron

Returns The number of CPU cycles

Return type int

get_n_threshold_parameters()
Get the number of threshold parameters

Returns The number of threshold parameters

Return type int

get_sdram_usage_per_neuron_in_bytes()
Get the amount of SDRAM used per neuron in bytes

Returns The number of bytes

Return type int

get_threshold_parameter_types()
Get the types of the threshold parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_threshold_parameters()
Get the threshold parameters

Returns An array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

set_threshold_parameters(*parameters*, *vertex_slice*)
Sets the threshold type parameters. Override if there are any variables that change.

Parameters

- **parameters** – the parameter values in a list of numpy arrays, ordered the same as get_threshold_type_parameters
- **vertex_slice** – The neurons to which the parameters apply

class spynnaker.pyNN.models.neuron.threshold_types.**ThresholdTypeStatic**(*n_neurons*, *v_thresh*)

Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*

A threshold that is a static value

get_n_cpu_cycles_per_neuron()

get_n_threshold_parameters()
Get the number of threshold parameters

Returns The number of threshold parameters

Return type int

get_threshold_parameter_types()
Get the types of the threshold parameters

Returns A list of DataType objects, in the order of the parameters

Return type list of *data_specification.enums.DataType*

get_threshold_parameters()

Get the threshold parameters

Returns An array of parameters

Return type array of *spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter*

get_units(*variable*)

get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

v_thresh

class spynnaker.pyNN.models.neuron.threshold_types.**ThresholdTypeMaassStochastic**(*n_neurons*, *du_th*, *tau_th*, *v_thresh*)

Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*

A stochastic threshold

du_th

get_n_cpu_cycles_per_neuron()

get_n_threshold_parameters()

get_threshold_parameter_types()

get_threshold_parameters()

tau_th

v_thresh

Submodules

spynnaker.pyNN.models.neuron.abstract_population_vertex module

class spynnaker.pyNN.models.neuron.abstract_population_vertex.**AbstractPopulationVertex**(*n_neuro*
binary,
label,
max_ato
spikes_p
ring_buf
incoming_spik
model_n
neuron_mod
input_type
*synapse_*
threshold_type
additional_in
constraints=

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits*, *spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable*, *spynnaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*
spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraint
AbstractProvidesIncomingPartitionConstraints, *spynnaker.pyNN.models.abstract_models.abstract_population_initializable.AbstractPopulationInitializable*, *spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable*, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*, *pacman.model.abstract_classes.abstract_has_global_max_atoms.AbstractHasGlobalMaxAtoms*, *spinn_front_end_common.abstract_models.abstract_rewrites_data_specification.AbstractRewritesDataSpecification*, *spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set.AbstractReadParametersBeforeSet*, *spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

Underlying vertex model for Neural Populations.

BASIC_MALLOC_USAGE = 2

BYTES_TILL_START_OF_GLOBAL_PARAMETERS = 24

GSYN_EXCITATORY_RECORDING_REGION = 2

GSYN_INHIBITORY_RECORDING_REGION = 3

N_RECORDING_REGIONS = 4

RECORDING_REGION = {'gsyn_exc': 2, 'gsyn_inh': 3, 'spikes': 0, 'v': 1}

RUNTIME_SDP_PORT_SIZE = 4

SPIKE_RECORDING_REGION = 0

VARIABLE_LONG = {'gsyn_exc': 'gsyn_excitatory', 'gsyn_inh': 'gsyn_inhibitory', 'spikes': 'spikes', 'v': 'membrane volt

V_RECORDING_REGION = 1

add_pre_run_connection_holder(*connection_holder*, *edge*, *synapse_info*)

clear_connection_cache()

clear_recording(*variable*, *buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

create_machine_vertex(*obj*, **args*, ***kwargs*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

describe()

Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

generate_data_specification(*obj*, **args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_connections_from_machine(*transceiver*, *placement*, *edge*, *graph_mapper*, *routing_infos*, *synapse_info*, *machine_time_step*)

get_cpu_usage_for_atoms(*vertex_slice*)

get_data(*variable*, *n_machine_time_steps*, *placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)

Parameters

- **variable** –
- **n_machine_time_steps** –
- **placements** –
- **graph_mapper** –
- **buffer_manager** –
- **machine_time_step** –

Returns

get_dtcm_usage_for_atoms(*vertex_slice*)

get_incoming_partition_constraints(*partition*)
Get constraints to be added to the given edge that goes in to a vertex of this vertex

Parameters

- **partition** (`pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition`) – An partition that goes in to this vertex
- **partition** – partition that goes into this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint* Gets the constraints for partitions going into this vertex

Returns list of constraints

get_maximum_delay_supported_in_ms(*machine_time_step*)

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters

- **partition** – An edge that comes out of this vertex
- **partition** – the partition that leaves this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint* Gets the constraints for partitions going out of this vertex

Returns list of constraints

get_recordable_variables()
Returns a list of the variables this models is expected to collect

get_resources_used_by_atoms(*obj*, **args*, ***kwargs*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

get_sdram_usage_for_atoms(*vertex_slice*, *graph*, *machine_time_step*)

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

get_value(*key*)
Get a property Get a property of the overall model

initialize(*variable*, *value*)
Set the initial value of one of the state variables of the neurons in this population.

input_type

is_recording(*variable*)
Determines if variable is being recorded

Returns True if vavriable are being recorded, False otherwise

Return type bool

is_recording_spikes()
Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

mark_regions_reloaded()
Indicate that the regions have been reloaded

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

none_pynn_default_parameters = {'incoming_spike_buffer_size': None, 'label': None, 'spikes_per_second': None

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)
Read the parameters from the machine before any are changed

Parameters

- **transceiver** – the SpinnMan interface
- **placement** – the placement of a vertex
- **vertex_slice** – the slice of atoms for this vertex

regenerate_data_specification(*obj*, **args*, ***kwargs*)
Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_mapping
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

requires_memory_regions_to_be_reloaded()
Return true if any data region needs to be reloaded

Return type bool

ring_buffer_sigma

set_recording(*variable*, *new_state=True*)
Sets v to being recorded

set_recording_spikes(*new_state=True*)
Sets spikes to being recorded

set_synapse_dynamics(*synapse_dynamics*)

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property of the overall model

spikes_per_second

synapse_dynamics

synapse_type

weight_scale

spynnaker.pyNN.models.neuron.connection_holder module

class spynnaker.pyNN.models.neuron.connection_holder.**ConnectionHolder**(*data_items_to_return*, *as_list*, *n_pre_atoms*, *n_post_atoms*, *connections=None*, *fixed_values=None*, *notify=None*)

Bases: object

Holds a set of connections to be returned in a PyNN-specific format

Parameters

- **data_items_to_return** – A list of data fields to be returned
- **as_list** – True if the data will be returned as a list, False if it is to be returned as a matrix (or series of matrices)
- **n_pre_atoms** – The number of atoms in the pre-vertex
- **n_post_atoms** – The number of atoms in the post-vertex
- **connections** – Any initial connections, as a numpy structured array of source, target, weight and delay
- **fixed_values** – A list of tuples of field names and fixed values to be appended to the other fields per connection, formatted as [(field_name, value), ...]. Note that if the field is to be returned, the name must also appear in data_items_to_return, which determines the order of items in the result
- **notify** – A callback to call when the connections have all been added. This should accept a single parameter, which will contain the data requested

add_connections(*connections*)

Add connections to the holder to be returned

Parameters **connections** – The connection to add, as a numpy structured array of source, target, weight and delay

connections

The connections stored

finish()

Finish adding connections

spynnaker.pyNN.models.neuron.population_machine_vertex module

class spynnaker.pyNN.models.neuron.population_machine_vertex.**PopulationMachineVertex**(*resources_r is_recordin min- i- mum_buffe buffered_sa la- bel, con- straints=No*

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*, *spinn_front_end_common.interface.profiling.abstract_has_profile_data.AbstractHasProfileData*

class EXTRA_PROVENANCE_DATA_ENTRIES

Bases: enum.Enum

BUFFER_OVERFLOW_COUNT = 2

CURRENT_TIMER_TIC = 3

PRE_SYNAPTIC_EVENT_COUNT = 0

SATURATION_COUNT = 1

N_ADDITIONAL_PROVENANCE_DATA_ITEMS = 4

PROFILE_TAG_LABELS = {0: 'TIMER', 1: 'DMA_READ', 2: 'INCOMING_SPIKE', 3: 'PROCESS_FIXED_SYNAPSE

get_minimum_buffer_sdram_usage()

Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)

Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_profile_data(*transceiver*, *placement*)

Get the profile data recorded during simulation

Return type *spinn_front_end_common.interface.profiling.profile_data.ProfileData*

get_provenance_data_from_machine(*transceiver*, *placement*)

get_recorded_region_ids()

Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)

Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

is_recording()

Deduce if the recorder is actually recording

resources_required

The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

spynnaker.pyNN.models.neuron.synaptic_manager module

class spynnaker.pyNN.models.neuron.synaptic_manager.**SynapticManager**(*synapse_type*, *ring_buffer_sigma*, *spikes_per_second*, *config*, *population_table_type=None*, *synapse_io=None*)

Bases: object

Deals with synapses

add_pre_run_connection_holder(*connection_holder*, *edge*, *synapse_info*)

clear_connection_cache()

get_connections_from_machine(*transceiver*, *placement*, *machine_edge*, *graph_mapper*, *routing_infos*, *synapse_info*, *machine_time_step*)

get_dtcm_usage_in_bytes()

get_incoming_partition_constraints()

get_maximum_delay_supported_in_ms(*machine_time_step*)

get_n_cpu_cycles()

get_number_of_mallocs_used_by_dsg()

get_sdram_usage_in_bytes(*vertex_slice*, *in_edges*, *machine_time_step*)

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)

regenerate_data_specification(*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *vertex_slice*)

ring_buffer_sigma

spikes_per_second

synapse_dynamics

synapse_type

vertex_executable_suffix

write_data_spec(*spec*, *application_vertex*, *post_vertex_slice*, *machine_vertex*, *placement*, *machine_graph*, *application_graph*, *routing_info*, *graph_mapper*, *input_type*, *machine_time_step*)

Module contents

class spynnaker.pyNN.models.neuron.**AbstractPopulationVertex**(*n_neurons*, *binary*, *label*, *max_atoms_per_core*, *spikes_per_second*, *ring_buffer_sigma*, *incoming_spike_buffer_size*, *model_name*, *neuron_model*, *input_type*, *synapse_type*, *threshold_type*, *additional_input=None*, *constraints=None*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification, spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary, spynnaker.pyNN.models.abstract_models.abstract_contains_units.AbstractContainsUnits, spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable, spynnaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable, spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraint AbstractProvidesIncomingPartitionConstraints, spynnaker.pyNN.models.abstract_models.abstract_population_initializable.AbstractPopulationInitializable, spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable, spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun, pacman.model.abstract_classes.abstract_has_global_max_atoms.AbstractHasGlobalMaxAtoms, spinn_front_end_common.abstract_models.abstract_rewrites_data_specification.AbstractRewritesDataSpecification, spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set.AbstractReadParametersBeforeSet, spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses, spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl

Underlying vertex model for Neural Populations.

BASIC_MALLOC_USAGE = 2

BYTES_TILL_START_OF_GLOBAL_PARAMETERS = 24

GSYN_EXCITATORY_RECORDING_REGION = 2

GSYN_INHIBITORY_RECORDING_REGION = 3

N_RECORDING_REGIONS = 4

RECORDING_REGION = {'gsyn_exc': 2, 'gsyn_inh': 3, 'spikes': 0, 'v': 1}

RUNTIME_SDP_PORT_SIZE = 4

SPIKE_RECORDING_REGION = 0

VARIABLE_LONG = {'gsyn_exc': 'gsyn_excitatory', 'gsyn_inh': 'gsyn_inhibitory', 'spikes': 'spikes', 'v': 'membrane volt

V_RECORDING_REGION = 1

add_pre_run_connection_holder(*connection_holder*, *edge*, *synapse_info*)

clear_connection_cache()

clear_recording(*variable*, *buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

create_machine_vertex(*obj*, **args*, ***kwargs*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

describe()

Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to

- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_connections_from_machine(*transceiver*, *placement*, *edge*, *graph_mapper*, *routing_infos*, *synapse_info*, *machine_time_step*)

get_cpu_usage_for_atoms(*vertex_slice*)

get_data(*variable*, *n_machine_time_steps*, *placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)

Parameters

- **variable** –
- **n_machine_time_steps** –
- **placements** –
- **graph_mapper** –
- **buffer_manager** –
- **machine_time_step** –

Returns

get_dtcm_usage_for_atoms(*vertex_slice*)

get_incoming_partition_constraints(*partition*)
Get constraints to be added to the given edge that goes in to a vertex of this vertex

Parameters

- **partition** (`pacman.utilities.utility_objs.outgoing_partition.OutgoingPartition`) – An partition that goes in to this vertex
- **partition** – partition that goes into this vertex

Returns A list of constraints

Return type list of *`pacman.model.constraints.abstract_constraint.AbstractConstraint`* Gets the constraints for partitions going into this vertex

Returns list of constraints

get_maximum_delay_supported_in_ms(*machine_time_step*)

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters

- **partition** – An edge that comes out of this vertex
- **partition** – the partition that leaves this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint* Gets the constraints for partitions going out of this vertex

Returns list of constraints

get_recordable_variables()
Returns a list of the variables this models is expected to collect

get_resources_used_by_atoms(*obj*, **args*, ***kwargs*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

get_sdram_usage_for_atoms(*vertex_slice*, *graph*, *machine_time_step*)

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

get_units(*variable*)
get units for a given variable

Parameters **variable** – the variable to find units from

Returns the units as a string.

get_value(*key*)
Get a property Get a property of the overall model

initialize(*variable*, *value*)
Set the initial value of one of the state variables of the neurons in this population.

input_type

is_recording(*variable*)
Determines if variable is being recorded

Returns True if vavriable are being recorded, False otherwise

Return type bool

is_recording_spikes()
Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

mark_regions_reloaded()
Indicate that the regions have been reloaded

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

none_pynn_default_parameters = {'incoming_spike_buffer_size': None, 'label': None, 'spikes_per_second': None

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)
Read the parameters from the machine before any are changed

Parameters

- **transceiver** – the SpinnMan interface
- **placement** – the placement of a vertex
- **vertex_slice** – the slice of atoms for this vertex

regenerate_data_specification(*obj*, **args*, ***kwargs*)
Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_mapping
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

requires_memory_regions_to_be_reloaded()
Return true if any data region needs to be reloaded

Return type bool

ring_buffer_sigma

set_recording(*variable*, *new_state=True*)
Sets v to being recorded

set_recording_spikes(*new_state=True*)
Sets spikes to being recorded

set_synapse_dynamics(*synapse_dynamics*)

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property of the overall model

spikes_per_second

synapse_dynamics

synapse_type

weight_scale

class spynnaker.pyNN.models.neuron.**ConnectionHolder**(*data_items_to_return*, *as_list*, *n_pre_atoms*, *n_post_atoms*, *connections=None*, *fixed_values=None*, *notify=None*)

Bases: object

Holds a set of connections to be returned in a PyNN-specific format

Parameters

- **data_items_to_return** – A list of data fields to be returned
- **as_list** – True if the data will be returned as a list, False if it is to be returned as a matrix (or series of matrices)
- **n_pre_atoms** – The number of atoms in the pre-vertex
- **n_post_atoms** – The number of atoms in the post-vertex
- **connections** – Any initial connections, as a numpy structured array of source, target, weight and delay
- **fixed_values** – A list of tuples of field names and fixed values to be appended to the other fields per connection, formatted as [(field_name, value), ...]. Note that if the field is to be returned, the name must also appear in data_items_to_return, which determines the order of items in the result
- **notify** – A callback to call when the connections have all been added. This should accept a single parameter, which will contain the data requested

add_connections(*connections*)

Add connections to the holder to be returned

Parameters **connections** – The connection to add, as a numpy structured array of source, target, weight and delay

connections

The connections stored

finish()

Finish adding connections

class spynnaker.pyNN.models.neuron.**SynapticManager**(*synapse_type*, *ring_buffer_sigma*, *spikes_per_second*, *config*, *population_table_type=None*, *synapse_io=None*)

Bases: object

Deals with synapses

add_pre_run_connection_holder(*connection_holder*, *edge*, *synapse_info*)

clear_connection_cache()

get_connections_from_machine(*transceiver*, *placement*, *machine_edge*, *graph_mapper*, *routing_infos*, *synapse_info*, *machine_time_step*)

get_dtcm_usage_in_bytes()

get_incoming_partition_constraints()

get_maximum_delay_supported_in_ms(*machine_time_step*)

get_n_cpu_cycles()

get_number_of_mallocs_used_by_dsg()

get_sdram_usage_in_bytes(*vertex_slice*, *in_edges*, *machine_time_step*)

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)

regenerate_data_specification(*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *vertex_slice*)

ring_buffer_sigma

spikes_per_second

synapse_dynamics

synapse_type

vertex_executable_suffix

write_data_spec(*spec*, *application_vertex*, *post_vertex_slice*, *machine_vertex*, *placement*, *machine_graph*, *application_graph*, *routing_info*, *graph_mapper*, *input_type*, *machine_time_step*)

class spynnaker.pyNN.models.neuron.**PopulationMachineVertex**(*resources_required*, *is_recording*, *minimum_buffer_sdram_usage*, *buffered_sdram_per_timestep*, *label*, *constraints=None*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*, *spinn_front_end_common.interface.profiling.abstract_has_profile_data.AbstractHasProfileData*

class EXTRA_PROVENANCE_DATA_ENTRIES

Bases: enum.Enum

BUFFER_OVERFLOW_COUNT = 2

CURRENT_TIMER_TIC = 3

PRE_SYNAPTIC_EVENT_COUNT = 0

SATURATION_COUNT = 1

N_ADDITIONAL_PROVENANCE_DATA_ITEMS = 4

PROFILE_TAG_LABELS = {0: 'TIMER', 1: 'DMA_READ', 2: 'INCOMING_SPIKE', 3: 'PROCESS_FIXED_SYNAPSE

get_minimum_buffer_sdram_usage()

Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)

Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_profile_data(*transceiver*, *placement*)
Get the profile data recorded during simulation

Return type *spinn_front_end_common.interface.profiling.profile_data.ProfileData*

get_provenance_data_from_machine(*transceiver*, *placement*)

get_recorded_region_ids()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)
Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

is_recording()
Deduce if the recorder is actually recording

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

spynnaker.pyNN.models.spike_source package

Submodules

spynnaker.pyNN.models.spike_source.spike_source_array module

class spynnaker.pyNN.models.spike_source.spike_source_array.**SpikeSourceArray**(*n_neurons*, *spike_times=None*, *port=None*, *tag=None*, *ip_address=None*, *board_address=None*, *max_on_chip_memory_*
space_before_notificati
constraints=None, *label=None*, *spike_recorder_buffer_*
buffer_size_before_rece

Bases: *spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.ReverseIpTagMultiCastSource*, *spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable*, *spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable*,

spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

Model for play back of spikes

DEFAULT1 = 1048576

SPIKE_RECORDING_REGION_ID = 0

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

default_parameters = {'spike_times': None}

describe()

Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

static get_max_atoms_per_core()

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)

Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes()

Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

mark_no_changes()

Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

none_pynn_default_parameters = {'tag': None, 'board_address': None, 'label': None, 'spike_recorder_buffer_size

requires_mapping

True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

static set_model_max_atoms_per_core(*new_value=9223372036854775807*)

set_recording_spikes(*new_state=True*)
Sets spikes to being recorded

spike_times
The spike times of the spike source array

spynnaker.pyNN.models.spike_source.spike_source_from_file module

class spynnaker.pyNN.models.spike_source.spike_source_from_file.**SpikeSourceFromFile**(*n_neurons, spike_time_f ma- chine_time_s timescale_fa port=None, tag=None, ip_address= board_addre min_atom=N max_atom=N min_time=N max_time=N max_on_chip con- straints=Non split_value= la- bel='SpikeS*

Bases: *spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray*

SpikeSourceArray that works from a file

spike_times

spynnaker.pyNN.models.spike_source.spike_source_poisson module

class spynnaker.pyNN.models.spike_source.spike_source_poisson.**SpikeSourcePoisson**(*n_neurons, con- straints=None, la- bel=None, rate=1.0, start=0.0, du- ra- tion=None, seed=None*)

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.*

AbstractProvidesOutgoingPartitionConstraints, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*, *spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set.AbstractReadParametersBeforeSet*, *spinn_front_end_common.abstract_models.abstract_rewrites_data_specification.AbstractRewritesDataSpecification*, *spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

A Poisson Spike source object

DEFAULT_MAX_ATOMS_PER_CORE = 500

SPIKE_RECORDING_REGION_ID = 0

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

create_machine_vertex(*obj*, **args*, ***kwargs*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

default_parameters = {'duration': None, 'start': 0.0, 'rate': 1.0}

describe()

Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

duration

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

static **get_cpu_usage_for_atoms**()

static **get_dtcm_usage_for_atoms**()

static **get_max_atoms_per_core**()

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

static **get_params_bytes**(*vertex_slice*)
Gets the size of the poisson parameters in bytes

Parameters **vertex_slice** –

get_resources_used_by_atoms(*obj*, **args*, ***kwargs*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

get_sdram_usage_for_atoms(*vertex_slice*)
calculates total sdram usage for a set of atoms

Parameters **vertex_slice** – the atoms to calculate sdram usage for

Returns sdram usage as a number of bytes

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes()
Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

mark_no_changes()
Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

mark_regions_reloaded()
Indicate that the regions have been reloaded

n_atoms

none_pynn_default_parameters = {'seed': None, 'label': None, 'constraints': None}

rate

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)
Read the parameters from the machine before any are changed

Parameters

- **transceiver** – the SpinnMan interface
- **placement** – the placement of a vertex
- **vertex_slice** – the slice of atoms for this vertex

regenerate_data_specification(*obj*, **args*, ***kwargs*)
Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_mapping
True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

requires_memory_regions_to_be_reloaded()
Return true if any data region needs to be reloaded

Return type bool

reserve_memory_regions(*spec*, *placement*, *graph_mapper*)
Reserve memory regions for poisson source parameters and output buffer.

Parameters

- **spec** – the data specification writer
- **placement** – the location this vertex resides on in the machine
- **graph_mapper** – the mapping between app and machine graphs

Returns None

seed

static set_model_max_atoms_per_core(*new_value=500*)

set_recording_spikes(*new_state=True*)
Sets spikes to being recorded

set_value(*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

start

spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex module

class spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.**SpikeSourcePoiss**

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*

class POISSON_SPIKE_SOURCE_REGIONS

Bases: enum.Enum

POISSON_PARAMS_REGION = 1

PROVENANCE_REGION = 3

SPIKE_HISTORY_REGION = 2

SYSTEM_REGION = 0

get_minimum_buffer_sdram_usage()

Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space(*buffer_space*, *machine_time_step*)

Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_region_ids()

Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)

Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

is_recording()
Deduce if the recorder is actually recording

resources_required
The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

Module contents

class spynnaker.pyNN.models.spike_source.**SpikeSourceArray**(*n_neurons*, *spike_times=None*, *port=None*, *tag=None*, *ip_address=None*, *board_address=None*, *max_on_chip_memory_usage_for_spikes_in_bytes=*, *space_before_notification=640*, *constraints=None*, *label=None*, *spike_recorder_buffer_size=1048576*, *buffer_size_before_receive=524288*)

Bases: *spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.ReverseIpTagMultiCastSource*, *spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable*, *spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable*, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

Model for play back of spikes

DEFAULT1 = 1048576

SPIKE_RECORDING_REGION_ID = 0

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)
clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

default_parameters = {'spike_times': None}

describe()
Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

static get_max_atoms_per_core()

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes()

Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

mark_no_changes()

Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

none_pynn_default_parameters = {'tag': None, 'board_address': None, 'label': None, 'spike_recorder_buffer_size

requires_mapping

True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

static set_model_max_atoms_per_core(*new_value=9223372036854775807*)

set_recording_spikes(*new_state=True*)

Sets spikes to being recorded

spike_times

The spike times of the spike source array

class spynnaker.pyNN.models.spike_source.**SpikeSourceFromFile**(*n_neurons*, *spike_time_file*, *machine_time_step*, *timescale_factor*, *port=None*, *tag=None*, *ip_address=None*, *board_address=None*, *min_atom=None*, *max_atom=None*, *min_time=None*, *max_time=None*, *max_on_chip_memory_usage_for_spikes_in_by*, *constraints=None*, *split_value='t'*, *label='SpikeSourceArray'*)

Bases: *spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray*

SpikeSourceArray that works from a file

spike_times

class spynnaker.pyNN.models.spike_source.**SpikeSourcePoisson**(*n_neurons*, *constraints=None*, *label=None*, *rate=1.0*, *start=0.0*, *duration=None*, *seed=None*)

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionConstraints*, *spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun*, *spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set.AbstractReadParametersBeforeSet*, *spinn_front_end_common.abstract_models.abstract_rewrites_data_specification.AbstractRewritesDataSpecification*, *spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable*, *spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl*

A Poisson Spike source object

DEFAULT_MAX_ATOMS_PER_CORE = 500

SPIKE_RECORDING_REGION_ID = 0

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

create_machine_vertex(*obj*, **args*, ***kwargs*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

default_parameters = {'duration': None, 'start': 0.0, 'rate': 1.0}

describe()

Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

duration

generate_data_specification(*obj*, **args*, ***kwargs*)
Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

static **get_cpu_usage_for_atoms**()

static **get_dtcm_usage_for_atoms**()

static **get_max_atoms_per_core**()

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

static **get_params_bytes**(*vertex_slice*)
Gets the size of the poisson parameters in bytes

Parameters **vertex_slice** –

get_resources_used_by_atoms(*obj*, **args*, ***kwargs*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

get_sdram_usage_for_atoms(*vertex_slice*)
calculates total sdram usage for a set of atoms

Parameters **vertex_slice** – the atoms to calculate sdram usage for

Returns sdram usage as a number of bytes

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes()

Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

mark_no_changes()

Marks the point after which changes are reported. Immediately after calling this method, requires_mapping should return False.

mark_regions_reloaded()

Indicate that the regions have been reloaded

n_atoms

none_pynn_default_parameters = {'seed': None, 'label': None, 'constraints': None}

rate

read_parameters_from_machine(*transceiver*, *placement*, *vertex_slice*)

Read the parameters from the machine before any are changed

Parameters

- **transceiver** – the SpinnMan interface
- **placement** – the placement of a vertex
- **vertex_slice** – the slice of atoms for this vertex

regenerate_data_specification(*obj*, **args*, ***kwargs*)

Regenerate the data specification, only generating regions that have changed and need to be reloaded

requires_mapping

True if changes that have been made require that mapping be performed. Note that this should return True the first time it is called, as the vertex must require mapping as it has been created!

requires_memory_regions_to_be_reloaded()

Return true if any data region needs to be reloaded

Return type bool

reserve_memory_regions(*spec*, *placement*, *graph_mapper*)

Reserve memory regions for poisson source parameters and output buffer.

Parameters

- **spec** – the data specification writer
- **placement** – the location this vertex resides on in the machine
- **graph_mapper** – the mapping between app and machine graphs

Returns None

seed

static **set_model_max_atoms_per_core** (*new_value=500*)

set_recording_spikes (*new_state=True*)
Sets spikes to being recorded

set_value (*key*, *value*)
Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

start

class spynnaker.pyNN.models.spike_source.**SpikeSourcePoissonMachineVertex** (*resources_required*, *is_recording*, *minimum_buffer_sdram*, *buffered_sdram_per_timestep*, *constraints=None*, *label=None*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl*, *spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable*

class **POISSON_SPIKE_SOURCE_REGIONS**
Bases: enum.Enum

POISSON_PARAMS_REGION = 1

PROVENANCE_REGION = 3

SPIKE_HISTORY_REGION = 2

SYSTEM_REGION = 0

get_minimum_buffer_sdram_usage ()
Get the minimum amount of SDRAM to reserve for buffers

get_n_timesteps_in_buffer_space (*buffer_space*, *machine_time_step*)
Get the number of timesteps that can be stored fully in the given buffer space in bytes

Parameters

- **buffer_space** – The buffer space in bytes
- **machine_time_step** – The size of each time step

Returns The number of time steps that can be stored in the buffer

Return type int

get_recorded_region_ids ()
Get the recording region ids that have been recorded using buffering

Returns The region numbers that have active recording

Return type iterable of int

get_recording_region_base_address(*txrx*, *placement*)

Get the recording region base address

Parameters

- **txrx** – the SpiNNMan instance
- **placement** – the placement object of the core to find the address of

Returns the base address of the recording region

is_recording()

Deduce if the recorder is actually recording

resources_required

The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

spynnaker.pyNN.models.utility_models package

Submodules

spynnaker.pyNN.models.utility_models.delay_block module

class spynnaker.pyNN.models.utility_models.delay_block.**DelayBlock**(*n_delay_stages*, *delay_per_stage*, *vertex_slice*)

Bases: object

A block of delays for a vertex

add_delay(*source_id*, *stage*)

delay_block

spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex module

class spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex.**DelayExtensionMachi**

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*

class EXTRA_PROVENANCE_DATA_ENTRIES

Bases: enum.Enum

N_BUFFER_OVERFLOWS = 4

N_DELAYS = 5

N_PACKETS_ADDED = 2

N_PACKETS_PROCESSED = 1

N_PACKETS_RECEIVED = 0

N_PACKETS_SENT = 3

get_provenance_data_from_machine(*transceiver*, *placement*)

resources_required

The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

spynnaker.pyNN.models.utility_models.delay_extension_vertex module

class spynnaker.pyNN.models.utility_models.delay_extension_vertex.**DelayExtensionVertex**(*n_neuro de- lay_per_ source_v ma- chine_ti timescal con- straints= la- bel='De*

Bases: *pacman.model.graphs.application.application_vertex.ApplicationVertex*, *spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification*, *spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons* *spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition.AbstractProvidesNKeysForPartition*

Provide delays to incoming spikes in multiples of the maximum delays of a neuron (typically 16 or 32)

Creates a new DelayExtension Object.

add_delays(*vertex_slice*, *source_ids*, *stages*)

Add delayed connections for a given vertex slice

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (*data_specification.DataSpecificationGenerator*) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()
Get the binary name to be run for this vertex

Return type str

get_binary_start_type()
Get the start type of the binary to be run

Return type `spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType`

get_cpu_usage_for_atoms(*vertex_slice*)

get_dtcm_usage_for_atoms(*vertex_slice*)

get_n_keys_for_partition(*partition*, *graph_mapper*)

get_outgoing_partition_constraints(*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms(*vertex_slice*)
Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (`pacman.model.graph.slice.Slice`) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

get_sdram_usage_for_atoms()

n_atoms
The number of atoms in the vertex

Returns The number of atoms

Return type int

n_delay_stages
The maximum number of delay stages required by any connection out of this delay extension vertex

source_vertex

write_delay_parameters(*spec*, *vertex_slice*, *key*, *incoming_key*, *incoming_mask*, *n_vertices*, *machine_time_step*, *time_scale_factor*)
Generate Delay Parameter data

write_setup_info(*spec*, *machine_time_step*, *time_scale_factor*)

spynnaker.pyNN.models.utility_models.spike_injector module

class spynnaker.pyNN.models.utility_models.spike_injector.**SpikeInjector**(*n_neurons*, *label='spikeInjector'*, *port=None*, *virtual_key=None*, *spike_buffer_max_size=None*, *buffer_size_before_receive=No*, *time_between_requests=None*, *buffer_notification_ip_address=*, *buffer_notification_port=None*)

Bases: *spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.ReverseIpTagMultiCastSource*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons*, *spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable*, *spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable*

An Injector of Spikes for PyNN populations. This only allows the user to specify the virtual_key of the population to identify the population

SPIKE_RECORDING_REGION_ID = 0

clear_spike_recording(*buffer_manager*, *placements*, *graph_mapper*)

clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

default_parameters = {'virtual_key': None, 'port': None, 'label': 'spikeInjector'}

describe()

Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_spikes(*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)

Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes()

Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

port

set_recording_spikes(*new_state=True*)

Sets spikes to being recorded

virtual_key

Module contents

class spynnaker.pyNN.models.utility_models.**DelayBlock**(*n_delay_stages*, *delay_per_stage*, *vertex_slice*)

Bases: object

A block of delays for a vertex

add_delay(*source_id*, *stage*)

delay_block

class spynnaker.pyNN.models.utility_models.**DelayExtensionMachineVertex**(*resources_required*, *label*, *constraints=None*)

Bases: *pacman.model.graphs.machine.machine_vertex.MachineVertex*, *spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl ProvidesProvenanceDataFromMachineImpl*

class EXTRA_PROVENANCE_DATA_ENTRIES

Bases: enum.Enum

N_BUFFER_OVERFLOWS = 4

N_DELAYS = 5

N_PACKETS_ADDED = 2

N_PACKETS_PROCESSED = 1

N_PACKETS_RECEIVED = 0

N_PACKETS_SENT = 3

get_provenance_data_from_machine(*transceiver*, *placement*)

resources_required

The resources required by the vertex

Return type *pacman.model.resources.ResourceContainer*

class spynnaker.pyNN.models.utility_models.**DelayExtensionVertex**(*n_neurons*, *delay_per_stage*, *source_vertex*, *machine_time_step*, *timescale_factor*, *constraints=None*, *label='DelayExtension'*)

Bases: pacman.model.graphs.application.application_vertex.ApplicationVertex, spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification, spinn_front_end_common.abstract_models.abstract_has_associated_binary.AbstractHasAssociatedBinary, spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition.AbstractProvidesNKeysForPartition

Provide delays to incoming spikes in multiples of the maximum delays of a neuron (typically 16 or 32)

Creates a new DelayExtension Object.

add_delays(*vertex_slice*, *source_ids*, *stages*)

Add delayed connections for a given vertex slice

create_machine_vertex(*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** – The slice of atoms that the machine vertex will cover
- **resources_required** – the resources used by the machine vertex
- **constraints** – Constraints to be passed on to the machine vertex

generate_data_specification(*obj*, **args*, ***kwargs*)

Generate a data specification

Parameters

- **spec** (data_specification.DataSpecificationGenerator) – The data specification to write to
- **placement** – the placement object this spec is associated with

Return type None

get_binary_file_name()

Get the binary name to be run for this vertex

Return type str

get_binary_start_type()

Get the start type of the binary to be run

Return type spinn_front_end_common.utilities.utility_objs.executable_start_type.ExecutableType

get_cpu_usage_for_atoms(*vertex_slice*)

get_dtcm_usage_for_atoms(*vertex_slice*)

get_n_keys_for_partition(*partition*, *graph_mapper*)

get_outgoing_partition_constraints(*partition*)

Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_resources_used_by_atoms(*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (pacman.model.graph.slice.Slice) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *pacman.model.resources.ResourceContainer*

Raises **None** – this method does not raise any known exception

get_sdram_usage_for_atoms()

n_atoms

The number of atoms in the vertex

Returns The number of atoms

Return type int

n_delay_stages

The maximum number of delay stages required by any connection out of this delay extension vertex

source_vertex

write_delay_parameters(*spec*, *vertex_slice*, *key*, *incoming_key*, *incoming_mask*, *n_vertices*, *machine_time_step*, *time_scale_factor*)

Generate Delay Parameter data

write_setup_info(*spec*, *machine_time_step*, *time_scale_factor*)

class spynnaker.pyNN.models.utility_models.**SpikeInjector**(*n_neurons*, *label='spikeInjector'*, *port=None*, *virtual_key=None*, *spike_buffer_max_size=None*, *buffer_size_before_receive=None*, *time_between_requests=None*, *buffer_notification_ip_address=None*, *buffer_notification_port=None*)

Bases: *spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.ReverseIpTagMultiCastSource*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartitionCons* *spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable*, *spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable*

An Injector of Spikes for PyNN populations. This only allows the user to specify the virtual_key of the population to identify the population

SPIKE_RECORDING_REGION_ID = **0**

clear_spike_recording (*buffer_manager*, *placements*, *graph_mapper*)
clears the recorded data from the object

Parameters

- **buffer_manager** – the buffer manager object
- **placements** – the placements object
- **graph_mapper** – the graph mapper object

Return type None

default_parameters = {'virtual_key': None, 'port': None, 'label': 'spikeInjector'}

describe ()
Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

get_outgoing_partition_constraints (*partition*)
Get constraints to be added to the given edge that comes out of this vertex

Parameters **partition** – An edge that comes out of this vertex

Returns A list of constraints

Return type list of *pacman.model.constraints.abstract_constraint.AbstractConstraint*

get_spikes (*placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)
Get the recorded spikes from the object

Parameters

- **placements** – the placements object
- **graph_mapper** – the graph mapper object
- **buffer_manager** – the buffer manager object
- **machine_time_step** – the time step of the simulation

Returns A numpy array of 2-element arrays of (neuron_id, time) ordered by time

is_recording_spikes ()
Determines if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

port

set_recording_spikes (*new_state=True*)
Sets spikes to being recorded

virtual_key

Submodules

spynnaker.pyNN.models.pynn_population_common module

class `spynnaker.pyNN.models.pynn_population_common.`**`PyNNPopulationCommon`**(*spinnaker_control*, *size*, *vertex*, *structure*, *initial_values*)

Bases: `object`

`add_placement_constraint`(*x*, *y*, *p=None*)

Add a placement constraint

Parameters

- **`x`** (*`int`*) – The x-coordinate of the placement constraint
- **`y`** (*`int`*) – The y-coordinate of the placement constraint
- **`p`** (*`int`*) – The processor id of the placement constraint (optional)

`all`()

Iterator over cell ids on all nodes.

`can_record`(*variable*)

Determine whether *variable* can be recorded from this population.

`conductance_based`

True if the population uses conductance inputs

static `create_label`(*model_label*, *pop_level_label*)

helper method for choosing a label from model and population levels

Parameters

- **`model_label`** – the model level label
- **`pop_level_label`** – the pop level label

Returns the new model level label

`first_id`

`get`(*parameter_name*, *gather=False*)

Get the values of a parameter for every local cell in the population.

`get_spike_counts`(*spikes*, *gather=True*)

Return the number of spikes for each neuron.

`id_to_index`(*id*)

Given the ID(s) of cell(s) in the Population, return its (their) index (order in the Population).

`id_to_local_index`(*cell_id*)

Given the ID(s) of cell(s) in the Population, return its (their) index (order in the Population), counting only cells on the local MPI node.

`initialize`(*variable*, *value*)

Set the initial value of one of the state variables of the neurons in this population.

inject (*current_source*)
Connect a current source to all cells in the Population.

label
The label of the population

last_id

local_size
The number of local cells

mark_no_changes ()

requires_mapping

set (*parameter*, *value=None*)
Set one or more parameters for every cell in the population.

param can be a dict, in which case value should not be supplied, or a string giving the parameter name, in which case value is the parameter value. value can be a numeric value, or list of such (e.g. for setting spike times):

```
p.set("tau_m", 20.0).
p.set({'tau_m':20, 'v_rest':-65})
```

Parameters

- **parameter** – the parameter to set
- **value** – the value of the parameter to set.

set_constraint (*constraint*)
Apply a constraint to a population that restricts the processor onto which its atoms will be placed.

set_mapping_constraint (*constraint_dict*)
Add a placement constraint - for backwards compatibility

Parameters **constraint_dict** (*dict of str->int*) – A dictionary containing "x", "y" and optionally "p" as keys, and ints as values

set_model_based_max_atoms_per_core (*new_value*)
Supports the setting of each models max atoms per core parameter

Parameters **new_value** – the new value for the max atoms per core.

size
The number of neurons in the population

structure
Return the structure for the population.

spynnaker.pyNN.models.pynn_projection_common module

class spynnaker.pyNN.models.pynn_projection_common.**PyNNProjectionCommon**(*spinnaker_control*, *connector*, *synapse_dynamics_stdp*, *target*, *pre_synaptic_population*, *post_synaptic_population*, *rng*, *machine_time_step*, *user_max_delay*, *label*, *time_scale_factor*)

Bases: object

A container for all the connections of a given type (same synapse type and plasticity mechanisms) between two populations, together with methods to set parameters of those connections, including of plasticity mechanisms.

mark_no_changes()

requires_mapping

size(*gather=True*)

Return the total number of connections. - only local connections, if gather is False, - all connections, if gather is True (default)

spynnaker.pyNN.models.recording_common module

class spynnaker.pyNN.models.recording_common.**RecordingCommon**(*population*, *sampling_interval=None*)

Bases: object

object to hold recording behaviour

Parameters

- **population** – the population to record for
- **simulator** – the spinnaker control class

sampling_interval

forced by the public nature of pynn variables

Returns

Module contents

spynnaker.pyNN.overridden_pacman_functions package

Submodules

spynnaker.pyNN.overridden_pacman_functions.graph_edge_filter module

class spynnaker.pyNN.overridden_pacman_functions.graph_edge_filter.**GraphEdgeFilter**
Bases: object

Removes graph edges that aren't required

spynnaker.pyNN.overridden_pacman_functions.graph_edge_weight_updater module

class spynnaker.pyNN.overridden_pacman_functions.graph_edge_weight_updater.**GraphEdgeWeightUpd**
Bases: object

Removes graph edges that aren't required

spynnaker.pyNN.overridden_pacman_functions.spynnaker_data_specification_writer module

class spynnaker.pyNN.overridden_pacman_functions.spynnaker_data_specification_writer.**Spynnake**
Bases: *spinn_front_end_common.interface.interface_functions.graph_data_specification_writer.GraphDataSpecificationWriter*

Executes data specification generation for sPyNNaker

Module contents

spynnaker.pyNN.protocols package

Submodules

spynnaker.pyNN.protocols.munich_io_ethernet_protocol module

class spynnaker.pyNN.protocols.munich_io_ethernet_protocol.**MunichIoEthernetProtocol**
Bases: object

static disable_motor()

static disable_retina()

static enable_motor()

static enable_retina()

static laser_active_time(*active_time*)

static laser_frequency(*frequency*)

static laser_total_period(*total_period*)

static led_back_active_time(*active_time*)

static led_frequency (*frequency*)

static led_front_active_time (*active_time*)

static led_total_period (*total_period*)

static motor_0_leaky_velocity (*velocity*)

static motor_0_permanent_velocity (*velocity*)

static motor_1_leaky_velocity (*velocity*)

static motor_1_permanent_velocity (*velocity*)

static set_retina_transmission (*event_format*)

static speaker_active_time (*active_time*)

static speaker_frequency (*frequency*)

static speaker_total_period (*total_period*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol module

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**GET_RETINA_KEY_VALUE** (*payload*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**GET_RETINA_PAYLOAD_VALUE** (*payload*)

class spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**MunichIoSpiNNakerLinkProtoco**

Bases: object

Provides Multicast commands for the Munich SpiNNaker-Link protocol

Parameters

- **mode** – The mode of operation of the protocol
- **instance_key** – The optional instance key to use
- **uart_id** – The id of the UART when needed

class MODES

Bases: enum.Enum

BALL_BALANCER = 3

FREE = 5

MY_ORO_BOTICS = 4

PUSH_BOT = 1

RESET_TO_DEFAULT = 0

SPOMNIBOT = 2

add_payload_logic_to_current_output (*payload*, *time=None*)

add_payload_logic_to_current_output_key

bias_values (*bias_id*, *bias_value*, *time=None*)

bias_values_key

configure_master_key (*new_key*, *time=None*)

configure_master_key_key

disable_retina(*time=None*)

disable_retina_key

enable_disable_motor_key

generic_motor0_raw_output_leak_to_0(*pwm_signal*, *time=None*)

generic_motor0_raw_output_leak_to_0_key

generic_motor0_raw_output_permanent(*pwm_signal*, *time=None*)

generic_motor0_raw_output_permanent_key

generic_motor1_raw_output_leak_to_0(*pwm_signal*, *time=None*)

generic_motor1_raw_output_leak_to_0_key

generic_motor1_raw_output_permanent(*pwm_signal*, *time=None*)

generic_motor1_raw_output_permanent_key

generic_motor_disable(*time=None*)

generic_motor_enable(*time=None*)

generic_motor_total_period(*time_in_ms*, *uart_id=0*, *time=None*)

generic_motor_total_period_key

instance_key

The key of this instance of the protocol

master_slave_key

master_slave_set_master_clock_active(*time=None*)

master_slave_set_master_clock_not_started(*time=None*)

master_slave_set_slave(*time=None*)

master_slave_use_internal_counter(*time=None*)

mode

poll_individual_sensor_continuously(*sensor_id*, *time_in_ms*, *time=None*)

poll_individual_sensor_continuously_key

poll_sensors_once(*sensor_id*, *time=None*)

poll_sensors_once_key

protocol_instance = **0**

push_bot_laser_config_active_time(*active_time*, *time=None*)

push_bot_laser_config_active_time_key

push_bot_laser_config_total_period(*total_period*, *time=None*)

push_bot_laser_config_total_period_key

push_bot_laser_set_frequency(*frequency*, *time=None*)

push_bot_laser_set_frequency_key

push_bot_led_back_active_time(*active_time*, *time=None*)

push_bot_led_back_active_time_key

push_bot_led_front_active_time(*active_time*, *time=None*)

push_bot_led_front_active_time_key

push_bot_led_set_frequency(*frequency*, *time=None*)

push_bot_led_set_frequency_key

push_bot_led_total_period(*total_period*, *time=None*)

push_bot_led_total_period_key

push_bot_motor_0_leaking_towards_zero(*velocity*, *time=None*)

push_bot_motor_0_leaking_towards_zero_key

push_bot_motor_0_permanent(*velocity*, *time=None*)

push_bot_motor_0_permanent_key

push_bot_motor_1_leaking_towards_zero(*velocity*, *time=None*)

push_bot_motor_1_leaking_towards_zero_key

push_bot_motor_1_permanent(*velocity*, *time=None*)

push_bot_motor_1_permanent_key

push_bot_speaker_config_active_time(*active_time*, *time=None*)

push_bot_speaker_config_active_time_key

push_bot_speaker_config_total_period(*total_period*, *time=None*)

push_bot_speaker_config_total_period_key

push_bot_speaker_set_melody(*melody*, *time=None*)

push_bot_speaker_set_melody_key

push_bot_speaker_set_tone(*frequency*, *time=None*)

push_bot_speaker_set_tone_key

pwm_pin_output_timer_a_channel_0_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_a_channel_0_ratio_key

pwm_pin_output_timer_a_channel_1_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_a_channel_1_ratio_key

pwm_pin_output_timer_a_duration(*timer_period*, *time=None*)

pwm_pin_output_timer_a_duration_key

pwm_pin_output_timer_b_channel_0_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_b_channel_0_ratio_key()

pwm_pin_output_timer_b_channel_1_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_b_channel_1_ratio_key

pwm_pin_output_timer_b_duration(*timer_period*, *time=None*)

pwm_pin_output_timer_b_duration_key

pwm_pin_output_timer_c_channel_0_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_c_channel_0_ratio_key

pwm_pin_output_timer_c_channel_1_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_c_channel_1_ratio_key()

pwm_pin_output_timer_c_duration(*timer_period*, *time=None*)

pwm_pin_output_timer_c_duration_key

query_state_of_io_lines(*time=None*)

query_state_of_io_lines_key

remove_payload_logic_to_current_output(*payload*, *time=None*)

remove_payload_logic_to_current_output_key

reset_retina(*time=None*)

reset_retina_key

sensor_transmission_key(*sensor_id*)

static **sent_mode_command**()

True if the mode command has ever been requested by any instance

set_mode(*time=None*)

set_mode_key

set_output_pattern_for_payload(*payload*, *time=None*)

set_output_pattern_for_payload_key

set_payload_pins_to_high_impedance(*payload*, *time=None*)

set_payload_pins_to_high_impedance_key

set_retina_key(*new_key*, *time=None*)

set_retina_key_key

set_retina_transmission(*retina_key=<RetinaKey.NATIVE_128_X_128: 67108864>*, *retina_payload=None*, *time=None*)

set the retina transmission key

Parameters

- **retina_key** – the new key for the retina
- **retina_payload** (*enum or None*) – the new payload for the set retina key command packet
- **time** – when to transmit this packet

Returns the command to send

Return type *MultiCastCommand*

set_retina_transmission_key

turn_off_sensor_reporting(*sensor_id*, *time=None*)

turn_off_sensor_reporting_key

uart_id

class spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**RetinaKey**(*value*, *pixels*, *bits_per_coordinate*)

Bases: enum.Enum

DOWNSAMPLE_16_X_16 = 268435456

DOWNSAMPLE_32_X_32 = 201326592

DOWNSAMPLE_64_X_64 = 134217728

FIXED_KEY = 0

NATIVE_128_X_128 = 67108864

bits_per_coordinate

n_neurons

pixels

class spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**RetinaPayload**(*value*, *n_payload_bytes*)

Bases: enum.Enum

ABSOLUTE_2_BYTE_TIMESTAMPS = 1073741824

ABSOLUTE_3_BYTE_TIMESTAMPS = 1610612736

ABSOLUTE_4_BYTE_TIMESTAMPS = 2147483648

DELTA_TIMESTAMPS = 536870912

EVENTS_IN_PAYLOAD = 0

NO_PAYLOAD = 0

n_payload_bytes

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**get_munich_d**(*key*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**get_munich_f**(*key*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**get_munich_i**(*key*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**get_push_bot_laser_led_speaker_**

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**get_push_bot_motor_i**(*key*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**get_retina_i**(*key*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**munich_key**(*I*, *F*, *D*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**munich_key_i**(*I*)

spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.**munich_key_i_d**(*I*, *D*)

Module contents

class spynnaker.pyNN.protocols.**MunichIoEthernetProtocol**

Bases: object

static disable_motor()

static **disable_retina**()

static **enable_motor**()

static **enable_retina**()

static **laser_active_time**(*active_time*)

static **laser_frequency**(*frequency*)

static **laser_total_period**(*total_period*)

static **led_back_active_time**(*active_time*)

static **led_frequency**(*frequency*)

static **led_front_active_time**(*active_time*)

static **led_total_period**(*total_period*)

static **motor_0_leaky_velocity**(*velocity*)

static **motor_0_permanent_velocity**(*velocity*)

static **motor_1_leaky_velocity**(*velocity*)

static **motor_1_permanent_velocity**(*velocity*)

static **set_retina_transmission**(*event_format*)

static **speaker_active_time**(*active_time*)

static **speaker_frequency**(*frequency*)

static **speaker_total_period**(*total_period*)

class spynnaker.pyNN.protocols.**MunichIoSpiNNakerLinkProtocol**(*mode*, *instance_key=None*, *uart_id=0*)

Bases: object

Provides Multicast commands for the Munich SpiNNaker-Link protocol

Parameters

- **mode** – The mode of operation of the protocol
- **instance_key** – The optional instance key to use
- **uart_id** – The id of the UART when needed

class MODES

Bases: enum.Enum

BALL_BALANCER = 3

FREE = 5

MY_ORO_BOTICS = 4

PUSH_BOT = 1

RESET_TO_DEFAULT = 0

SPOMNIBOT = 2

add_payload_logic_to_current_output(*payload*, *time=None*)

add_payload_logic_to_current_output_key

bias_values(*bias_id*, *bias_value*, *time=None*)

bias_values_key

configure_master_key(*new_key*, *time=None*)

configure_master_key_key

disable_retina(*time=None*)

disable_retina_key

enable_disable_motor_key

generic_motor0_raw_output_leak_to_0(*pwm_signal*, *time=None*)

generic_motor0_raw_output_leak_to_0_key

generic_motor0_raw_output_permanent(*pwm_signal*, *time=None*)

generic_motor0_raw_output_permanent_key

generic_motor1_raw_output_leak_to_0(*pwm_signal*, *time=None*)

generic_motor1_raw_output_leak_to_0_key

generic_motor1_raw_output_permanent(*pwm_signal*, *time=None*)

generic_motor1_raw_output_permanent_key

generic_motor_disable(*time=None*)

generic_motor_enable(*time=None*)

generic_motor_total_period(*time_in_ms*, *uart_id=0*, *time=None*)

generic_motor_total_period_key

instance_key

The key of this instance of the protocol

master_slave_key

master_slave_set_master_clock_active(*time=None*)

master_slave_set_master_clock_not_started(*time=None*)

master_slave_set_slave(*time=None*)

master_slave_use_internal_counter(*time=None*)

mode

poll_individual_sensor_continuously(*sensor_id*, *time_in_ms*, *time=None*)

poll_individual_sensor_continuously_key

poll_sensors_once(*sensor_id*, *time=None*)

poll_sensors_once_key

protocol_instance = **0**

push_bot_laser_config_active_time(*active_time*, *time=None*)

push_bot_laser_config_active_time_key

push_bot_laser_config_total_period(*total_period*, *time=None*)

push_bot_laser_config_total_period_key

push_bot_laser_set_frequency(*frequency*, *time=None*)

push_bot_laser_set_frequency_key

push_bot_led_back_active_time(*active_time*, *time=None*)

push_bot_led_back_active_time_key

push_bot_led_front_active_time(*active_time*, *time=None*)

push_bot_led_front_active_time_key

push_bot_led_set_frequency(*frequency*, *time=None*)

push_bot_led_set_frequency_key

push_bot_led_total_period(*total_period*, *time=None*)

push_bot_led_total_period_key

push_bot_motor_0_leaking_towards_zero(*velocity*, *time=None*)

push_bot_motor_0_leaking_towards_zero_key

push_bot_motor_0_permanent(*velocity*, *time=None*)

push_bot_motor_0_permanent_key

push_bot_motor_1_leaking_towards_zero(*velocity*, *time=None*)

push_bot_motor_1_leaking_towards_zero_key

push_bot_motor_1_permanent(*velocity*, *time=None*)

push_bot_motor_1_permanent_key

push_bot_speaker_config_active_time(*active_time*, *time=None*)

push_bot_speaker_config_active_time_key

push_bot_speaker_config_total_period(*total_period*, *time=None*)

push_bot_speaker_config_total_period_key

push_bot_speaker_set_melody(*melody*, *time=None*)

push_bot_speaker_set_melody_key

push_bot_speaker_set_tone(*frequency*, *time=None*)

push_bot_speaker_set_tone_key

pwm_pin_output_timer_a_channel_0_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_a_channel_0_ratio_key

pwm_pin_output_timer_a_channel_1_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_a_channel_1_ratio_key

pwm_pin_output_timer_a_duration(*timer_period*, *time=None*)

pwm_pin_output_timer_a_duration_key

pwm_pin_output_timer_b_channel_0_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_b_channel_0_ratio_key()

pwm_pin_output_timer_b_channel_1_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_b_channel_1_ratio_key

pwm_pin_output_timer_b_duration(*timer_period*, *time=None*)

pwm_pin_output_timer_b_duration_key

pwm_pin_output_timer_c_channel_0_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_c_channel_0_ratio_key

pwm_pin_output_timer_c_channel_1_ratio(*timer_period*, *time=None*)

pwm_pin_output_timer_c_channel_1_ratio_key()

pwm_pin_output_timer_c_duration(*timer_period*, *time=None*)

pwm_pin_output_timer_c_duration_key

query_state_of_io_lines(*time=None*)

query_state_of_io_lines_key

remove_payload_logic_to_current_output(*payload*, *time=None*)

remove_payload_logic_to_current_output_key

reset_retina(*time=None*)

reset_retina_key

sensor_transmission_key(*sensor_id*)

static **sent_mode_command**()

True if the mode command has ever been requested by any instance

set_mode(*time=None*)

set_mode_key

set_output_pattern_for_payload(*payload*, *time=None*)

set_output_pattern_for_payload_key

set_payload_pins_to_high_impedance(*payload*, *time=None*)

set_payload_pins_to_high_impedance_key

set_retina_key(*new_key*, *time=None*)

set_retina_key_key

set_retina_transmission(*retina_key=<RetinaKey.NATIVE_128_X_128: 67108864>*, *retina_payload=None*, *time=None*)

set the retina transmission key

Parameters

- **retina_key** – the new key for the retina
- **retina_payload** (`enum or None`) – the new payload for the set retina key command packet
- **time** – when to transmit this packet

Returns the command to send

Return type *MultiCastCommand*

set_retina_transmission_key

turn_off_sensor_reporting(*sensor_id*, *time=None*)

turn_off_sensor_reporting_key

uart_id

class spynnaker.pyNN.protocols.**RetinaKey**(*value*, *pixels*, *bits_per_coordinate*)
Bases: enum.Enum

DOWNSAMPLE_16_X_16 = 268435456

DOWNSAMPLE_32_X_32 = 201326592

DOWNSAMPLE_64_X_64 = 134217728

FIXED_KEY = 0

NATIVE_128_X_128 = 67108864

bits_per_coordinate

n_neurons

pixels

class spynnaker.pyNN.protocols.**RetinaPayload**(*value*, *n_payload_bytes*)
Bases: enum.Enum

ABSOLUTE_2_BYTE_TIMESTAMPS = 1073741824

ABSOLUTE_3_BYTE_TIMESTAMPS = 1610612736

ABSOLUTE_4_BYTE_TIMESTAMPS = 2147483648

DELTA_TIMESTAMPS = 536870912

EVENTS_IN_PAYLOAD = 0

NO_PAYLOAD = 0

n_payload_bytes

spynnaker.pyNN.utilities package

Subpackages

spynnaker.pyNN.utilities.random_stats package

Submodules

spynnaker.pyNN.utilities.random_stats.abstract_random_stats module

class spynnaker.pyNN.utilities.random_stats.abstract_random_stats.**AbstractRandomStats**
Bases: object

Statistics about PyNN RandomDistribution objects

cdf(*dist*, *v*)
Return the cumulative distribution function value for the value v

high(*dist*)
Return the high cutoff value of the distribution, or None if the distribution is unbounded

low(*dist*)
Return the low cutoff value of the distribution, or None if the distribution is unbounded

mean(*dist*)
Return the mean of the distribution

ppf (*dist*, *p*)
Return the percent point function value for the probability p

std (*dist*)
Return the standard deviation of the distribution

var (*dist*)
Return the variance of the distribution

Module contents

class spynnaker.pyNN.utilities.random_stats.**AbstractRandomStats**
Bases: object

Statistics about PyNN RandomDistribution objects

cdf (*dist*, *v*)
Return the cumulative distribution function value for the value v

high (*dist*)
Return the high cutoff value of the distribution, or None if the distribution is unbounded

low (*dist*)
Return the low cutoff value of the distribution, or None if the distribution is unbounded

mean (*dist*)
Return the mean of the distribution

ppf (*dist*, *p*)
Return the percent point function value for the probability p

std (*dist*)
Return the standard deviation of the distribution

var (*dist*)
Return the variance of the distribution

Submodules

spynnaker.pyNN.utilities.constants module

class spynnaker.pyNN.utilities.constants.**POPULATION_BASED_REGIONS**
Bases: enum.Enum

NEURON_PARAMS = 1

POPULATION_TABLE = 3

PROFILING = 8

PROVENANCE_DATA = 7

RECORDING = 6

SYNAPSE_DYNAMICS = 5

SYNAPSE_PARAMS = 2

SYNAPTIC_MATRIX = 4

SYSTEM = 0

spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider module

class spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.**FakeHBPPortalMachineProvider**(

Bases: object

create()

destroy()

get_machine_info()

wait_till_not_ready()

wait_until_ready()

spynnaker.pyNN.utilities.running_stats module

class spynnaker.pyNN.utilities.running_stats.**RunningStats**
Bases: object

Keeps running statistics From: http://www.johndcook.com/blog/skewness_kurtosis/

add_item(*x*)

add_items(*mean*, *variance*, *n_items*)

mean

n_items

standard_deviation

variance

spynnaker.pyNN.utilities.spynnaker_connection_holder_generations module

class spynnaker.pyNN.utilities.spynnaker_connection_holder_generations.**SpYNNakerConnectionHol**
Bases: object

method that sets up connection holders for reports to use.

spynnaker.pyNN.utilities.spynnaker_failed_state module

class spynnaker.pyNN.utilities.spynnaker_failed_state.**SpynnakerFailedState**
Bases: *spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface*, *spinn_front_end_common.utilities.failed_state.FailedState*, object

get_current_time()

get_distribution_to_stats()

get_pynn_NumpyRNG()

has_reset_last

is_a_pynn_random(*thing*)

max_delay

min_delay

static reset (*annotations=None*)

set_number_of_neurons_per_core (*neuron_type*, *max_permitted*)

spynnaker.pyNN.utilities.spynnaker_neuron_network_specification_report module

class spynnaker.pyNN.utilities.spynnaker_neuron_network_specification_report.**SpYNNakerNeuronG**
Bases: object

spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report module

class spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report.**SpYNNakerSynapticMatrixReport**
Bases: object

generates the synaptic matrix for reporting purposes

spynnaker.pyNN.utilities.utility_calls module

utility class containing simple helper methods

spynnaker.pyNN.utilities.utility_calls.**check_directory_exists_and_create_if_not** (*filename*)
Create a parent directory for a file if it doesn't exist

Parameters **filename** – The file whose parent directory is to be created

spynnaker.pyNN.utilities.utility_calls.**convert_param_to_numpy** (*param*, *no_atoms*)
Convert parameters into numpy arrays

Parameters

- **param** – the param to convert
- **no_atoms** – the number of atoms available for conversion of param

Return numpy.array the converted param in whatever format it was given

spynnaker.pyNN.utilities.utility_calls.**get_maximum_probable_value** (*dist*, *n_items*, *chance=0.01*)
Get the likely maximum value of a RandomDistribution given a number of draws

spynnaker.pyNN.utilities.utility_calls.**get_mean** (*dist*)
Get the mean of a RandomDistribution

spynnaker.pyNN.utilities.utility_calls.**get_minimum_probable_value** (*dist*, *n_items*, *chance=0.01*)
Get the likely minimum value of a RandomDistribution given a number of draws

spynnaker.pyNN.utilities.utility_calls.**get_parameters_size_in_bytes** (*parameters*)
Get the total size of a list of parameters in bytes

Parameters **parameters** – the parameters to compute the total size of

Returns size of all the parameters in bytes

Return type int

spynnaker.pyNN.utilities.utility_calls.**get_probability_within_range**(*dist*, *lower*, *upper*)

Get the probability that a value will fall within the given range for a given RandomDistribution

spynnaker.pyNN.utilities.utility_calls.**get_probable_maximum_selected**(*n_total_selections*, *n_selected*, *selection_prob*, *chance=0.01*)

Get the likely maximum number of items that will be selected from a set of n_selected from a total set of n_total_selections with a probability of selection of selection_prob

spynnaker.pyNN.utilities.utility_calls.**get_standard_deviation**(*dist*)

Get the standard deviation of a RandomDistribution

spynnaker.pyNN.utilities.utility_calls.**get_variance**(*dist*)

Get the variance of a RandomDistribution

spynnaker.pyNN.utilities.utility_calls.**high**(*dist*)

Gets the high or max boundry value for this distribution

Could return None

spynnaker.pyNN.utilities.utility_calls.**low**(*dist*)

Gets the high or min boundry value for this distribution

Could return None

spynnaker.pyNN.utilities.utility_calls.**read_in_data_from_file**(*file_path*, *min_atom*, *max_atom*, *min_time*, *max_time*)

Read in a file of data values where the values are in a format of: <time> <atom id> <data value>

Parameters

- **file_path** – absolute path to a file containing the data
- **min_atom** – min neuron id to which neurons to read in
- **max_atom** – max neuron id to which neurons to read in
- **min_time** – min time slot to read neurons values of.
- **max_time** – max time slot to read neurons values of.

Returns a numpy array of (time stamp, atom id, data value)

spynnaker.pyNN.utilities.utility_calls.**read_spikes_from_file**(*file_path*, *min_atom=0*, *max_atom=inf*, *min_time=0*, *max_time=inf*, *split_value='\t'*)

Read spikes from a file formatted as: <time> <neuron id> :param file_path: absolute path to a file containing spike values :type file_path: str :param min_atom: min neuron id to which neurons to read in :type min_atom: int :param max_atom: max neuron id to which neurons to read in :type max_atom: int :param min_time: min time slot to read neurons values of. :type min_time: int :param max_time: max time slot to read neurons values

of. :type max_time: int :param split_value: the pattern to split by :type split_value: str :return: a numpy array with max_atom elements each of which is a list of spike times. :rtype: numpy array of (int, int)

`spynnaker.pyNN.utilities.utility_calls.`**`set_slice_values`**(*arrays*, *values*, *vertex_slice*)

Set a vertex slice of atoms in a set of arrays to the given values

Parameters

- **`array`** – The array of arrays to set the values in
- **`value`** – The array of arrays of values to set
- **`vertex_slice`** – The slice of parameters to set

`spynnaker.pyNN.utilities.utility_calls.`**`translate_parameters`**(*types*, *byte_array*, *offset*, *vertex_slice*)

Translate an array of data into a set of parameters

Parameters

- **`types`** – the DataType of each of the parameters to translate
- **`byte_array`** – the byte array to read parameters out of
- **`offset`** – where in the byte array to start reading from
- **`vertex_slice`** – the map of atoms from a application vertex

Returns An array of arrays of parameter values, and the new offset

`spynnaker.pyNN.utilities.utility_calls.`**`validate_mars_kiss_64_seed`**(*seed*)

Update the seed to make it compatible with the rng algorithm

`spynnaker.pyNN.utilities.utility_calls.`**`write_parameters_per_neuron`**(*spec*, *vertex_slice*, *parameters*)

Module contents

Submodules

spynnaker.pyNN.abstract_spinnaker_common module

class spynnaker.pyNN.abstract_spinnaker_common.**AbstractSpiNNakerCommon**(*graph_label*, *database_socket_addresses*, *n_chips_required*, *timestep*, *max_delay*, *min_delay*, *host-name*, *user_extra_algorithm_xml_path*, *user_extra_mapping_inputs=No*, *user_extra_algorithms_pre_run=*, *time_scale_factor=None*, *ex-tra_post_run_algorithms=None*, *ex-tra_mapping_algorithms=None*, *ex-tra_load_algorithms=None*, *front_end_versions=None*)

Bases: *spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase*, *spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface*

main interface for neural code

CONFIG_FILE_NAME = 'spynnaker.cfg'

add_application_vertex(*vertex_to_add*)

add_population(*population*)
Called by each population to add itself to the list

add_projection(*projection*)
Called by each projection to add itself to the list

id_counter
property for id_counter, currently used by the populations. (maybe it could live in the pop class???)

Returns

max_delay
The maximum supported delay based in milliseconds

min_delay
The minimum supported delay based in milliseconds

static register_binary_search_path(*search_path*)
Registers an additional binary search path for executables :param search_path: absolute search path for binaries

reset_number_of_neurons_per_core()

run (*run_time*)
Run the model created

Parameters **run_time** – the time in ms to run the simulation for

set_number_of_neurons_per_core (*neuron_type*, *max_permitted*)

stop (*turn_off_machine=None*, *clear_routing_tables=None*, *clear_tags=None*)

Parameters

- **turn_off_machine** (*bool*) – decides if the machine should be powered down after running the execution. Note that this powers down all boards connected to the BMP connections given to the transceiver
- **clear_routing_tables** (*bool*) – informs the tool chain if it should turn off the clearing of the routing tables
- **clear_tags** (*boolean*) – informs the tool chain if it should clear the tags off the machine at stop

Return type None

time_scale_factor
the multiplicative scaling from application time to real execution time

Returns the time scale factor

spynnaker.pyNN.exceptions module

exception spynnaker.pyNN.exceptions.**DelayExtensionException**
Bases: *spinn_front_end_common.utilities.exceptions.ConfigurationException*

Raised when a delay extension vertex fails

exception spynnaker.pyNN.exceptions.**FilterableException**
Bases: *spynnaker.pyNN.exceptions.SpynnakerException*

Raised when it is not possible to determine if an edge should be filtered

exception spynnaker.pyNN.exceptions.**InvalidParameterType**
Bases: *spynnaker.pyNN.exceptions.SpynnakerException*

Raised when a parameter is not recognised

exception spynnaker.pyNN.exceptions.**MemReadException**
Bases: *spynnaker.pyNN.exceptions.SpynnakerException*

Raised when the pynn front end fails to read a certain memory region

exception spynnaker.pyNN.exceptions.**SpynnakerException**
Bases: exceptions.Exception

Superclass of all exceptions from the pynn module

exception spynnaker.pyNN.exceptions.**SynapseRowTooBigException** (*max_size*, *message*)
Bases: *spynnaker.pyNN.exceptions.SpynnakerException*

Raised when a synapse row is bigger than is allowed

max_size
The maximum size allowed

exception spynnaker.pyNN.exceptions.**SynapticBlockGenerationException**
Bases: *spinn_front_end_common.utilities.exceptions.ConfigurationException*

Raised when the synaptic manager fails to generate a synaptic block

exception spynnaker.pyNN.exceptions.**SynapticBlockReadException**
Bases: *spinn_front_end_common.utilities.exceptions.ConfigurationException*

Raised when the synaptic manager fails to read a synaptic block or convert it into readable values

exception spynnaker.pyNN.exceptions.**SynapticConfigurationException**
Bases: *spinn_front_end_common.utilities.exceptions.ConfigurationException*

Raised when the synaptic manager fails for some reason

exception spynnaker.pyNN.exceptions.**SynapticMaxIncomingAtomsSupportException**
Bases: *spinn_front_end_common.utilities.exceptions.ConfigurationException*

Raised when a synaptic sublist exceeds the max atoms possible to be supported

spynnaker.pyNN.spynnaker_external_device_plugin_manager module

class spynnaker.pyNN.spynnaker_external_device_plugin_manager.**SpynnakerExternalDevicePluginMa**
Bases: object

main entrance for the external device plugin manager

static activate_live_output_for(*population*, *database_notify_host=None*, *database_notify_port_num=None*, *database_ack_port_num=None*, *board_address=None*, *port=None*, *host=None*, *tag=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOType.KEY_32_BIT: 2>*, *right_shift=0*, *payload_as_time_stamps=True*, *notify=True*, *use_payload_prefix=True*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_per_time_step=0*)

Output the spikes from a given population from SpiNNaker as they occur in the simulation

Parameters

- **population** (*spynnaker.pyNN.models.pynn_population.Population*) – The population to activate the live output for
- **database_notify_host** (*str*) – the hostname for the device which is listening to the database notification.
- **database_ack_port_num** (*int*) – the port number to which a external device will acknowledge that they have finished reading the database and are ready for it to start execution
- **database_notify_port_num** (*int*) – The port number to which a external device will receive the database is ready command
- **board_address** (*str*) – A fixed board address required for the tag, or None if any address is OK
- **key_prefix** (*int or None*) – the prefix to be applied to the key
- **prefix_type** – if the prefix type is 32 bit or 16 bit

- **message_type** – if the message is a eieio_command message, or eieio data message with 16 bit or 32 bit keys.
- **payload_as_time_stamps** –
- **right_shift** –
- **use_payload_prefix** –
- **notify** –
- **payload_prefix** –
- **payload_right_shift** –
- **number_of_packets_sent_per_time_step** –
- **port** (*int*) – The UDP port to which the live spikes will be sent. If not specified, the port will be taken from the "live_spike_port" parameter in the "Recording" section of the spynnaker cfg file.
- **host** (*str*) – The host name or IP address to which the live spikes will be sent. If not specified, the host will be taken from the "live_spike_host" parameter in the "Recording" section of the spynnaker cfg file.
- **tag** (*int*) – The IP tag to be used for the spikes. If not specified, one will be automatically assigned
- **strip_sdp** (*bool*) – Determines if the SDP headers will be stripped from the transmitted packet.
- **use_prefix** (*bool*) – Determines if the spike packet will contain a common prefix for the spikes

static activate_live_output_to(*population*, *device*)

Activate the output of spikes from a population to an external device. Note that all spikes will be sent to the device.

Parameters

- **population** – The pyNN population object from which spikes will be sent.
- **device** – The pyNN population external device to which the spikes will be sent.

static add_application_vertex(*vertex*)

static add_database_socket_address(*database_notify_host*, *database_notify_port_num*, *database_ack_port_num*)

static add_edge(*vertex*, *device_vertex*, *partition_id*)

adds a edge between two vertices (often a vertex and a external device) on a given partition

Parameters

- **vertex** – the pre vertex to connect the edge from
- **device_vertex** – the post vertex to connect the edge to
- **partition_id** – the partition identifier for making nets

Return type None

static add_socket_address(*socket_address*)

Add a socket address to the list to be checked by the notification protocol

Parameters **socket_address** – the socket address

Return type None:

static machine_time_step()

static time_scale_factor()

static update_live_packet_gather_tracker(*vertex_to_record_from*, *port*, *hostname*, *tag=None*, *board_address=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOType.KEY_32_BIT: 2>*, *right_shift=0*, *payload_as_time_stamps=True*, *use_payload_prefix=True*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_per_time_step=0*, *partition_id=None*)

adds a edge from a vertex to the LPG object, builds as needed and has all the parameters for the creation of the LPG if needed

spynnaker.pyNN.spynnaker_simulator_interface module

class spynnaker.pyNN.spynnaker_simulator_interface.**SpynnakerSimulatorInterface**

Bases: *spinn_front_end_common.utilities.simulator_interface.SimulatorInterface*

get_current_time()

get_distribution_to_stats()

get_pynn_NumpyRNG()

has_reset_last

is_a_pynn_random(*thing*)

max_delay

min_delay

reset(*annotations=None*)

set_number_of_neurons_per_core(*neuron_type*, *max_permitted*)

Module contents

Submodules

spynnaker.gsyn_tools module

spynnaker.gsyn_tools.**check_gsyn**(*gsyn1*, *gsyn2*)

spynnaker.gsyn_tools.**check_path_gysn**(*path*, *n_neurons*, *runtime*, *gsyn*)

spynnaker.gsyn_tools.**check_sister_gysn**(*sister*, *n_neurons*, *runtime*, *gsyn*)

spynnaker.plot_utils module

`spynnaker.plot_utils.`**`get_colour`**`()`

`spynnaker.plot_utils.`**`grid`**`(`*length*`)`

`spynnaker.plot_utils.`**`heat_plot`**`(`*data_sets*, *ylabel=None*, *title=None*`)`

`spynnaker.plot_utils.`**`line_plot`**`(`*data_sets*, *title=None*`)`

`spynnaker.plot_utils.`**`plot_spikes`**`(`*spikes*, *title='spikes'*`)`

Parameters **`spikes`** – Numpy array of spikes

spynnaker.spike_checker module

`spynnaker.spike_checker.`**`synfire_multiple_lines_spike_checker`**`(`*spikes*, *nNeurons*, *lines*, *wrap_around=True*`)`

Checks that there are the expected number of spike lines

Parameters

- **`spikes`** – The spikes
- **`nNeurons`** – Number of neurons
- **`lines`** – Expected number of lines
- **`wrap_around`** – If True the lines will wrap around when reaching the last neuron

`spynnaker.spike_checker.`**`synfire_spike_checker`**`(`*spikes*, *nNeurons*`)`

Module contents

10.1.2 Indices and tables

- genindex
- modindex
- search

sPyNNaker_github

sPyNNaker_individual_docs

CHAPTER 11

sPyNNaker7

This package provides a PyNN implementation for SpiNNaker.

sPyNNaker7_github

sPyNNaker7_individual_docs

CHAPTER 12

spalloc_server

A SpiNNaker machine management application which manages the partitioning and allocation of large SpiNNaker machines into smaller fragments for many simultaneous users.

The spalloc_server module uses a different documentation style so is not included here.

Their documenation can be found at: spalloc_server_readthedocs

spalloc_server_github

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

data_specification, 413
data_specification.constants, 420
data_specification.data_specification_executor, 420
data_specification.data_specification_executor_functions, 421
data_specification.data_specification_generator, 424
data_specification.enums, 417
data_specification.enums.arithemetic_operation, 414
data_specification.enums.commands, 414
data_specification.enums.condition, 416
data_specification.enums.data_type, 416
data_specification.enums.logic_operation, 417
data_specification.enums.random_number_generator, 417
data_specification.exceptions, 450
data_specification.memory_region, 455
data_specification.memory_region_collection, 456
data_specification.utility_calls, 456

p

pacman, 77
pacman.exceptions, 211
pacman.executor, 97
pacman.executor.algorithm_classes, 84
pacman.executor.algorithm_classes.abstract_algorithm, 80
pacman.executor.algorithm_classes.abstract_python_algorithm, 81
pacman.executor.algorithm_classes.external_algorithm, 82
pacman.executor.algorithm_classes.python_class_algorithm, 82
pacman.executor.algorithm_classes.python_function_algorithm, 83
pacman.executor.algorithm_decorators, 91
pacman.executor.algorithm_decorators.abstract_input 87
pacman.executor.algorithm_decorators.algorithm_deco 87
pacman.executor.algorithm_decorators.all_of_input, 89
pacman.executor.algorithm_decorators.one_of_input, 89
pacman.executor.algorithm_decorators.output, 90
pacman.executor.algorithm_decorators.single_input, 90
pacman.executor.algorithm_metadata_xml_reader, 94
pacman.executor.injection_decorator, 94
pacman.executor.pacman_algorithm_executor, 95
pacman.model, 180
pacman.model.abstract_classes, 98
pacman.model.abstract_classes.abstract_has_global_r 98
pacman.model.constraints, 107
pacman.model.constraints.abstract_constraint, 107
pacman.model.constraints.key_allocator_constraints, 100
pacman.model.constraints.key_allocator_constraints 99
pacman.model.constraints.key_allocator_constraints 99
pacman.model.constraints.key_allocator_constraints 99
pacman.model.constraints.key_allocator_constraints 100
pacman.model.constraints.key_allocator_constraints 100
pacman.model.constraints.key_allocator_constraints

100
pacman.model.constraints.partitioner_constraints, 103
pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint, 102
pacman.model.constraints.partitioner_constraints.max_vertex_atoms_constraint, 102
pacman.model.constraints.partitioner_constraints.same_graphs_as_vertex_constraint, 102
pacman.model.constraints.placer_constraints, 105
pacman.model.constraints.placer_constraints.abstract_placer_constraint, 103
pacman.model.constraints.placer_constraints.board_constraint, 104
pacman.model.constraints.placer_constraints.chip_and_core_constraint, 104
pacman.model.constraints.placer_constraints.radial_placement_from_chip_constraint, 105
pacman.model.constraints.placer_constraints.same_chip_as_constraint, 105
pacman.model.constraints.router_constraints, 107
pacman.model.constraints.router_constraints.abstract_router_constraint, 107
pacman.model.decorators, 108
pacman.model.decorators.overrides, 108
pacman.model.graphs, 142
pacman.model.graphs.abstract_edge, 137
pacman.model.graphs.abstract_fpga_vertex, 137
pacman.model.graphs.abstract_graph, 137
pacman.model.graphs.abstract_outgoing_edge_partition, 140
pacman.model.graphs.abstract_spinnaker_link_vertex, 141
pacman.model.graphs.abstract_vertex, 141
pacman.model.graphs.abstract_virtual_vertex, 142
pacman.model.graphs.application, 114
pacman.model.graphs.application.application_edge, 109
pacman.model.graphs.application.application_fpga_vertex, 110
pacman.model.graphs.application.application_graph, 111
pacman.model.graphs.application.application_outgoing_edge_partition, 111
pacman.model.graphs.application.application_spinnaker_link_vertex, 112
pacman.model.graphs.application.application_vertex, 113
pacman.model.graphs.common, 121
pacman.model.graphs.common.constrained_object, 118
pacman.model.graphs.common.edge_traffic_type, 119
pacman.model.graphs.common.graph_partitioner_constraint, 119
pacman.model.graphs.common.constraint, 120
pacman.model.graphs.impl, 126
pacman.model.graphs.impl.graph_constraint, 123
pacman.model.graphs.impl.outgoing_edge_partition, 125
pacman.model.graphs.machine, 133
pacman.model.graphs.machine.machine_edge, 130
pacman.model.graphs.machine.machine_fpga_vertex, 131
pacman.model.graphs.machine.machine_graph, 131
pacman.model.graphs.machine.machine_outgoing_edge_p 132
pacman.model.graphs.machine.machine_spinnaker_link_ 132
pacman.model.graphs.machine.machine_vertex, 133
pacman.model.graphs.machine.simple_machine_vertex, 133
pacman.model.placements, 150
pacman.model.placements.placement, 147
pacman.model.placements.placements, 148
pacman.model.resources, 159
pacman.model.resources.abstract_resource, 152
pacman.model.resources.core_resource, 152
pacman.model.resources.cpu_cycles_per_tick_resource 153
pacman.model.resources.dtcm_resource, 153
pacman.model.resources.element_free_space, 153
pacman.model.resources.iptag_resource, 153
pacman.model.resources.pre_allocated_resource_conta 154
pacman.model.resources.resource_container, 155
pacman.model.resources.reverse_iptag_resource, 156
pacman.model.resources.sdram_resource, 156
pacman.model.resources.specific_board_iptag_resourc 157
pacman.model.resources.specific_board_reverse_iptag 158
pacman.model.resources.specific_chip_sdram_resource

159
pacman.model.resources.specific_core_resource, 159
pacman.model.routing_info, 167
pacman.model.routing_info.abstract_machine_partition_n_keys_map, 164
pacman.model.routing_info.base_key_and_mask, 164
pacman.model.routing_info.dict_based_machine_partition_n_keys_map, 165
pacman.model.routing_info.partition_routing_info, 166
pacman.model.routing_info.routing_info, 166
pacman.model.routing_table_by_partition, 172
pacman.model.routing_table_by_partition.multicast_routing_table_by_partition, 170
pacman.model.routing_table_by_partition.multicast_routing_table_by_partition_entry, 171
pacman.model.routing_tables, 176
pacman.model.routing_tables.multicast_routing_table, 174
pacman.model.routing_tables.multicast_routing_tables, 175
pacman.model.tags, 179
pacman.model.tags.tags, 178
pacman.operations, 195
pacman.operations.algorithm_reports, 184
pacman.operations.algorithm_reports.network_specification_report, 180
pacman.operations.algorithm_reports.reports, 180
pacman.operations.chip_id_allocator_algorithms, 185
pacman.operations.chip_id_allocator_algorithms.malloc_based_chip_id_allocator, 184
pacman.operations.fixed_route_router, 185
pacman.operations.fixed_route_router.fixed_route_router, 185
pacman.operations.multi_cast_router_check_functionality, 186
pacman.operations.multi_cast_router_check_functionality_validity_checker, 185
pacman.operations.partition_algorithms, 187
pacman.operations.partition_algorithms.basic_partitioner, 186
pacman.operations.partition_algorithms.partitioner_interfaces, 186
pacman.operations.placer_algorithms, 188
pacman.operations.placer_algorithms.basic_placer, 187
pacman.operations.placer_algorithms.connective_base, 187
pacman.operations.placer_algorithms.one_to_one_placer, 187
pacman.operations.placer_algorithms.radial_placer, 187
pacman.operations.rig_algorithms, 189
pacman.operations.rig_algorithms.rig_mc_route, 188
pacman.operations.rig_algorithms.rig_place, 188
pacman.operations.rig_algorithms.rig_place_and_route, 188
pacman.operations.rig_algorithms.rig_route, 188
pacman.operations.rigged_algorithms, 190
pacman.operations.rigged_algorithms.hilbert_placer, 189
pacman.operations.rigged_algorithms.hilbert_state, 189
pacman.operations.rigged_algorithms.isomorph_check, 189
pacman.operations.rigged_algorithms.random_placer, 190
pacman.operations.router_algorithms, 190
pacman.operations.router_algorithms.basic_dijkstra, 190
pacman.operations.router_compressors, 192
pacman.operations.router_compressors.basic_route_merger, 191
pacman.operations.router_compressors.malloc_based_mt, 192
pacman.operations.router_compressors.mundys_router, 191
pacman.operations.router_compressors.mundys_router_mt, 191
pacman.operations.routing_info_allocator_algorithms, 194
pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_info_allocator, 193
pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_info_allocator_mt, 193
pacman.operations.routing_info_allocator_algorithms.validity_checker, 192
pacman.operations.routing_info_allocator_algorithms.zoned_malloc_based_routing_info_allocator, 192
pacman.operations.routing_info_allocator_algorithms.zoned_malloc_based_routing_info_allocator_mt, 193
pacman.operations.routing_info_allocator_algorithms.zoned_malloc_based_routing_info_allocator_algorithms, 192
pacman.operations.routing_info_allocator_algorithms.zoned_malloc_based_routing_info_allocator_algorithms, 193
pacman.operations.routing_info_allocator_algorithms.zoned_malloc_based_routing_info_allocator_algorithms, 193

193
pacman.operations.routing_table_generators,
194
pacman.operations.routing_table_generators.basic,
194
pacman.operations.tag_allocator_algorithms,
194
pacman.operations.tag_allocator_algorithms.basic,
194
pacman.utilities, 210
pacman.utilities.algorithm_utilities,
197
pacman.utilities.algorithm_utilities.element_allocator_algorithm,
195
pacman.utilities.algorithm_utilities.field_based_system_utilities,
195
pacman.utilities.algorithm_utilities.machine_algorithm_utilities,
196
pacman.utilities.algorithm_utilities.partition_algorithm_utilities,
196
pacman.utilities.algorithm_utilities.placer_algorithm_utilities,
196
pacman.utilities.algorithm_utilities.routing_info_allocator_utilities,
196
pacman.utilities.constants, 207
pacman.utilities.file_format_converters,
198
pacman.utilities.file_format_converters.convert_to_file_core_allocations,
197
pacman.utilities.file_format_converters.convert_to_file_machine,
197
pacman.utilities.file_format_converters.convert_to_file_machine_graph,
197
pacman.utilities.file_format_converters.convert_to_file_machine_graph_pure_multicast,
198
pacman.utilities.file_format_converters.convert_to_file_placement,
198
pacman.utilities.file_format_converters.convert_to_memory_multi_cast_routes,
198
pacman.utilities.file_format_converters.convert_to_memory_placements,
198
pacman.utilities.file_format_converters.create_file_constraints,
198
pacman.utilities.file_format_schemas,
198
pacman.utilities.rig_converters, 208
pacman.utilities.utility_calls, 208
pacman.utilities.utility_objs, 203
pacman.utilities.utility_objs.field, 199
pacman.utilities.utility_objs.flexi_field,
199
pacman.utilities.utility_objs.resource_tracker,
199
pacman.utilities.vertex_sorter, 210

S

spinn_front_end_common, 493
spinn_front_end_common.abstract_models,
499
spinn_front_end_common.abstract_models.abstract_cha
495
spinn_front_end_common.abstract_models.abstract_ger
495
spinn_front_end_common.abstract_models.abstract_has
496
spinn_front_end_common.abstract_models.abstract_mac
496
spinn_front_end_common.abstract_models.abstract_pro
496
spinn_front_end_common.abstract_models.abstract_pro
497
spinn_front_end_common.abstract_models.abstract_pro
497
spinn_front_end_common.abstract_models.abstract_pro
497
spinn_front_end_common.abstract_models.abstract_rec
498
spinn_front_end_common.abstract_models.abstract_rev
498
spinn_front_end_common.abstract_models.abstract_ser
498
spinn_front_end_common.abstract_models.abstract_sup
498
spinn_front_end_common.abstract_models.abstract_ver
499
spinn_front_end_common.abstract_models.impl,
494
spinn_front_end_common.abstract_models.impl.machine
493
spinn_front_end_common.abstract_models.impl.needs_r
494
spinn_front_end_common.abstract_models.impl.provide
494
spinn_front_end_common.common_model_binaries,
502
spinn_front_end_common.interface, 550
spinn_front_end_common.interface.abstract_spinnaker
546
spinn_front_end_common.interface.buffer_management,
528
spinn_front_end_common.interface.buffer_management
524
spinn_front_end_common.interface.buffer_management
505
spinn_front_end_common.interface.buffer_management
502
spinn_front_end_common.interface.buffer_management
503
spinn_front_end_common.interface.buffer_management

504 spinn_front_end_common.interface.buffer_management.recording_utilities, 532 spinn_front_end_common.interface.interface_function
525 spinn_front_end_common.interface.buffer_management.storage_objects, 533 spinn_front_end_common.interface.interface_function
516 spinn_front_end_common.interface.buffer_management.storage_objects.buffer_interface_for_data, 533 spinn_front_end_common.interface.interface_function
508 spinn_front_end_common.interface.buffer_management.storage_objects.buffer_interface_function, 533 spinn_front_end_common.interface.interface_function
512 spinn_front_end_common.interface.buffer_management.storage_objects.buffer_interface_function, 533 spinn_front_end_common.interface.interface_function
514 spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state, 533 spinn_front_end_common.interface.interface_function
515 spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state, 533 spinn_front_end_common.interface.interface_function
516 spinn_front_end_common.interface.interface_functions, 533 spinn_front_end_common.interface.interface_function
537 spinn_front_end_common.interface.interface_functions.application_interface, 534 spinn_front_end_common.interface.interface_function
530 spinn_front_end_common.interface.interface_functions.application_interface, 534 spinn_front_end_common.interface.interface_function
530 spinn_front_end_common.interface.interface_functions.buffer_extractor, 534 spinn_front_end_common.interface.interface_function
530 spinn_front_end_common.interface.interface_functions.buffer_manager_creator, 535 spinn_front_end_common.interface.interface_function
530 spinn_front_end_common.interface.interface_functions.dipoint_interface, 535 spinn_front_end_common.interface.interface_function
530 spinn_front_end_common.interface.interface_functions.dipoint_interface, 535 spinn_front_end_common.interface.interface_function
531 spinn_front_end_common.interface.interface_functions.dipoint_interface, 535 spinn_front_end_common.interface.interface_function
531 spinn_front_end_common.interface.interface_functions.dipoint_interface, 535 spinn_front_end_common.interface.interface_function
531 spinn_front_end_common.interface.interface_functions.dipoint_interface, 535 spinn_front_end_common.interface.interface_function
531 spinn_front_end_common.interface.interface_functions.database_interface, 536 spinn_front_end_common.interface.interface_function
531 spinn_front_end_common.interface.interface_functions.dsg_region_interface, 536 spinn_front_end_common.interface.interface_function
531 spinn_front_end_common.interface.interface_functions.edge_tag_keys_mapper, 536 spinn_front_end_common.interface.interface_function
531 spinn_front_end_common.interface.interface_functions.graph_binary_gather, 536 spinn_front_end_common.interface.interface_function
532 spinn_front_end_common.interface.interface_functions.graph_data_specification_writer, 536 spinn_front_end_common.interface.interface_function
532 spinn_front_end_common.interface.interface_functions.graph_measurer, 536 spinn_front_end_common.interface.interface_function
532 spinn_front_end_common.interface.interface_functions.graph_provenance_gather, 537 spinn_front_end_common.interface.interface_function
532 spinn_front_end_common.interface.interface_functions.hdpcollocator, 537 spinn_front_end_common.interface.interface_function
532 spinn_front_end_common.interface.interface_functions.hdpcmaxmachine_generator, 537 spinn_front_end_common.interface.interface_function
532 spinn_front_end_common.interface.interface_functions.hdscomexecutedata_specification, 543 spinn_front_end_common.interface.interface_function
532 spinn_front_end_common.interface.interface_functions.idsetmchipnewface_profiling_graphs, 541 spinn_front_end_common.interface.interface_function

541 spinn_front_end_common.interface.profiling.profile_utils,
542 spinn_front_end_common.interface.provenance,
544 spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data,
543 spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_from_machine,
543 spinn_front_end_common.interface.provenance.pacman_provenance_extractor,
544 spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl,
544 spinn_front_end_common.interface.simulation,
546 spinn_front_end_common.interface.simulation.simulation_utilities,
546 spinn_front_end_common.interface.simulator_state,
550 spinn_front_end_common.mapping_algorithms,
551 spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression,
551 spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression.mundy_on_chip_ro
550 spinn_front_end_common.utilities,
553 spinn_front_end_common.utilities.connections,
551 spinn_front_end_common.utilities.connections.live_event_connection,
577 spinn_front_end_common.utilities.constants,
558 spinn_front_end_common.utilities.database,
555 spinn_front_end_common.utilities.database.database_connection,
555 spinn_front_end_common.utilities.database.database_reader,
557 spinn_front_end_common.utilities.database.database_writer,
578 spinn_front_end_common.utilities.exceptions,
578 spinn_front_end_common.utilities.failed_spins,
579 spinn_front_end_common.utilities.function_list,
579 spinn_front_end_common.utilities.globals_variables,
579 spinn_front_end_common.utilities.helpful_functions,
582 spinn_front_end_common.utilities.math_constants,
563 spinn_front_end_common.utilities.notification_protocol,
562 spinn_front_end_common.utilities.notification_protocol_utils,
563 spinn_front_end_common.utilities.report_functions,
565 spinn_front_end_common.utilities.report_functions.e
564 spinn_front_end_common.utilities.report_functions.i
564 spinn_front_end_common.utilities.report_functions.n
564 spinn_front_end_common.utilities.report_functions.n
565 spinn_front_end_common.utilities.report_functions.n
565 spinn_front_end_common.utilities.report_functions.n
565 spinn_front_end_common.utilities.scp,
566 spinn_front_end_common.utilities.scp.clear_iobuf_pr
565 spinn_front_end_common.utilities.scp.scp_clear_iobu
566 spinn_front_end_common.utilities.scp.scp_update_run
566 spinn_front_end_common.utilities.scp.update_runtime
566 spinn_front_end_common.utilities.simulator_interfac
582 spinn_front_end_common.utilities.utility_objs,
575 spinn_front_end_common.utilities.utility_objs.dpri_
572 spinn_front_end_common.utilities.utility_objs.execu
572 spinn_front_end_common.utilities.utility_objs.execu
572 spinn_front_end_common.utilities.utility_objs.extra
570 spinn_front_end_common.utilities.utility_objs.extra
567 spinn_front_end_common.utilities.utility_objs.extra
568 spinn_front_end_common.utilities.utility_objs.extra
568 spinn_front_end_common.utilities.utility_objs.extra
569 spinn_front_end_common.utilities.utility_objs.extra
570 spinn_front_end_common.utilities.utility_objs.extra
572 spinn_front_end_common.utilities.utility_objs.extra
570

spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.reset_counters_pr
571
spinn_front_end_common.utilities.utility
571
spinn_front_end_common.utilities.utility
571
spinn_front_end_common.utilities.utility
571
spinn_front_end_common.utilities.utility
573
spinn_front_end_common.utilities.utility
574
spinn_front_end_common.utilities.utility
574
spinn_front_end_common.utility_models,
603
spinn_front_end_common.utility_models.chi
584
spinn_front_end_common.utility_models.chi
585
spinn_front_end_common.utility_models.com
587
spinn_front_end_common.utility_models.com
589
spinn_front_end_common.utility_models.ext
590
spinn_front_end_common.utility_models.ext
591
spinn_front_end_common.utility_models.liv
594
spinn_front_end_common.utility_models.liv
596
spinn_front_end_common.utility_models.mul
597
spinn_front_end_common.utility_models.rev
599
spinn_front_end_common.utility_models.rev
601
spinn_machine, 15
spinn_machine.chip, 22
spinn_machine.core_subset, 25
spinn_machine.core_subsets, 26
spinn_machine.exceptions, 27
spinn_machine.fixed_route_entry, 28
spinn_machine.link, 28
spinn_machine.link_data_objects, 17
spinn_machine.link_data_objects.abstract_link_data,
16
spinn_machine.link_data_objects.fpga_link_data, 4
17
spinn_machine.link_data_objects.spinnaker_link_data,
17
spinn_machine.machine, 29
spinn_machine.multicast_routing_entry,
33
spinn_machine.processor, 34
spinn_machine.router, 35
spinn_machine.sdram, 37
spinn_machine.spinnaker_processor, 38
spinn_machine.tags, 20
spinn_machine.tags.abstract_tag, 18
spinn_machine.tags.iptag, 18
spinn_machine.tags.reverse_iptag, 19
spinn_machine.utilities, 22
spinn_machine.utilities.ordered_set, 21
spinn_machine.utilities.progress_bar,
22
spinn_machine.virtual_machine, 39
spinn_storage_handlers, 59
spinn_storage_handlers.abstract_classes,
65
spinn_storage_handlers.abstract_classes.abstract_bu
59
spinn_storage_handlers.abstract_classes.abstract_by
61
spinn_storage_handlers.abstract_classes.abstract_by
62
spinn_storage_handlers.abstract_classes.abstract_co
63
spinn_storage_handlers.abstract_classes.abstract_da
63
spinn_storage_handlers.abstract_classes.abstract_da
64
spinn_storage_handlers.buffered_bytearray_data_stor
69
spinn_storage_handlers.buffered_file_data_storage,
70
spinn_storage_handlers.buffered_tempfile_data_stora
70
spinn_storage_handlers.exceptions, 71
spinn_storage_handlers.file_data_reader,
71
spinn_storage_handlers.file_data_writer,
72
spinn_utilities, 3
spinn_utilities.abstract_base, 6
spinn_utilities.conf_loader, 7
spinn_utilities.configs, 5
spinn_utilities.configs.camel_case_config_parser,
3
spinn_utilities.configs.case_sensitive_parser,
4
spinn_utilities.configs.no_config_found_exception,
5
spinn_utilities.configs.unexpected_config_exception
5
spinn_utilities.executable_finder, 9

spinn_utilities.helpful_functions, 9
spinn_utilities.log, 10
spinn_utilities.logger_utils, 10
spinn_utilities.ordered_set, 10
spinn_utilities.overrides, 11
spinn_utilities.package_loader, 11
spinn_utilities.progress_bar, 12
spinn_utilities.safe_eval, 12
spinn_utilities.socket_address, 13
spinn_utilities.testing, 6
spinn_utilities.testing.log_checker, 5
spinn_utilities.timer, 13
spinnman, 217
spinnman.connections, 245
spinnman.connections.abstract_classes, 225
spinnman.connections.abstract_classes.connection, 219
spinnman.connections.abstract_classes.eieio_receiver, 219
spinnman.connections.abstract_classes.eieio_sender, 220
spinnman.connections.abstract_classes.listenable, 220
spinnman.connections.abstract_classes.multicast_receiver, 220
spinnman.connections.abstract_classes.multicast_sender, 221
spinnman.connections.abstract_classes.scp_receiver, 222
spinnman.connections.abstract_classes.scp_sender, 222
spinnman.connections.abstract_classes.sdp_receiver, 223
spinnman.connections.abstract_classes.sdp_sender, 223
spinnman.connections.abstract_classes.spinnaker_boot_receiver, 224
spinnman.connections.abstract_classes.spinnaker_boot_sender, 224
spinnman.connections.connection_listener, 243
spinnman.connections.scp_request_pipeline, 243
spinnman.connections.socket_address_with_chip, 244
spinnman.connections.token_bucket, 245
spinnman.connections.udp_packet_connections, 237
spinnman.connections.udp_packet_connections.bmp_connection, 230
spinnman.connections.udp_packet_connections.multicast_connection, 231
spinnman.connections.udp_packet_connections.multicast_connection, 232
spinnman.connections.udp_packet_connections.ip_addr 232
spinnman.connections.udp_packet_connections.scamp_c 233
spinnman.connections.udp_packet_connections.sdp_con 233
spinnman.connections.udp_packet_connections.udp_con 234
spinnman.connections.udp_packet_connections.udp_lis 236
spinnman.connections.udp_packet_connections.utils, 236
spinnman.constants, 380
spinnman.exceptions, 381
spinnman.messages, 325
spinnman.messages.eieio, 267
spinnman.messages.eieio.abstract_messages, 248
spinnman.messages.eieio.abstract_messages.eieio_mes 248
spinnman.messages.eieio.command_messages, 253
spinnman.messages.eieio.command_messages.database_c 248
spinnman.messages.eieio.command_messages.eieio_comm 248
spinnman.messages.eieio.command_messages.eieio_comm 249
spinnman.messages.eieio.command_messages.event_stop 250
spinnman.messages.eieio.command_messages.host_data_ 250
spinnman.messages.eieio.command_messages.host_data_ 250
spinnman.messages.eieio.command_messages.host_send_ 251
spinnman.messages.eieio.command_messages.notificati 251
spinnman.messages.eieio.command_messages.notificati 251
spinnman.messages.eieio.command_messages.padding_re 251
spinnman.messages.eieio.command_messages.spinnaker_ 252
spinnman.messages.eieio.command_messages.spinnaker_ 252
spinnman.messages.eieio.command_messages.start_requ 253
spinnman.messages.eieio.command_messages.stop_reque 253
spinnman.messages.eieio.create_eieio_command, 266
spinnman.messages.eieio.create_eieio_data,

266
spinnman.messages.eieio.data_messages, 262
spinnman.messages.eieio.data_messages.abstract_data_element, 257
spinnman.messages.eieio.data_messages.eieio_data_header, 258
spinnman.messages.eieio.data_messages.eieio_data_message, 259
spinnman.messages.eieio.data_messages.key_data_element, 261
spinnman.messages.eieio.data_messages.key_payload_data_element, 261
spinnman.messages.eieio.eieio_prefix, 266
spinnman.messages.eieio.eieio_type, 266
spinnman.messages.multicast_message, 324
spinnman.messages.scp, 308
spinnman.messages.scp.abstract_messages, 271
spinnman.messages.scp.abstract_messages.bmp_request, 268
spinnman.messages.scp.abstract_messages.bmp_response, 269
spinnman.messages.scp.abstract_messages.scp_request, 269
spinnman.messages.scp.abstract_messages.scp_response, 270
spinnman.messages.scp.enums, 277
spinnman.messages.scp.enums.alloc_free, 274
spinnman.messages.scp.enums.bmp_info, 274
spinnman.messages.scp.enums.iptag_command, 274
spinnman.messages.scp.enums.led_action, 275
spinnman.messages.scp.enums.power_command, 275
spinnman.messages.scp.enums.scp_command, 275
spinnman.messages.scp.enums.scp_result, 276
spinnman.messages.scp.enums.signal, 277
spinnman.messages.scp.impl, 296
spinnman.messages.scp.impl.app_stop, 280
spinnman.messages.scp.impl.application_run, 281
spinnman.messages.scp.impl.bmp_get_version, 281
spinnman.messages.scp.impl.bmp_set_led, 281
spinnman.messages.scp.impl.check_ok_response, 282
spinnman.messages.scp.impl.count_state, 282
spinnman.messages.scp.impl.count_state_response, 282
spinnman.messages.scp.impl.fill_request, 283
spinnman.messages.scp.impl.fixed_route_init, 283
spinnman.messages.scp.impl.fixed_route_read, 284
spinnman.messages.scp.impl.flood_fill_data, 284
spinnman.messages.scp.impl.flood_fill_end, 285
spinnman.messages.scp.impl.flood_fill_start, 285
spinnman.messages.scp.impl.get_chip_info, 285
spinnman.messages.scp.impl.get_chip_info_response, 286
spinnman.messages.scp.impl.get_version, 286
spinnman.messages.scp.impl.get_version_response, 287
spinnman.messages.scp.impl.iptag_clear, 287
spinnman.messages.scp.impl.iptag_get, 287
spinnman.messages.scp.impl.iptag_get_info, 288
spinnman.messages.scp.impl.iptag_get_info_response, 288
spinnman.messages.scp.impl.iptag_set, 288
spinnman.messages.scp.impl.iptag_set_tto, 289
spinnman.messages.scp.impl.read_adc, 289
spinnman.messages.scp.impl.read_fpga_register, 289
spinnman.messages.scp.impl.read_link, 290
spinnman.messages.scp.impl.read_memory, 290
spinnman.messages.scp.impl.reverse_iptag_set, 291
spinnman.messages.scp.impl.router_alloc, 291
spinnman.messages.scp.impl.router_clear, 291
spinnman.messages.scp.impl.router_init, 292
spinnman.messages.scp.impl.sdram_alloc, 292
spinnman.messages.scp.impl.sdram_de_alloc,

293
spinnman.messages.scp.impl.send_signal, 293
spinnman.messages.scp.impl.set_led, 294
spinnman.messages.scp.impl.set_power, 294
spinnman.messages.scp.impl.write_fpga_register, 294
spinnman.messages.scp.impl.write_link, 295
spinnman.messages.scp.impl.write_memory, 295
spinnman.messages.scp.scp_request_header, 307
spinnman.messages.scp.scp_response_header, 308
spinnman.messages.sdp, 312
spinnman.messages.sdp.sdp_flag, 309
spinnman.messages.sdp.sdp_header, 310
spinnman.messages.sdp.sdp_message, 312
spinnman.messages.spinnaker_boot, 320
spinnman.messages.spinnaker_boot.spinnaker_boot_messages, 315
spinnman.messages.spinnaker_boot.spinnaker_boot_messages, 316
spinnman.messages.spinnaker_boot.spinnaker_boot_messages, 317
spinnman.messages.spinnaker_boot.system_variable_boot_values, 317
spinnman.model, 347
spinnman.model.adc_info, 331
spinnman.model.bmp_connection_data, 332
spinnman.model.chip_info, 333
spinnman.model.chip_summary_info, 334
spinnman.model.cpu_info, 336
spinnman.model.cpu_infos, 338
spinnman.model.diagnostic_filter, 339
spinnman.model.enums, 328
spinnman.model.enums.cpu_state, 325
spinnman.model.enums.diagnostic_filter_default_routing_status, 325
spinnman.model.enums.diagnostic_filter_destination, 326
spinnman.model.enums.diagnostic_filter_emergency_routing_status, 326
spinnman.model.enums.diagnostic_filter_packet_type, 326
spinnman.model.enums.diagnostic_filter_payload_status, 327
spinnman.model.enums.diagnostic_filter_source, 327
spinnman.model.enums.mailbox_command, 327
spinnman.model.enums.p2p_table_route, 327
spinnman.model.enums.run_time_error, 328
spinnman.model.executable_targets, 340
spinnman.model.heap_element, 341
spinnman.model.io_buffer, 341
spinnman.model.machine_dimensions, 342
spinnman.model.p2p_table, 342
spinnman.model.router_diagnostics, 343
spinnman.model.version_info, 346
spinnman.processes, 368
spinnman.processes.abstract_multi_connection_process, 361
spinnman.processes.abstract_multi_connection_process, 362
spinnman.processes.abstract_process, 362
spinnman.processes.abstract_single_connection_process, 362
spinnman.processes.application_run_process, 362
spinnman.processes.de_alloc_sdram_process, 362
spinnman.processes.fill_process, 363
spinnman.processes.get_cpu_info_process, 363
spinnman.processes.get_heap_process, 363
spinnman.processes.get_machine_process, 363
spinnman.processes.get_routes_process, 364
spinnman.processes.get_tags_process, 364
spinnman.processes.get_version_process, 364
spinnman.processes.load_fixed_route_routing_entry_process, 364
spinnman.processes.load_routes_process, 365
spinnman.processes.malloc_sdram_process, 365
spinnman.processes.most_direct_connection_selector, 365
spinnman.processes.read_fixed_route_routing_entry_process, 365
spinnman.processes.read_iobuf_process, 366
spinnman.processes.read_memory_process, 366
spinnman.processes.read_router_diagnostics_process, 366
spinnman.processes.round_robin_connection_selector, 367
spinnman.processes.send_single_command_process, 367
spinnman.processes.write_memory_flood_process, 367

spinnman.processes.write_memory_process, 367
spinnman.transceiver, 383
spinnman.utilities, 380
spinnman.utilities.appid_tracker, 378
spinnman.utilities.io, 375
spinnman.utilities.io.abstract_io, 372
spinnman.utilities.io.file_io, 373
spinnman.utilities.io.memory_io, 374
spinnman.utilities.locate_connected_machine_ip_address, 379
spinnman.utilities.reports, 379
spinnman.utilities.utility_functions, 379
spynnaker, 618
spynnaker.gsyn_tools, 837
spynnaker.plot_utils, 838
spynnaker.pyNN, 837
spynnaker.pyNN.abstract_spinnaker_common, 833
spynnaker.pyNN.connections, 620
spynnaker.pyNN.connections.ethernet_command_connection, 618
spynnaker.pyNN.connections.ethernet_control_connection, 619
spynnaker.pyNN.connections.spynnaker_live_spikes_connection, 619
spynnaker.pyNN.exceptions, 834
spynnaker.pyNN.external_devices_models, 658
spynnaker.pyNN.external_devices_models.abstract_ethernet_controller, 650
spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor, 650
spynnaker.pyNN.external_devices_models.abstract_ethernet_translator, 651
spynnaker.pyNN.external_devices_models.abstract_multiple_external_devices_model, 651
spynnaker.pyNN.external_devices_models.arbitrary_pyNN_external_devices_models.push_bot.pus 652
spynnaker.pyNN.external_devices_models.external_devices_models.push_bot.pus 652
spynnaker.pyNN.external_devices_models.external_devices_models.push_bot.pus 653
spynnaker.pyNN.external_devices_models.external_devices_models.push_bot.pus 654
spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device, 655
spynnaker.pyNN.external_devices_models.munich_spinnaker_link_relative_device, 657
spynnaker.pyNN.external_devices_models.push_bot, 649
spynnaker.pyNN.external_devices_models.push_bot.abstract_ethernet_control_device, 649
spynnaker.pyNN.external_devices_models.push_bot.pus 649
spynnaker.pyNN.external_devices_models.push_bot.pus 624
spynnaker.pyNN.external_devices_models.push_bot.pus 622
spynnaker.pyNN.external_devices_models.push_bot.pus 623
spynnaker.pyNN.external_devices_models.push_bot.pus 632
spynnaker.pyNN.external_devices_models.push_bot.pus 625
spynnaker.pyNN.external_devices_models.push_bot.pus 626
spynnaker.pyNN.external_devices_models.push_bot.pus 627
spynnaker.pyNN.external_devices_models.push_bot.pus 628
spynnaker.pyNN.external_devices_models.push_bot.pus 629
spynnaker.pyNN.external_devices_models.push_bot.pus 629
spynnaker.pyNN.external_devices_models.push_bot.pus 630
spynnaker.pyNN.external_devices_models.push_bot.pus 631
spynnaker.pyNN.external_devices_models.push_bot.pus 631
spynnaker.pyNN.external_devices_models.push_bot.pus 640
spynnaker.pyNN.external_devices_models.push_bot.pus 639
spynnaker.pyNN.external_devices_models.push_bot.pus 639
spynnaker.pyNN.external_devices_models.push_bot.pus 639
spynnaker.pyNN.external_devices_models.push_bot.pus 639
spynnaker.pyNN.external_devices_models.push_bot.pus 639
spynnaker.pyNN.external_devices_models.push_bot.pus 640
spynnaker.pyNN.external_devices_models.push_bot.pus 640
spynnaker.pyNN.external_devices_models.push_bot.pus 646
spynnaker.pyNN.external_devices_models.push_bot.pus 642
spynnaker.pyNN.external_devices_models.push_bot.pus 643
spynnaker.pyNN.external_devices_models.push_bot.pus 644
spynnaker.pyNN.external_devices_models.push_bot.pus 644
spynnaker.pyNN.external_devices_models.push_bot.pus 645

spynnaker.pyNN.external_devices_models.threshold_type_multicast_device_control, 658
spynnaker.pyNN.model_binaries, 665
spynnaker.pyNN.models, 817
spynnaker.pyNN.models.abstract_models, 667
spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses, 665
spynnaker.pyNN.models.abstract_models.abstract_contains_units, 666
spynnaker.pyNN.models.abstract_models.abstract_filterable_edge, 666
spynnaker.pyNN.models.abstract_models.abstract_population_initializable, 666
spynnaker.pyNN.models.abstract_models.abstract_population_settable, 667
spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set, 667
spynnaker.pyNN.models.abstract_models.abstract_weight_updatable, 667
spynnaker.pyNN.models.common, 673
spynnaker.pyNN.models.common.abstract_neuron_recorder, 669
spynnaker.pyNN.models.common.abstract_spike_recorder, 670
spynnaker.pyNN.models.common.abstract_uint32_recorder, 670
spynnaker.pyNN.models.common.eieio_spike_recorder, 671
spynnaker.pyNN.models.common.gsyn_inhibitory_recorder, 671
spynnaker.pyNN.models.common.multi_spike_recorder, 672
spynnaker.pyNN.models.common.neuron_recorder, 672
spynnaker.pyNN.models.common.recording_utils, 672
spynnaker.pyNN.models.common.simple_population_settable, 673
spynnaker.pyNN.models.common.spike_recorder, 673
spynnaker.pyNN.models.neural_projections, 699
spynnaker.pyNN.models.neural_projections.connectors, 686
spynnaker.pyNN.models.neural_projections.connectors.abstract_connector, 677
spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector, 678
spynnaker.pyNN.models.neural_projections.connectors.abstract_dependent_probability_connector, 679
spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector, 680
spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector, 681
spynnaker.pyNN.models.neural_projections.connectors, 682
spynnaker.pyNN.models.neural_projections.connectors, 683
spynnaker.pyNN.models.neural_projections.connectors, 683
spynnaker.pyNN.models.neural_projections.connectors, 684
spynnaker.pyNN.models.neural_projections.connectors, 685
spynnaker.pyNN.models.neural_projections.connectors, 686
spynnaker.pyNN.models.neural_projections.delay_affe, 696
spynnaker.pyNN.models.neural_projections.delay_affe, 696
spynnaker.pyNN.models.neural_projections.delayed_ap, 696
spynnaker.pyNN.models.neural_projections.delayed_ma, 697
spynnaker.pyNN.models.neural_projections.projection, 697
spynnaker.pyNN.models.neural_projections.projection, 698
spynnaker.pyNN.models.neural_projections.synapse_in, 698
spynnaker.pyNN.models.neural_properties, 701
spynnaker.pyNN.models.neural_properties.neural_para, 701
spynnaker.pyNN.models.neuron, 786
spynnaker.pyNN.models.neuron.abstract_population_ve, 778
spynnaker.pyNN.models.neuron.additional_inputs, 703
spynnaker.pyNN.models.neuron.additional_inputs.abst, 702
spynnaker.pyNN.models.neuron.additional_inputs.addi, 703
spynnaker.pyNN.models.neuron.builds, 718
spynnaker.pyNN.models.neuron.builds.eif_cond_alpha_, 705
spynnaker.pyNN.models.neuron.builds.hh_cond_exp, 706
spynnaker.pyNN.models.neuron.builds.if_cond_alpha, 707
spynnaker.pyNN.models.neuron.builds.if_cond_exp_bas, 708
spynnaker.pyNN.models.neuron.builds.if_cond_exp_sto, 709
spynnaker.pyNN.models.neuron.builds.if_curr_alpha, 710
spynnaker.pyNN.models.neuron.builds.if_curr_delta,

711
spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base, 712
spynnaker.pyNN.models.neuron.builds.if_curr_exp_base, 713
spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive, 714
spynnaker.pyNN.models.neuron.builds.if_facets_hardware1, 715
spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base, 716
spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base, 717
spynnaker.pyNN.models.neuron.connection_holder, 783
spynnaker.pyNN.models.neuron.input_types, 726
spynnaker.pyNN.models.neuron.input_types.abstract_input_type, 724
spynnaker.pyNN.models.neuron.input_types.input_type_conductance, 725
spynnaker.pyNN.models.neuron.input_types.input_type_current, 725
spynnaker.pyNN.models.neuron.master_pop_table_generators, 731
spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory, 727
spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array, 729
spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search, 730
spynnaker.pyNN.models.neuron.neuron_models, 738
spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model, 733
spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh, 735
spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate, 736
spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire, 737
spynnaker.pyNN.models.neuron.plasticity, 755
spynnaker.pyNN.models.neuron.plasticity.stdp, 755
spynnaker.pyNN.models.neuron.plasticity.stdp.common, 743
spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers, 743
spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure, 744
spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure, 743
spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_weight_only, 744
spynnaker.pyNN.models.neuron.plasticity.stdp.synaps 744
spynnaker.pyNN.models.neuron.plasticity.stdp.timing 748
spynnaker.pyNN.models.neuron.plasticity.stdp.timing 745
spynnaker.pyNN.models.neuron.plasticity.stdp.timing 745
spynnaker.pyNN.models.neuron.plasticity.stdp.timing 746
spynnaker.pyNN.models.neuron.plasticity.stdp.timing 747
spynnaker.pyNN.models.neuron.plasticity.stdp.timing 747
spynnaker.pyNN.models.neuron.plasticity.stdp.timing 748
spynnaker.pyNN.models.neuron.plasticity.stdp.weight 753
spynnaker.pyNN.models.neuron.plasticity.stdp.weight 751
spynnaker.pyNN.models.neuron.plasticity.stdp.weight 751
spynnaker.pyNN.models.neuron.plasticity.stdp.weight 752
spynnaker.pyNN.models.neuron.plasticity.stdp.weight 752
spynnaker.pyNN.models.neuron.plasticity.stdp.weight 753
spynnaker.pyNN.models.neuron.plasticity.stdp.weight 784
spynnaker.pyNN.models.neuron.synapse_dynamics, 760
spynnaker.pyNN.models.neuron.synapse_dynamics.abstr 755
spynnaker.pyNN.models.neuron.synapse_dynamics.abstr 756
spynnaker.pyNN.models.neuron.synapse_dynamics.abstr 756
spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_ 758
spynnaker.pyNN.models.neuron.synapse_dynamics.synap 758
spynnaker.pyNN.models.neuron.synapse_dynamics.synap 759
spynnaker.pyNN.models.neuron.synapse_io, 766
spynnaker.pyNN.models.neuron.synapse_io.abstract_sy 765
spynnaker.pyNN.models.neuron.synapse_io.synapse_io_ 765
spynnaker.pyNN.models.neuron.synapse_types, 770
spynnaker.pyNN.models.neuron.synapse_types.abstract

767
spynnaker.pyNN.models.neuron.synapse_types.synaptic_delta,
768
spynnaker.pyNN.models.neuron.synapse_types.synaptic_dual_exponential,
769
spynnaker.pyNN.models.neuron.synapse_types.synaptic_exponential,
769
spynnaker.pyNN.models.neuron.synaptic_manager,
785
spynnaker.pyNN.models.neuron.threshold_types,
775
spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type,
773
spynnaker.pyNN.models.neuron.threshold_types.threshold_type_mass_stochastic,
774
spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static,
775
spynnaker.pyNN.models.pynn_population_common,
814
spynnaker.pyNN.models.pynn_projection_common,
816
spynnaker.pyNN.models.recording_common,
816
spynnaker.pyNN.models.spike_source, 800
spynnaker.pyNN.models.spike_source.spike_source_array,
793
spynnaker.pyNN.models.spike_source.spike_source_from_file,
795
spynnaker.pyNN.models.spike_source.spike_source_poisson,
795
spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex,
799
spynnaker.pyNN.models.utility_models,
810
spynnaker.pyNN.models.utility_models.delay_block,
806
spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex,
806
spynnaker.pyNN.models.utility_models.delay_extension_vertex,
807
spynnaker.pyNN.models.utility_models.spike_injector,
809
spynnaker.pyNN.overridden_pacman_functions,
817
spynnaker.pyNN.overridden_pacman_functions.graph_edge_filter,
817
spynnaker.pyNN.overridden_pacman_functions.graph_edge_weight_updater,
817
spynnaker.pyNN.overridden_pacman_functions.spynnaker_data_specification_writer,
817
spynnaker.pyNN.protocols, 822
spynnaker.pyNN.protocols.munich_io_ethernet_protocol,
817
spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol,
818
spynnaker.pyNN.sdnnaker_external_device_plugin_man
835
spynnaker.pyNN.sdnnaker_simulator_interface,
837
spynnaker.pyNN.utilities, 833
spynnaker.pyNN.utilities.constants, 828
spynnaker.pyNN.utilities.fake_HBP_Portal_machine_pr
829
spynnaker.pyNN.utilities.random_stats,
828
spynnaker.pyNN.utilities.random_stats.abstract_rand
827
spynnaker.pyNN.utilities.running_stats,
829
spynnaker.pyNN.utilities.spynnaker_connection_holde
829
spynnaker.pyNN.utilities.spynnaker_failed_state,
829
spynnaker.pyNN.utilities.spynnaker_neuron_network_s
830
spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_
830
spynnaker.pyNN.utilities.utility_calls,
830
spynnaker.spike_checker, 838

Index

A

a (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh attribute), 735
a (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 739
A3_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependence attribute), 752
A3_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet attribute), 755
A3_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceA attribute), 753
A3_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet attribute), 755
A_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus attribute), 751
A_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractHasAPlusAMinus attribute), 753
A_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus attribute), 751
A_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractHasAPlusAMinus attribute), 753
ABORT (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
ABORT (spinnman.model.enums.RunTimeError attribute), 329
ABSOLUTE_2_BYTE_TIMESTAMPS (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload attribute), 822
ABSOLUTE_2_BYTE_TIMESTAMPS (spynnaker.pyNN.protocols.RetinaPayload attribute), 827
ABSOLUTE_3_BYTE_TIMESTAMPS (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload attribute), 822
ABSOLUTE_3_BYTE_TIMESTAMPS (spynnaker.pyNN.protocols.RetinaPayload attribute), 827
ABSOLUTE_4_BYTE_TIMESTAMPS (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload attribute), 822
ABSOLUTE_4_BYTE_TIMESTAMPS (spynnaker.pyNN.protocols.RetinaPayload attribute), 827
AbstractAcceptsIncomingSynapses (class in spynnaker.pyNN.models.abstract_models), 667
AbstractAcceptsIncomingSynapses (class in spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_ 665
AbstractAdditionalInput (class in spynnaker.pyNN.models.neuron.additional_inputs), 703
AbstractAdditionalInput (class in spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional 702
AbstractAlgorithm (class in pacman.executor.algorithm_classes), 84
AbstractAlgorithm (class in pacman.executor.algorithm_classes.abstract_algorithm), 80
AbstractBase (class in spinn_utilities.abstract_base), 6
AbstractBufferedDataStorage (class in spinn_storage_handlers.abstract_classes), 65
AbstractBufferedDataStorage (class in spinn_storage_handlers.abstract_classes.abstract_buffered_data_s 59
AbstractByteReader (class in spinn_storage_handlers.abstract_classes), 66
AbstractByteReader (class in spinn_storage_handlers.abstract_classes.abstract_byte_reader), 61
AbstractByteWriter (class in spinn_storage_handlers.abstract_classes), 67
AbstractByteWriter (class in spinn_storage_handlers.abstract_classes.abstract_byte_writer), 62
AbstractChangableAfterRun (class in

spinn_front_end_common.abstract_models), 499
AbstractChangableAfterRun (class in spinn_front_end_common.abstract_models.abstract_changable_after_run), 495
AbstractConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 686
AbstractConnector (class in spynnaker.pyNN.models.neural_projections.connectors.abstract_connector), 677
AbstractConstraint (class in pacman.model.constraints), 107
AbstractConstraint (class in pacman.model.constraints.abstract_constraint), 107
AbstractContainsUnits (class in spynnaker.pyNN.models.abstract_models), 669
AbstractContainsUnits (class in spynnaker.pyNN.models.abstract_models.abstract_contains_units), 666
AbstractContextManager (class in spinn_storage_handlers.abstract_classes), 68
AbstractContextManager (class in spinn_storage_handlers.abstract_classes.abstract_context_manager), 63
AbstractDataElement (class in spinnman.messages.eieio.data_messages), 262
AbstractDataElement (class in spinnman.messages.eieio.data_messages.abstract_data_element), 257
AbstractDataReader (class in spinn_storage_handlers.abstract_classes), 68
AbstractDataReader (class in spinn_storage_handlers.abstract_classes.abstract_data_reader), 63
AbstractDataWriter (class in spinn_storage_handlers.abstract_classes), 69
AbstractDataWriter (class in spinn_storage_handlers.abstract_classes.abstract_data_writer), 64
AbstractEdge (class in pacman.model.graphs), 142
AbstractEdge (class in pacman.model.graphs.abstract_edge), 137
AbstractEIEIOMessage (class in spinnman.messages.eieio.abstract_messages), 248
AbstractEIEIOMessage (class in spinnman.messages.eieio.abstract_messages.eieio_message), 248
AbstractEthernetController (class in spynnaker.pyNN.external_devices_models), 658
AbstractEthernetController (class in spynnaker.pyNN.external_devices_models.abstract_ethernet_controller), 650
AbstractEthernetSensor (class in spynnaker.pyNN.external_devices_models), 658
AbstractEthernetSensor (class in spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor), 650
AbstractEthernetTranslator (class in spynnaker.pyNN.external_devices_models), 659
AbstractEthernetTranslator (class in spynnaker.pyNN.external_devices_models.abstract_ethernet_translator), 651
AbstractFilterableEdge (class in spynnaker.pyNN.models.abstract_models), 668
AbstractFilterableEdge (class in spynnaker.pyNN.models.abstract_models.abstract_filterable_edge), 666
AbstractFPGAVertex (class in pacman.model.graphs), 143
AbstractFPGAVertex (class in pacman.model.graphs.abstract_fpga_vertex), 137
AbstractGeneratesDataSpecification (class in spinn_front_end_common.abstract_models), 499
AbstractGeneratesDataSpecification (class in spinn_front_end_common.abstract_models.abstract_generates_data_specification), 495
AbstractGraph (class in pacman.model.graphs), 143
AbstractGraph (class in pacman.model.graphs.abstract_graph), 137
AbstractHasAPlusAMinus (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 753
AbstractHasAPlusAMinus (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus), 751
AbstractHasAssociatedBinary (class in spinn_front_end_common.abstract_models), 499
AbstractHasAssociatedBinary (class in spinn_front_end_common.abstract_models.abstract_has_associated_binary), 496
AbstractHasGlobalMaxAtoms (class in pacman.model.abstract_classes), 98
AbstractHasGlobalMaxAtoms (class in pacman.model.abstract_classes.abstract_has_global_max_atoms), 98
AbstractHasProfileData (class in spinn_front_end_common.interface.profiling.abstract_has_profile_data), 541
AbstractInput (class in pac-

man.executor.algorithm_decorators), 91
AbstractInput (class in pacman.executor.algorithm_decorators.abstract_input), 87
AbstractInputType (class in spynnaker.pyNN.models.neuron.input_types), 726
AbstractInputType (class in spynnaker.pyNN.models.neuron.input_types.abstract_input_type), 724
AbstractIO (class in spinnman.utilities.io), 375
AbstractIO (class in spinnman.utilities.io.abstract_io), 372
AbstractKeyAllocatorConstraint (class in pacman.model.constraints.key_allocator_constraints), 100
AbstractKeyAllocatorConstraint (class in pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint), 99
AbstractLinkData (class in spinn_machine.link_data_objects), 17
AbstractLinkData (class in spinn_machine.link_data_objects.abstract_link_data), 16
AbstractMachineAllocationController (class in spinn_front_end_common.abstract_models), 499
AbstractMachineAllocationController (class in spinn_front_end_common.abstract_models.abstract_machine_allocation_controller), 496
AbstractMachinePartitionNKeysMap (class in pacman.model.routing_info), 167
AbstractMachinePartitionNKeysMap (class in pacman.model.routing_info.abstract_machine_partition_n_keys_map), 164
AbstractMasterPopTableFactory (class in spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory), 727
abstractmethod() (in module spinn_utilities.abstract_base), 6
AbstractMulticastControllableDevice (class in spynnaker.pyNN.external_devices_models), 659
AbstractMulticastControllableDevice (class in spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device), 651
AbstractMultiConnectionProcess (class in spinnman.processes), 368
AbstractMultiConnectionProcess (class in spinnman.processes.abstract_multi_connection_process), 361
AbstractMultiConnectionProcessConnectionSelector (class in spinnman.processes.abstract_multi_connection_process_connection_selector), 362
AbstractNeuronModel (class in spynnaker.pyNN.models.neuron.neuron_models), 738
AbstractNeuronModel (class in spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_mod 733
AbstractNeuronRecordable (class in spynnaker.pyNN.models.common), 673
AbstractNeuronRecordable (class in spynnaker.pyNN.models.common.abstract_neuron_recordable), 669
AbstractOutgoingEdgePartition (class in pacman.model.graphs), 145
AbstractOutgoingEdgePartition (class in pacman.model.graphs.abstract_outgoing_edge_partition), 140
AbstractPartitionerConstraint (class in pacman.model.constraints.partitioner_constraints), 103
AbstractPartitionerConstraint (class in pacman.model.constraints.partitioner_constraints.abstract_partitioner 102
AbstractPlacerConstraint (class in pacman.model.constraints.placer_constraints), 105
AbstractPlacerConstraint (class in pacman.model.constraints.placer_constraints.abstract_placer_constra 103
AbstractPlasticSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics), 762
AbstractPlasticSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_s 756
AbstractPopulationInitializable (class in spynnaker.pyNN.models.abstract_models), 668
AbstractPopulationInitializable (class in spynnaker.pyNN.models.abstract_models.abstract_population_initializ 666
AbstractPopulationSettable (class in spynnaker.pyNN.models.abstract_models), 668
AbstractPopulationSettable (class in spynnaker.pyNN.models.abstract_models.abstract_population_settable 667
AbstractPopulationVertex (class in spynnaker.pyNN.models.neuron), 786
AbstractPopulationVertex (class in spynnaker.pyNN.models.neuron.abstract_population_vertex), 778
AbstractProcess (class in spinnman.processes.abstract_process), 362
abstractproperty (class in spinn_utilities.abstract_base), 6
AbstractProvidesIncomingPartitionConstraints (class in spinn_front_end_common.abstract_models),

500
AbstractProvidesIncomingPartitionConstraints (class in spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraints), 496
AbstractProvidesKeyToAtomMapping (class in spinn_front_end_common.abstract_models), 500
AbstractProvidesKeyToAtomMapping (class in spinn_front_end_common.abstract_models.abstract_provides_key_to_atom_mapping), 497
AbstractProvidesLocalProvenanceData (class in spinn_front_end_common.interface.provenance), 544
AbstractProvidesLocalProvenanceData (class in spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data), 543
AbstractProvidesNKeysForPartition (class in spinn_front_end_common.abstract_models), 500
AbstractProvidesNKeysForPartition (class in spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition), 497
AbstractProvidesOutgoingPartitionConstraints (class in spinn_front_end_common.abstract_models), 500
AbstractProvidesOutgoingPartitionConstraints (class in spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints), 497
AbstractProvidesProvenanceDataFromMachine (class in spinn_front_end_common.interface.provenance), 545
AbstractProvidesProvenanceDataFromMachine (class in spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_from_machine), 543
AbstractPushBotOutputDevice (class in spynnaker.pyNN.external_devices_models.push_bot), 649
AbstractPushBotOutputDevice (class in spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device), 649
AbstractPushBotRetinaDevice (class in spynnaker.pyNN.external_devices_models.push_bot), 649
AbstractPushBotRetinaDevice (class in spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device), 649
AbstractPythonAlgorithm (class in pacman.executor.algorithm_classes), 84
AbstractPythonAlgorithm (class in pacman.executor.algorithm_classes.abstract_python_algorithm), 81
AbstractRandomStats (class in spynnaker.pyNN.utilities.random_stats), 828
AbstractRandomStats (class in spynnaker.pyNN.utilities.random_stats.abstract_random_stats), 827
AbstractReadParametersBeforeSet (class in spynnaker.pyNN.models.abstract_models), 668
AbstractReadParametersBeforeSet (class in spynnaker.pyNN.models.abstract_models.abstract_read_parameters_b 667
AbstractReceiveBuffersToHost (class in spinn_front_end_common.interface.buffer_management.buffer_m 505
AbstractReceiveBuffersToHost (class in spinn_front_end_common.interface.buffer_management.buffer_m 502
AbstractRecordable (class in spinn_front_end_common.abstract_models), 501
AbstractRecordable (class in spinn_front_end_common.abstract_models.abstract_recordable), 498
AbstractResource (class in pacman.model.resources), 159
AbstractResource (class in pacman.model.resources.abstract_resource), 152
AbstractRewritesDataSpecification (class in spinn_front_end_common.abstract_models), 501
AbstractRewritesDataSpecification (class in spinn_front_end_common.abstract_models.abstract_rewrites_data 498
AbstractRouterConstraint (class in pacman.model.constraints.router_constraints), 107
AbstractRouterConstraint (class in pacman.model.constraints.router_constraints.abstract_router_constra 107
AbstractSCPRequest (class in spinnman.messages.scp.abstract_messages), 272
AbstractSCPRequest (class in spinnman.messages.scp.abstract_messages.scp_request), 269
AbstractSCPResponse (class in spinnman.messages.scp.abstract_messages), 273
AbstractSCPResponse (class in spinnman.messages.scp.abstract_messages.scp_response), 270
AbstractSendMeMulticastCommandsVertex (class in spinn_front_end_common.abstract_models), 501
AbstractSendMeMulticastCommandsVertex (class in spinn_front_end_common.abstract_models.abstract_send_me_mu 498
AbstractSendsBuffersFromHost (class in spinn_front_end_common.interface.buffer_management.buffer_m 506

AbstractSendsBuffersFromHost (class in spinn_front_end_common.interface.buffer_management.abstract_sends_buffers_from_host), 503
AbstractSingleConnectionProcess (class in spinnman.processes.abstract_single_connection_process), 362
AbstractSpikeRecordable (class in spynnaker.pyNN.models.common), 674
AbstractSpikeRecordable (class in spynnaker.pyNN.models.common.abstract_spike_recordable), 670
AbstractSpinnakerBase (class in spinn_front_end_common.interface.abstract_spinnaker_base), 546
AbstractSpiNNakerCommon (class in spynnaker.pyNN.abstract_spinnaker_common), 833
AbstractSpiNNakerLinkVertex (class in pacman.model.graphs), 146
AbstractSpiNNakerLinkVertex (class in pacman.model.graphs.abstract_spinnaker_link_vertex), 141
AbstractStaticSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics), 761
AbstractStaticSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics), 756
AbstractSupportsDatabaseInjection (class in spinn_front_end_common.abstract_models), 501
AbstractSupportsDatabaseInjection (class in spinn_front_end_common.abstract_models.abstract_supports_database_injection), 498
AbstractSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics), 760
AbstractSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics), 756
AbstractSynapseIO (class in spynnaker.pyNN.models.neuron.synapse_io), 766
AbstractSynapseIO (class in spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io), 765
AbstractSynapseStructure (class in spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure), 744
AbstractSynapseStructure (class in spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure), 743
AbstractSynapseType (class in spynnaker.pyNN.models.neuron.synapse_types), 770
AbstractSynapseType (class in spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type), 767
AbstractTag (class in spinn_machine.tags), 20
AbstractTag (class in spinn_machine.tags.abstract_tag), 18
AbstractThresholdType (class in spynnaker.pyNN.models.neuron.threshold_types), 775
AbstractThresholdType (class in spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_ty 773
AbstractTimingDependence (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 748
AbstractTimingDependence (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abs 745
AbstractUInt32Recorder (class in spynnaker.pyNN.models.common), 674
AbstractUInt32Recorder (class in spynnaker.pyNN.models.common.abstract_uint32_recorder), 670
AbstractVertex (class in pacman.model.graphs), 146
AbstractVertex (class in pacman.model.graphs.abstract_vertex), 141
AbstractVertexWithEdgeToDependentVertices (class in spinn_front_end_common.abstract_models), 501
AbstractVertexWithEdgeToDependentVertices (class in spinn_front_end_common.abstract_models.abstract_vertex_with_ 498
AbstractVirtualVertex (class in pacman.model.graphs), 147
AbstractVirtualVertex (class in pacman.model.graphs.abstract_virtual_vertex), 142
AbstractWeightDependence (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 754
AbstractWeightDependence (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.ab 751
AbstractWeightUpdatable (class in spynnaker.pyNN.models.abstract_models), 669
AbstractWeightUpdatable (class in spynnaker.pyNN.models.abstract_models.abstract_weight_updatable), 667
activate_live_output_for() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnaI static method), 835
activate_live_output_to() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnaI

static method), 836
ADC (spinnman.messages.scp.enums.bmp_info.BMPInfo attribute), 274
ADC (spinnman.messages.scp.enums.BMPInfo attribute), 278
ADCInfo (class in spinnman.model), 347
ADCInfo (class in spinnman.model.adc_info), 331
ADD (data_specification.enums.arithemetic_operation.ArithmeticOperation attribute), 414
ADD (data_specification.enums.ArithmeticOperation attribute), 417
add() (spinn_machine.utilities.ordered_set.OrderedSet method), 21
add() (spinn_utilities.ordered_set.OrderedSet method), 10
add_application_edge() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 546
add_application_vertex() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 547
add_application_vertex() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon method), 833
add_application_vertex() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager static method), 836
add_application_vertices() (spinn_front_end_common.utilities.database.DatabaseWriter method), 557
add_application_vertices() (spinn_front_end_common.utilities.database.DatabaseWriter method), 560
add_callback() (spinnman.connections.connection_listener.ConnectionListener method), 243
add_callback() (spinnman.connections.ConnectionListener method), 245
add_chip() (spinn_machine.Machine method), 47
add_chip() (spinn_machine.machine.Machine method), 30
add_chip() (spinn_machine.virtual_machine.VirtualMachine method), 39
add_chip() (spinn_machine.VirtualMachine method), 56
add_chips() (spinn_machine.Machine method), 47
add_chips() (spinn_machine.machine.Machine method), 30
add_command_container() (spynnaker.pyNN.connections.ethernet_command_connection.EthernetCommandConnection method), 618
add_command_container() (spynnaker.pyNN.connections.EthernetCommandConnection method), 620
add_commands() (spinn_front_end_common.utility_models.commandsender.CommandSender method), 588
add_commands() (spinn_front_end_common.utility_models.CommandSender method), 603
add_connections() (spynnaker.pyNN.models.neuron.connection_holder.ConnectionHolder method), 783
add_connections() (spynnaker.pyNN.models.neuron.ConnectionHolder method), 791
add_constraint() (pacman.model.graphs.abstract_graph.AbstractGraph method), 137
add_constraint() (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition method), 140
add_constraint() (pacman.model.graphs.abstract_vertex.AbstractVertex method), 141
add_constraint() (pacman.model.graphs.AbstractGraph method), 143
add_constraint() (pacman.model.graphs.AbstractOutgoingEdgePartition method), 145
add_constraint() (pacman.model.graphs.AbstractVertex method), 146
add_constraint() (pacman.model.graphs.common.constrained_object.ConstrainedObject method), 118
add_constraint() (pacman.model.graphs.common.ConstrainedObject method), 121
add_constraints() (pacman.model.graphs.abstract_graph.AbstractGraph method), 137
add_constraints() (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition method), 140
add_constraints() (pacman.model.graphs.abstract_vertex.AbstractVertex method), 141
add_constraints() (pacman.model.graphs.AbstractGraph method), 143
add_constraints() (pacman.model.graphs.AbstractOutgoingEdgePartition method), 145
add_constraints() (pacman.model.graphs.AbstractVertex method), 147
add_constraints() (pacman.model.graphs.common.constrained_object.ConstrainedObject method), 119
add_constraints() (pacman.model.graphs.common.ConstrainedObject method), 121
add_core_subset() (spinn_machine.core_subsets.CoreSubsets method), 26
add_core_subset() (spinn_machine.CoreSubsets method), 44
add_data() (spinn_front_end_common.interface.profiling.profile_data.ProfileData method), 541
add_database_callback() (spinn_front_end_common.utilities.database.DatabaseConnection method), 555
add_database_callback() (spinn_front_end_common.utilities.database.DatabaseConnection method), 559
add_database_socket_address() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager static method), 836
add_delay() (spynnaker.pyNN.models.utility_models.delay_block.DelayBlock method), 806

add_delay() (spynnaker.pyNN.models.utility_models.DelayBlock method), 810
add_delays() (spynnaker.pyNN.models.utility_models.delay_extension_vertex method), 807
add_delays() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 811
add_edge() (pacman.model.graphs.abstract_graph.AbstractGraph method), 138
add_edge() (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition method), 140
add_edge() (pacman.model.graphs.AbstractGraph method), 143
add_edge() (pacman.model.graphs.AbstractOutgoingEdgePartition method), 145
add_edge() (pacman.model.graphs.impl.Graph method), 127
add_edge() (pacman.model.graphs.impl.graph.Graph method), 123
add_edge() (pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition method), 125
add_edge() (pacman.model.graphs.impl.OutgoingEdgePartition method), 129
add_edge() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager static method), 836
add_edge_mapping() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 119
add_edge_mapping() (pacman.model.graphs.common.GraphMapper method), 121
add_edges() (pacman.model.graphs.abstract_graph.AbstractGraph method), 138
add_edges() (pacman.model.graphs.AbstractGraph method), 143
add_element() (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage method), 259
add_element() (spinnman.messages.eieio.data_messages.EIEIODataMessage method), 263
add_extraction_timing() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 547
add_field_constraints() (pacman.operations.routing_info_allocator_algorithms.field_based_routing_allocator.VertexMulticastRoutingInfo static method), 192
add_fpga_links() (spinn_machine.Machine method), 47
add_fpga_links() (spinn_machine.machine.Machine method), 30
add_init_callback() (spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection method), 551
add_init_callback() (spinn_front_end_common.utilities.connections.LiveEventConnection method), 553
add_ip_tag() (pacman.model.tags.Tags method), 179
add_ip_tag() (pacman.model.tags.tags.Tags method), 178
add_item() (spynnaker.pyNN.utilities.running_stats.RunningStats method), 829
add_items() (spynnaker.pyNN.utilities.running_stats.RunningStats method), 829
add_key() (spinn_front_end_common.interface.buffer_management.storage_extension.vertex_DelayExtensionVertex method), 513
add_key() (spinn_front_end_common.interface.buffer_management.storage_extension.vertex method), 520
add_key() (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage method), 259
add_key() (spinnman.messages.eieio.data_messages.EIEIODataMessage method), 264
add_key_and_payload() (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage method), 260
add_key_and_payload() (spinnman.messages.eieio.data_messages.EIEIODataMessage method), 264
add_keys() (spinn_front_end_common.interface.buffer_management.storage_extension.vertex_DelayExtensionVertex method), 513
add_keys() (spinn_front_end_common.interface.buffer_management.storage_extension.vertex method), 520
add_link() (spinn_machine.Router method), 53
add_link() (spinn_machine.router.Router method), 36
add_live_packet_gather_parameters() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 547
add_machine_edge() (spinn_front_end_common.interface.abstract_spinnaker_base method), 547
add_machine_objects() (spinn_front_end_common.utilities.database.database method), 557
add_machine_objects() (spinn_front_end_common.utilities.database.Database method), 560
add_machine_vertex() (spinn_front_end_common.interface.abstract_spinnaker_base method), 547
add_message_to_send() (spinn_front_end_common.interface.buffer_management.storage_extension.EIEIODataMessage method), 514
add_message_to_send() (spinn_front_end_common.interface.buffer_management.storage_extension.vertex method), 522
add_multicast_routing_entry() (pacman.model.routing_table.MulticastRoutingTable method), 174
add_multicast_routing_entry() (pacman.model.routing_table.MulticastRoutingTable method), 176
add_outgoing_edge_partition() (pacman.model.graphs.abstract_graph.AbstractGraph method), 138
add_outgoing_edge_partition() (pacman.model.graphs.AbstractGraph method), 142
add_outgoing_edge_partition() (pacman.model.graphs.impl.Graph method), 127
add_outgoing_edge_partition() (pacman.model.graphs.impl.graph.Graph method),

123
add_partition_info() (pacman.model.routing_info.routing_info.RoutingInfo method), 166
add_partition_info() (pacman.model.routing_info.RoutingInfo method), 169
add_path() (spinn_utilities.executable_finder.ExecutableFinder method), 9
add_path_entry() (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartition method), 170
add_path_entry() (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartition method), 172
add_pause_stop_callback() (spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection method), 552
add_pause_stop_callback() (spinn_front_end_common.utilities.connections.LiveEventConnection method), 553
add_payload_logic_to_current_output() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 818
add_payload_logic_to_current_output() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 823
add_payload_logic_to_current_output_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 818
add_payload_logic_to_current_output_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 823
add_placement() (pacman.model.placements.Placements method), 151
add_placement() (pacman.model.placements.placements.Placements method), 148
add_placement_constraint() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 814
add_placements() (pacman.model.placements.Placements method), 151
add_placements() (pacman.model.placements.placements.Placements method), 149
add_placements() (spinn_front_end_common.utilities.database.database_writer.DatabaseWriter method), 557
add_placements() (spinn_front_end_common.utilities.database.database_writer.DatabaseWriter method), 560
add_population() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon method), 833
add_pre_run_connection_holder() (spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses method), 665
add_pre_run_connection_holder() (spynnaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSy method), 667
add_pre_run_connection_holder() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP method), 779
add_pre_run_connection_holder() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 787
add_pre_run_connection_holder() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
add_pre_run_connection_holder() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
add_processor() (spinn_machine.core_subset.CoreSubset method), 25
add_processor() (spinn_machine.core_subsets.CoreSubsets method), 26
add_processor() (spinn_machine.CoreSubset method), 43
add_processor() (spinn_machine.CoreSubsets method), 44
add_processor() (spinnman.model.cpu_infos.CPUInfos method), 338
add_processor() (spinnman.model.CPUInfos method), 353
add_processor() (spinnman.model.executable_targets.ExecutableTargets method), 340
add_processor() (spinnman.model.ExecutableTargets method), 355
add_projection() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCon method), 833
add_receive_callback() (spinn_front_end_common.utilities.connections.live method), 552
add_receive_callback() (spinn_front_end_common.utilities.connections.Liv method), 554
add_receiving_vertex() (spinn_front_end_common.interface.buffer_manage method), 524
add_receiving_vertex() (spinn_front_end_common.interface.buffer_manage method), 529
add_reverse_ip_tag() (pacman.model.tags.Tags method), 179
add_reverse_ip_tag() (pacman.model.tags.tags.Tags method), 178
add_routing_infos() (spinn_front_end_common.utilities.database.database_ method), 557
add_routing_infos() (spinn_front_end_common.utilities.database.DatabaseW method), 561
add_routing_table() (pacman.model.routing_tables.multicast_routing_tables.MulticastRou method), 175
add_routing_table() (pac-

man.model.routing_tables.MulticastRoutingTables method), 177
add_routing_tables() (spinn_front_end_common.utilities.database.database_writer.DatabaseWriter method), 557
add_routing_tables() (spinn_front_end_common.utilities.database.DatabaseWriter method), 561
add_sender_vertex() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 524
add_sender_vertex() (spinn_front_end_common.interface.buffer_management.BufferManager method), 529
add_socket_address() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 547
add_socket_address() (spinn_front_end_common.utilities.failed_state.FailedState static method), 578
add_socket_address() (spinn_front_end_common.utilities.FailedState static method), 583
add_socket_address() (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface method), 582
add_socket_address() (spinn_front_end_common.utilities.SimulatorInterface method), 583
add_socket_address() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager static method), 836
add_spinnaker_links() (spinn_machine.Machine method), 47
add_spinnaker_links() (spinn_machine.machine.Machine method), 30
add_start_callback() (spinn_front_end_common.utilities.connections.LiveEventConnection method), 552
add_start_callback() (spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection method), 554
add_start_resume_callback() (spinn_front_end_common.utilities.connections.LiveEventConnection method), 552
add_start_resume_callback() (spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection method), 554
add_subsets() (spinnman.model.executable_targets.ExecutableTargets method), 340
add_subsets() (spinnman.model.ExecutableTargets method), 355
add_synapse_information() (spynnaker.pyNN.models.neural_projections.delayed_application_edge.DelayedApplicationEdge method), 696
add_synapse_information() (spynnaker.pyNN.models.neural_projections.DelayedApplicationEdge method), 699
add_synapse_information() (spynnaker.pyNN.models.neural_projections.projection_application_edge.ProjectionApplicationEdge method), 697
add_synapse_information() (spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge method), 700
add_system_params() (spinn_front_end_common.utilities.database.database_writer.DatabaseWriter method), 557
add_system_params() (spinn_front_end_common.utilities.database.DatabaseWriter method), 561
add_tags() (spinn_front_end_common.utilities.database.database_writer.DatabaseWriter method), 557
add_tags() (spinn_front_end_common.utilities.database.DatabaseWriter method), 561
add_vertex() (pacman.model.graphs.abstract_graph.AbstractGraph method), 138
add_vertex() (pacman.model.graphs.AbstractGraph method), 144
add_vertex() (pacman.model.graphs.impl.Graph method), 127
add_vertex() (pacman.model.graphs.impl.graph.Graph method), 123
add_vertex_mapping() (pacman.model.graphs.interface.IGraphMapper method), 119
add_vertex_mapping() (pacman.model.graphs.common.GraphMapper method), 122
add_vertex() (pacman.model.graphs.abstract_graph.AbstractGraph method), 138
add_vertices() (pacman.model.graphs.AbstractGraph method), 144
add_vertices() (spinn_front_end_common.utilities.database.database_writer.DatabaseWriter method), 558
add_vertices() (spinn_front_end_common.utilities.database.DatabaseWriter method), 561
AdditionalInputCurrentAdaptive (class in spynnaker.pyNN.models.neuron.additional_inputs), 704
AdditionalInputCurrentAdaptive (class in spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca 703
address (spinn_utilities.io.abstract_io.AbstractIO attribute), 372
address (spinnman.utilities.io.AbstractIO attribute), 375
address (spinnman.utilities.io.file_io.FileIO attribute), 373
address (spinnman.utilities.io.FileIO attribute), 376
address (spinnman.utilities.io.memory_io.MemoryIO attribute), 375
address (spinnman.utilities.io.MemoryIO attribute), 377
ADDRESS_LIST_DTYPE (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ attribute), 730
ADDRESS_LIST_DTYPE (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterP attribute), 732
ADDRESS_MASK (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ attribute), 730
ADDRESS_MASK (spyn-

naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch attribute), 732
algorithm (pacman.exceptions.PacmanAlgorithmFailedToCompleteException attribute), 211
algorithm() (in module pacman.executor.algorithm_decorators), 93
algorithm() (in module pacman.executor.algorithm_decorators.algorithm_decorator), 88
algorithm_id (pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgorithm attribute), 80
algorithm_id (pacman.executor.algorithm_classes.AbstractAlgorithm attribute), 84
algorithm_timings (pacman.executor.pacman_algorithm_executor.PACMANAlgorithmExecutor attribute), 96
algorithm_timings (pacman.executor.PACMANAlgorithmExecutor attribute), 98
AlgorithmMetadataXmlReader (class in pacman.executor), 97
AlgorithmMetadataXmlReader (class in pacman.executor.algorithm_metadata_xml_reader), 94
algorithms() (in module pacman.executor.algorithm_decorators), 94
algorithms() (in module pacman.executor.algorithm_decorators.algorithm_decorator), 88
ALIGN_WR_PTR (data_specification.enums.Commands attribute), 417
ALIGN_WR_PTR (data_specification.enums.commands.Commands attribute), 414
align_write_pointer() (data_specification.data_specification_generator.DataSpecificationGenerator method), 425
align_write_pointer() (data_specification.DataSpecificationGenerator method), 463
all() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 814
all_core_subsets (spinnman.model.executable_targets.ExecutableTargets attribute), 340
all_core_subsets (spinnman.model.ExecutableTargets attribute), 355
all_modules() (in module spinn_utilities.package_loader), 11
ALLOC_ROUTING (spinnman.messages.scp.enums.alloc_free.AllocFree attribute), 274
ALLOC_ROUTING (spinnman.messages.scp.enums.AllocFree attribute), 277
ALLOC_SDRAM (spinnman.messages.scp.enums.alloc_free.AllocFree attribute), 274
ALLOC_SDRAM (spinnman.messages.scp.enums.AllocFree attribute), 277
allocate_constrained_group_resources() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 200
allocate_constrained_group_resources() (pacman.utilities.utility_objs.ResourceTracker method), 204
allocate_constrained_resources() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 200
allocate_constrained_resources() (pacman.utilities.utility_objs.ResourceTracker method), 204
allocate_group_resources() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 200
allocate_group_resources() (pacman.utilities.utility_objs.ResourceTracker method), 204
allocate_id() (spinnman.utilities.appid_tracker.AppIdTracker method), 378
allocate_resources() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 201
allocate_resources() (pacman.utilities.utility_objs.ResourceTracker method), 205
allocated_size (data_specification.memory_region.MemoryRegion attribute), 455
allocated_size (data_specification.MemoryRegion attribute), 438
allocated_tag_table_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 317
allocated_tag_table_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
AllocFree (class in spinnman.messages.scp.enums), 277
AllocFree (class in spinnman.messages.scp.enums.alloc_free), 274
AllOf (class in pacman.executor.algorithm_decorators), 92
AllOf (class in pacman.executor.algorithm_decorators.algorithm_decorator) 87
AllOfInput (class in pacman.executor.algorithm_decorators), 91
AllOfInput (class in pacman.executor.algorithm_decorators.all_of_input), 89
allow_self_connections (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_con

attribute), 678
allow_self_connections (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector attribute), 688
allow_self_connections (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector attribute), 680
allow_self_connections (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector attribute), 689
allow_self_connections (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector attribute), 680
allow_self_connections (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector attribute), 690
AllToAllConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 688
AllToAllConnector (class in spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector), 678
AND (data_specification.enums.logic_operation.LogicOperation attribute), 417
AND (data_specification.enums.LogicOperation attribute), 420
API (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
API (spinnman.model.enums.RunTimeError attribute), 329
app_data_table_address (spinnman.messages.spinnaker_boot.system_variable_boot_messages.SystemVariableBoot attribute), 317
app_data_table_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
app_id (spinnman.model.heap_element.HeapElement attribute), 341
app_id (spinnman.model.HeapElement attribute), 356
app_id_tracker (spinnman.transceiver.Transceiver attribute), 384
AppIdTracker (class in spinnman.utilities.appid_tracker), 378
APPLICATION (pacman.utilities.utility_objs.flexi_field.SUPPORTED_TAGS attribute), 199
application_graph (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 547
application_id (spinnman.model.cpu_info.CPUInfo attribute), 336
application_id (spinnman.model.CPUInfo attribute), 351
application_mailbox_command (spinnman.model.cpu_info.CPUInfo attribute), 336
application_mailbox_command (spinnman.model.CPUInfo attribute), 351
application_mailbox_data_address (spinnman.model.cpu_info.CPUInfo attribute), 336
application_mailbox_data_address (spinnman.model.CPUInfo attribute), 351
application_name (spinnman.model.cpu_info.CPUInfo attribute), 336
application_name (spinnman.model.CPUInfo attribute), 351
ApplicationEdge (class in pacman.model.graphs.application), 114
ApplicationEdge (class in pacman.model.graphs.application.application_edge), 109
ApplicationFinisher (class in spinn_front_end_common.interface.interface_functions), 537
ApplicationFinisher (class in spinn_front_end_common.interface.interface_functions.application_finisher), 530
ApplicationFPGAVertex (class in pacman.model.graphs.application), 115
ApplicationFPGAVertex (class in pacman.model.graphs.application.application_fpga_vertex), 110
ApplicationGraph (class in pacman.model.graphs.application), 116
ApplicationGraph (class in pacman.model.graphs.application.application_graph), 111
ApplicationOutgoingEdgePartition (class in pacman.model.graphs.application), 116
ApplicationOutgoingEdgePartition (class in pacman.model.graphs.application.application_outgoing_edge_partition), 111
ApplicationRun (class in spinnman.messages.scp.impl), 296
ApplicationRun (class in spinnman.messages.scp.impl.application_run), 281
ApplicationRunner (class in spinn_front_end_common.interface.interface_functions), 537
ApplicationRunner (class in spinn_front_end_common.interface.interface_functions.application_runner), 530
ApplicationRunProcess (class in spinnman.processes), 368
ApplicationRunProcess (class in spinnman.processes.application_run_process), 362
ApplicationSpiNNakerLinkVertex (class in pacman.model.graphs.application), 117

ApplicationSpiNNakerLinkVertex (class in pacman.model.graphs.application.application_spinnaker_link_vertex), 112
ApplicationVertex (class in pacman.model.graphs.application), 116
ApplicationVertex (class in pacman.model.graphs.application.application_vertex), 113
AppStop (class in spinnman.messages.scp.impl), 296
AppStop (class in spinnman.messages.scp.impl.app_stop), 280
ArbitraryFPGADevice (class in spynnaker.pyNN.external_devices_models), 659
ArbitraryFPGADevice (class in spynnaker.pyNN.external_devices_models.arbitrary_fpga_device), 652
are_weights_signed() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
are_weights_signed() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamicsStatic method), 760
are_weights_signed() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
are_weights_signed() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 759
are_weights_signed() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 762
are_weights_signed() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 763
argument_1 (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 272
argument_1 (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest attribute), 270
argument_2 (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 272
argument_2 (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest attribute), 270
argument_3 (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 272
argument_3 (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest attribute), 270
ARITH_OP (data_specification.enums.Commands attribute), 417
ARITH_OP (data_specification.enums.commands.Commands attribute), 414
ArithmeticOperation (class in data_specification.enums), 417
ArithmeticOperation (class in data_specification.enums.arithemetic_operation), 414
array_size (spinnman.messages.spinnaker_boot.system_variable_boot_value attribute), 317
array_size (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
assert_logs_contains() (in module spinn_utilities.testing.log_checker), 5
assert_logs_contains_once() (in module spinn_utilities.testing.log_checker), 5
assert_logs_error_contains() (in module spinn_utilities.testing.log_checker), 5
assert_logs_error_not_contains() (in module spinn_utilities.testing.log_checker), 5
assert_logs_info_contains() (in module spinn_utilities.testing.log_checker), 5
assert_logs_info_not_contains() (in module spinn_utilities.testing.log_checker), 6
assert_logs_not_contains() (in module spinn_utilities.testing.log_checker), 6
auto_detect_database() (spinn_front_end_common.utilities.database.DatabaseConnection static method), 558
auto_detect_database() (spinn_front_end_common.utilities.database.Database static method), 561

B

b (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh attribute), 739
b (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 740
BAEL_BALANCER (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 818
BAEL_BALANCER (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES attribute), 823
base_address (spinnman.processes.malloc_sdram_process.MallocSDRAMProcessRequest attribute), 365
base_address (spinnman.processes.MallocSDRAMProcess attribute), 370
BaseKeyAndMask (class in pacman.model.routing_info), 168
BaseKeyAndMask (class in pacman.model.routing_info.base_key_and_mask), 164
BASIC_MALLOC_USAGE (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 778
BASIC_MALLOC_USAGE (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 786
BasicDijkstraRouting (class in pacman.operations.router_algorithms), 190
BasicDijkstraRouting (class in pacman.operations.router_algorithms.basic_dijkstra_routing),

190
BasicPartitioner (class in pacman.operations.partition_algorithms), 187
BasicPartitioner (class in pacman.operations.partition_algorithms.basic_partitioner), 186
BasicPlacer (class in pacman.operations.placer_algorithms), 188
BasicPlacer (class in pacman.operations.placer_algorithms.basic_placer), 187
BasicRouteMerger (class in pacman.operations.router_compressors.basic_route_merger), 191
BasicRoutingInfoAllocator (class in pacman.operations.routing_info_allocator_algorithms), 194
BasicRoutingInfoAllocator (class in pacman.operations.routing_info_allocator_algorithms.basic_routing_info_allocator), 193
BasicRoutingTableGenerator (class in pacman.operations.routing_table_generators.basic_routing_table_generator), 194
BasicTagAllocator (class in pacman.operations.tag_allocator_algorithms), 194
BasicTagAllocator (class in pacman.operations.tag_allocator_algorithms.basic_tag_allocator), 194
bias_values() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 818
bias_values() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 823
bias_values_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 818
bias_values_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 823
binaries (spinnman.model.executable_targets.ExecutableTargets attribute), 340
binaries (spinnman.model.ExecutableTargets attribute), 355
BINARY_FILE_NAME (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 589
BINARY_FILE_NAME (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 605
binary_file_name() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex static method), 586
binary_paths (spinn_utilities.executable_finder.ExecutableFinder attribute), 9
binary_start_type() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex static method), 586
bits_per_coordinate (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey attribute), 822
bits_per_coordinate (spynnaker.pyNN.protocols.RetinaKey attribute), 827
block_address (spinnman.model.heap_element.HeapElement attribute), 341
block_address (spinnman.model.HeapElement attribute), 356
BLOCK_COPY (data_specification.enums.Commands attribute), 418
BLOCK_COPY (data_specification.enums.commands.Commands attribute), 414
bmp_connection (spinnman.transceiver.Transceiver attribute), 384
BMPConnection (class in spinnman.connections.udp_packet_connections), 237
BMPConnection (class in spinnman.connections.udp_packet_connections.bmp_connection), 230
BMPConnectionData (class in spinnman.model), 347
BMPConnectionData (class in spinnman.model.bmp_connection_data), 332
BMPGetVersion (class in spinnman.messages.scp.impl), 297
BMPGetVersion (class in spinnman.messages.scp.impl.bmp_get_version), 281
BMPInfo (class in spinnman.messages.scp.enums), 278
BMPInfo (class in spinnman.messages.scp.enums.bmp_info), 274
BMPRequest (class in spinnman.messages.scp.abstract_messages), 271
BMPRequest (class in spinnman.messages.scp.abstract_messages.bmp_request), 268
BMPResponse (class in spinnman.messages.scp.abstract_messages), 271
BMPResponse (class in spinnman.messages.scp.abstract_messages.bmp_response), 269
BMPSetLed (class in spinnman.messages.scp.impl), 296
BMPSetLed (class in spinnman.messages.scp.impl.bmp_set_led), 281
board (spinnman.model.resources.specific_board_iptag_resource.SpecificBoardIptagResource attribute), 157
board (spinnman.model.resources.specific_board_iptag_resource.SpecificBoardIptagResource attribute), 158
BOARD_48_CHIP_GAPS (spinn_machine.Machine attribute), 47
BOARD_48_CHIP_GAPS (spinn_machine.machine.Machine attribute), 29

board_address (pacman.model.constraints.placer_constraints.board_constraint.BoardConstraint attribute), 104
board_address (pacman.model.constraints.placer_constraints.board_constraint.BoardConstraint attribute), 105
board_address (pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex attribute), 142
board_address (pacman.model.graphs.AbstractVirtualVertex attribute), 147
board_address (pacman.model.graphs.application.application_fpga_vertex.ApplicationFPGAVertex attribute), 110
board_address (pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex attribute), 112
board_address (pacman.model.graphs.application.ApplicationFPGAVertex attribute), 115
board_address (pacman.model.graphs.application.ApplicationSpiNNakerLinkVertex attribute), 117
board_address (pacman.model.graphs.machine.machine_fpga_vertex.MachineFPGAVertex attribute), 131
board_address (pacman.model.graphs.machine.machine_spinnaker_link_vertex.MachineSpiNNakerLinkVertex attribute), 132
board_address (pacman.model.graphs.machine.MachineFPGAVertex attribute), 134
board_address (pacman.model.graphs.machine.MachineSpiNNakerLinkVertex attribute), 135
board_address (spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.LivePacketGatherParameters attribute), 573
board_address (spinn_front_end_common.utilities.utility_objs.LivePacketGatherParameters attribute), 576
board_address (spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData attribute), 16
board_address (spinn_machine.link_data_objects.AbstractLinkData attribute), 17
board_address (spinn_machine.tags.abstract_tag.AbstractTag attribute), 18
board_address (spinn_machine.tags.AbstractTag attribute), 20
board_info (spinnman.messages.spinnaker_boot.system_variable_boot.SystemVariableDefinition attribute), 317
board_info (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
board_test_flags (spinnman.messages.spinnaker_boot.system_variable_boot.SystemVariableDefinition attribute), 317
board_test_flags (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
BoardConstraint (class in pacman.model.constraints.placer_constraints), 105
BoardConstraint (class in pacman.model.constraints.placer_constraints.board_constraint), 104
boards (spinnman.connections.udp_packet_connections.bmp_connection.BMPConnection attribute), 230
boards (spinnman.connections.udp_packet_connections.BMPConnection attribute), 237
boards (spinnman.model.bmp_connection_data.BMPConnectionData attribute), 332
boards (spinnman.model.BMPConnectionData attribute), 347
boot_board() (spinnman.transceiver.Transceiver method), 384
boot_image (spinn_machine.FPGAVertex attribute), 47
boot_chip (spinn_machine.machine.Machine attribute), 30
boot_signature (spinnman.messages.spinnaker_boot.system_variable_boot.SystemVariableDefinition attribute), 317
boot_signature (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
boot_x (spinn_machine.Machine attribute), 47
boot_y (spinn_machine.Machine attribute), 47
boot_x (spinn_machine.machine.Machine attribute), 30
boot_y (spinn_machine.machine.Machine attribute), 30
BootConnection (class in spinnman.connections.udp_packet_connections), 237
BootConnection (class in spinnman.connections.udp_packet_connections.boot_connection), 281
BREAK (data_specification.enums.Commands attribute), 418
BREAK (data_specification.enums.commands.Commands attribute), 414
BREAK_LOOP (data_specification.enums.Commands attribute), 418
BREAK_LOOP (data_specification.enums.commands.Commands attribute), 414
break_loop() (data_specification.data_specification_generator.DataSpecificationGenerator method), 426
break_loop() (data_specification.DataSpecificationGenerator method), 463
buffer_manager (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 547
buffer_manager (spinn_front_end_common.utilities.failed_state.FailedState attribute), 578
buffer_manager (spinn_front_end_common.utilities.FailedState attribute), 583
buffer_manager (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 582
buffer_manager (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
BUFFER_OVERFLOW_COUNT (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex attribute), 784
BUFFER_OVERFLOW_COUNT (spynnaker.pyNN.models.neuron.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ITEMS attribute), 792
BUFFER_READ (spinn_front_end_common.utilities.constants.BUFFERING

attribute), 577
buffer_size (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region.BufferedSendingRegion attribute), 513
buffer_size (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedSendingRegion attribute), 521
BUFFER_WRITE (spinn_front_end_common.utilities.constants.BUFFERING_OPERATIONS attribute), 577
BufferableRegionTooSmall, 578
BufferedBytearrayDataStorage (class in spinn_storage_handlers), 72
BufferedBytearrayDataStorage (class in spinn_storage_handlers.buffered_bytearray_data_storage), 69
BufferedBytearrayOperationNotImplemented, 71
BufferedFileDataStorage (class in spinn_storage_handlers), 73
BufferedFileDataStorage (class in spinn_storage_handlers.buffered_file_data_storage), 70
BufferedReceivingData (class in spinn_front_end_common.interface.buffer_management.storage_objects), 516
BufferedReceivingData (class in spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data), 508
BufferedRegionNotPresent, 578
BufferedSendingRegion (class in spinn_front_end_common.interface.buffer_management.storage_objects), 520
BufferedSendingRegion (class in spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region), 512
BufferedTempfileDataStorage (class in spinn_storage_handlers), 73
BufferedTempfileDataStorage (class in spinn_storage_handlers.buffered_tempfile_data_storage), 70
BufferExtractor (class in spinn_front_end_common.interface.interface_functions), 537
BufferExtractor (class in spinn_front_end_common.interface.interface_functions.buffer_extractor), 530
buffering_input() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 503
buffering_input() (spinn_front_end_common.interface.buffer_management.buffer_manager.AbstractBufferManager method), 506
buffering_input() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 504
buffering_input() (spinn_front_end_common.interface.buffer_management.buffer_manager.AbstractBufferManager method), 507
BUFFERING_OPERATIONS (class in spinn_front_end_common.utilities.constants), 577
buffering_out_fsm_state (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region.BufferedSendingRegion attribute), 516
buffering_out_fsm_state (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedSendingRegion attribute), 524
BufferManager (class in spinn_front_end_common.interface.buffer_management), 528
BufferManager (class in spinn_front_end_common.interface.buffer_management.buffer_manager), 524
BufferManagerCreator (class in spinn_front_end_common.interface.interface_functions), 537
BufferManagerCreator (class in spinn_front_end_common.interface.interface_functions.buffer_manager), 530
BuffersSentDeque (class in spinn_front_end_common.interface.buffer_management.storage_objects), 522
BuffersSentDeque (class in spinn_front_end_common.interface.buffer_management.storage_objects), 514
build_date (spinnman.model.version_info.VersionInfo attribute), 346
build_date (spinnman.model.VersionInfo attribute), 360
BW_PER_ROUTE_ENTRY (pacman.operations.router_algorithms.basic_dijkstra_routing.BasicDijkstraRoutingAlgorithm attribute), 190
BW_PER_ROUTE_ENTRY (pacman.operations.router_algorithms.BasicDijkstraRoutingAlgorithm attribute), 190
BYTE (spinnman.processes.fill_process.FillDataType attribute), 363
BYTE (spinnman.processes.FillDataType attribute), 369
BYTES_TILL_START_OF_GLOBAL_PARAMETERS (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 779
BYTES_TILL_START_OF_GLOBAL_PARAMETERS (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 786
bytestring (spinnman.messages.eieio.command_messages.database_confirm_message.DatabaseConfirmMessage attribute), 248
bytestring (spinnman.messages.eieio.command_messages.DatabaseConfirmMessage attribute), 253
bytestring (spinnman.messages.eieio.command_messages.eieio_command_message.AbstractSendsBuffersFromHost attribute), 248
bytestring (spinnman.messages.eieio.command_messages.eieio_command_message.AbstractSendsBuffersFromHost attribute), 253
bytestring (spinnman.messages.eieio.command_messages.eieio_command_message.sends_buffers_from_host_pre_buffered_impl.SendsBuffersFromHostPreBufferedImpl attribute), 249
bytestring (spinnman.messages.eieio.command_messages.EIEIOCommandMessage attribute), 254
bytestring (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead attribute), 250

bytestring (spinnman.messages.eieio.command_messages.host_data_read_ack.HostDataReadAck attribute), 250
bytestring (spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData attribute), 251
bytestring (spinnman.messages.eieio.command_messages.HostDataRead attribute), 255
bytestring (spinnman.messages.eieio.command_messages.HostDataReadAck attribute), 256
bytestring (spinnman.messages.eieio.command_messages.HostSendSequencedData attribute), 255
bytestring (spinnman.messages.eieio.command_messages.notification_protocol_pause_stop.NotificationProtocolPauseStop attribute), 251
bytestring (spinnman.messages.eieio.command_messages.notification_protocol_start_resume.NotificationProtocolStartResume attribute), 251
bytestring (spinnman.messages.eieio.command_messages.NotificationProtocolPauseStop attribute), 255
bytestring (spinnman.messages.eieio.command_messages.NotificationProtocolStartResume attribute), 255
bytestring (spinnman.messages.eieio.command_messages.spinnaker_request_buffers.SpinnakerRequestBuffers attribute), 252
bytestring (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData attribute), 252
bytestring (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffers attribute), 256
bytestring (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData attribute), 256
bytestring (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader attribute), 258
bytestring (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage attribute), 260
bytestring (spinnman.messages.eieio.data_messages.EIEIODataHeader attribute), 262
bytestring (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 264
bytestring (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 272
bytestring (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest attribute), 270
bytestring (spinnman.messages.scp.impl.flood_fill_data.FloodFillData attribute), 284
bytestring (spinnman.messages.scp.impl.FloodFillData attribute), 298
bytestring (spinnman.messages.scp.impl.write_link.WriteLink attribute), 295
bytestring (spinnman.messages.scp.impl.write_memory.WriteMemory attribute), 296
bytestring (spinnman.messages.scp.impl.WriteLink attribute), 306
bytestring (spinnman.messages.scp.impl.WriteMemory attribute), 306
bytestring (spinnman.messages.scp.scp_request_header.SCPRequestHeader attribute), 307
bytestring (spinnman.messages.scp.SCPRequestHeader attribute), 309
bytestring (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 310
bytestring (spinnman.messages.sdp.sdp_message.SDPMessage attribute), 312
bytestring (spinnman.messages.sdp.SDPHeader attribute), 313
bytestring (spinnman.messages.sdp.SDPMessage attribute), 314
bytestring (spinnman.messages.spinnaker_boot.spinnaker_boot_message.Sp attribute), 315
bytestring (spinnman.messages.spinnaker_boot.SpinnakerBootMessage attribute), 320
bytestring (spinnman.messages.spinnaker_boot.system_variable_boot_value attribute), 317

C

c (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.Neur attribute), 735
ic (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 740
IC_MAIN (spinnman.model.enums.cpu_state.CPUState attribute), 325
IC_MAIN (spinnman.model.enums.CPUState attribute), 328
cabinet (spinnman.connections.udp_packet_connections.bmp_connection.B attribute), 230
cabinet (spinnman.connections.udp_packet_connections.BMPConnection attribute), 237
cabinet (spinnman.model.bmp_connection_data.BMPConnectionData attribute), 332
cabinet (spinnman.model.BMPConnectionData attribute), 348
call() (pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgo method), 80
call() (pacman.executor.algorithm_classes.abstract_python_algorithm.Abstr method), 81
call() (pacman.executor.algorithm_classes.AbstractAlgorithm method), 84
call() (pacman.executor.algorithm_classes.AbstractPythonAlgorithm method), 85
call() (pacman.executor.algorithm_classes.external_algorithm.ExternalAlgo method), 82
call() (pacman.executor.algorithm_classes.ExternalAlgorithm method), 85
call_arithmetic_operation() (data_specification.data_specification_generator.DataSpecificatio method), 426
call_arithmetic_operation() (data_specification.DataSpecificationGenerator method), 464
call_function() (data_specification.data_specification_generator.DataSpecifi method), 426
call_function() (data_specification.DataSpecificationGenerator method), 464

call_python() (pacman.executor.algorithm_classes.abstract_python_algorithm.AbstractPythonAlgorithm method), 81
call_python() (pacman.executor.algorithm_classes.AbstractPythonAlgorithm method), 85
call_python() (pacman.executor.algorithm_classes.python_classes.PythonClassAlgorithm method), 83
call_python() (pacman.executor.algorithm_classes.python_functions.PythonFunctionAlgorithm method), 83
call_python() (pacman.executor.algorithm_classes.PythonClassAlgorithm method), 86
call_python() (pacman.executor.algorithm_classes.PythonFunctionAlgorithm method), 86
call_random_distribution() (data_specification.data_specification_generator.DataSpecificationGenerator method), 427
call_random_distribution() (data_specification.DataSpecificationGenerator method), 465
CALLBACK_QUEUE_OVERLOADED (spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFr attribute), 544
CALLBACK_QUEUE_OVERLOADED (spinn_front_end_common.interface.provenance.ProvidesProvenanceDataFromMachineImpl.PROVENANCE_DATA_ENTR attribute), 545
CamelCaseConfigParser (class in spinn_utilities.configs.camel_case_config_parser), 3
can_record() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 814
CAN_STATUS (spinnman.messages.scp.enums.bmp_info.BMPInfo attribute), 274
CAN_STATUS (spinnman.messages.scp.enums.BMPInfo attribute), 278
CaseSensitiveParser (class in spinn_utilities.configs.case_sensitive_parser), 4
cdf() (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats method), 827
cdf() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
channel() (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead method), 250
channel() (spinnman.messages.eieio.command_messages.HostDataRead method), 255
channel() (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData method), 252
channel() (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData method), 256
channel_buffer_state() (spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state.EndBufferingState method), 516
channel_buffer_state() (spinn_front_end_common.interface.buffer_management.storage_objects.EndBufferingState method), 524
ChannelBufferState (class in spinn_front_end_common.interface.buffer_management.storage_objects), 522
ChannelBufferState (class in spinn_front_end_common.interface.buffer_management.storage_ 515
ChannelBufferState (class in spinn_front_end_common.interface.buffer_manag attribute), 515
ChannelBufferState (class in spinn_front_end_common.interface.buffer_manag attribute), 523
check_algorithm (pacman.operations.rigged_algorithms.isomorph_check.Isomorphic method), 189
check_algorithm_can_support_constraints() (in module pacman.utilities.utility_calls), 208
check_config() (in module spinn_utilities.conf_loader), 7
check_constraint_value() (in module pacman.utilities.utility_calls), 208
check_constraints() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker static method), 201
check_constraints() (pacman.utilities.utility_objs.resource_tracker.ProvidesProvenanceDataFr static method), 205
check_directory_exists_and_create_if_not() (in module spynnaker.pyNN.utilities.utility_calls), 831
check_for_error() (spinnman.processes.abstract_process.AbstractProcess method), 362
check_gsyn() (in module spynnaker.gsyn_tools), 837
check_gsyn() (in module spynnaker.gsyn_tools), 837
check_register_gysn() (in module spynnaker.gsyn_tools), 837
check_types_of_edge_constraint() (in module pacman.utilities.algorithm_utilities.routing_info_allocator_utilities), 196
CheckOKResponse (class in spinnman.messages.scp.impl), 297
CheckOKResponse (class in spinnman.messages.scp.impl.check_ok_response), 282
Chip (class in spinn_machine), 40
Chip (class in spinn_machine.chip), 22
chip (pacman.model.resources.core_resource.CoreResource attribute), 152
chip (pacman.model.resources.CoreResource attribute), 163
chip (pacman.model.resources.specific_chip_sdram_resource.SpecificChipS attribute), 159
chip (pacman.model.resources.specific_core_resource.SpecificCoreResourc attribute), 159
chip (pacman.model.resources.SpecificChipSDRAMResource attribute), 163
chip (pacman.model.resources.SpecificCoreResource attribute), 164
chip (pacman.model.resources.spinn_machine.Machine attribute), 47

chip_coordinates (spinn_machine.machine.Machine attribute), 30
chip_coordinates (spinn_machine.virtual_machine.VirtualMachine attribute), 39
chip_coordinates (spinn_machine.VirtualMachine attribute), 56
chip_info (spinnman.messages.scp.impl.get_chip_info_response.GetChipInfoResponse attribute), 286
chip_x (spinnman.connections.abstract_classes.scp_sender.SCPSender attribute), 222
chip_x (spinnman.connections.abstract_classes.SCPSender attribute), 227
chip_x (spinnman.connections.socket_address_with_chip.SocketAddressWithChip attribute), 244
chip_x (spinnman.connections.SocketAddressWithChip attribute), 247
chip_x (spinnman.connections.udp_packet_connections.bmp_connection.BMPConnection attribute), 230
chip_x (spinnman.connections.udp_packet_connections.BMPConnection attribute), 237
chip_x (spinnman.connections.udp_packet_connections.scamp_connection.SCAMPConnection attribute), 233
chip_x (spinnman.connections.udp_packet_connections.SCAMPConnection attribute), 241
chip_y (spinnman.connections.abstract_classes.scp_sender.SCPSender attribute), 222
chip_y (spinnman.connections.abstract_classes.SCPSender attribute), 227
chip_y (spinnman.connections.socket_address_with_chip.SocketAddressWithChip attribute), 244
chip_y (spinnman.connections.SocketAddressWithChip attribute), 247
chip_y (spinnman.connections.udp_packet_connections.bmp_connection.BMPConnection attribute), 230
chip_y (spinnman.connections.udp_packet_connections.BMPConnection attribute), 237
chip_y (spinnman.connections.udp_packet_connections.scamp_connection.SCAMPConnection attribute), 233
chip_y (spinnman.connections.udp_packet_connections.SCAMPConnection attribute), 241
ChipAndCoreConstraint (class in pacman.model.constraints.placer_constraints), 105
ChipAndCoreConstraint (class in pacman.model.constraints.placer_constraints.chip_and_core_constraint), 104
ChipInfo (class in spinnman.model), 348
ChipInfo (class in spinnman.model.chip_info), 333
ChipIOBufClearer (class in spinn_front_end_common.interface.interface_functions), 537
ChipIOBufClearer (class in spinn_front_end_common.interface.interface_functions.chip_iobuf_clearer), 530
ChipIOBufExtractor (class in spinn_front_end_common.interface.interface_functions), 537
ChipIOBufExtractor (class in spinn_front_end_common.interface.interface_functions.chip_iobu 531
ChipPowerMonitorApplicationVertex (class in spinn_front_end_common.utility_models.chip_power_monitor_a 584
ChipPowerMonitorMachineVertex (class in spinn_front_end_common.utility_models.chip_power_monitor_m 585
ChipPowerMonitorMachineVertex.CHIP_POWER_MONITOR_REGIONS (class in spinn_front_end_common.utility_models.chip_power_m 586
ChipProvenanceUpdater (class in spinn_front_end_common.interface.interface_functions), 537
ChipProvenanceUpdater (class in spinn_front_end_common.interface.interface_functions.chip_prov 531
ChipRuntimeUpdater (class in spinn_front_end_common.interface.interface_functions), 537
ChipRuntimeUpdater (class in spinn_front_end_common.interface.interface_functions.chip_runt 531
chips (spinn_machine.Machine attribute), 48
chips (spinn_machine.machine.Machine attribute), 30
chips (spinn_machine.virtual_machine.VirtualMachine attribute), 39
chips (spinn_machine.VirtualMachine attribute), 56
chips_available (pacman.utilities.utility_objs.resource_tracker.ResourceTrac attribute), 201
chips_available (pacman.utilities.utility_objs.ResourceTracker attribute), 205
chips_used (pacman.utilities.utility_objs.resource_tracker.ResourceTracker attribute), 201
chips_used (pacman.utilities.utility_objs.ResourceTracker attribute), 205
ChipSummaryInfo (class in spinnman.model), 350
ChipSummaryInfo (class in spinnman.model.chip_summary_info), 334
clear() (spinn_front_end_common.interface.buffer_management.storage_ob method), 508
clear() (spinn_front_end_common.interface.buffer_management.storage_ob method), 513
clear() (spinn_front_end_common.interface.buffer_management.storage_ob method), 516
clear() (spinn_front_end_common.interface.buffer_management.storage_ob method), 521
clear() (spinn_front_end_common.interface.provenance.pacman_provenanc method), 544
clear() (spinn_front_end_common.interface.provenance.PacmanProvenance

method), 545
clear_connection_cache() (spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses method), 665
clear_connection_cache() (spynnaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses method), 667
clear_connection_cache() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 779
clear_connection_cache() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 787
clear_connection_cache() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
clear_connection_cache() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
clear_injectables() (in module pacman.executor.injection_decorator), 94
clear_iobuf() (spinn_front_end_common.utilities.scp.clear_iobuf_process.ClearIOBUFProcess method), 565
clear_iobuf() (spinn_front_end_common.utilities.scp.ClearIOBUFProcess method), 566
clear_ip_tag() (spinnman.transceiver.Transceiver method), 384
clear_multicast_routes() (spinnman.transceiver.Transceiver method), 385
clear_recorded_data() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 524
clear_recorded_data() (spinn_front_end_common.interface.buffer_management.BufferManager method), 529
clear_recording() (spynnaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable method), 669
clear_recording() (spynnaker.pyNN.models.common.AbstractNeuronRecordable method), 673
clear_recording() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 779
clear_recording() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 787
clear_router_diagnostic_counters() (spinnman.transceiver.Transceiver method), 385
clear_spike_recording() (spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable method), 670
clear_spike_recording() (spynnaker.pyNN.models.common.AbstractSpikeRecordable method), 674
clear_spike_recording() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 779
clear_spike_recording() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 787
clear_spike_recording() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray method), 794
clear_spike_recording() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 796
clear_spike_recording() (spynnaker.pyNN.models.spike_source.SpikeSourceArray method), 800
clear_spike_recording() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 802
clear_spike_recording() (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector method), 809
clear_spike_recording() (spynnaker.pyNN.models.utility_models.SpikeInjector method), 812
ClearIOBUFProcess (class in spinn_front_end_common.utilities.scp), 566
ClearIOBUFProcess (class in spinn_front_end_common.utilities.scp.clear_iobuf_process), 565
clock_divisor (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 318
clock_divisor (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
clock_milliseconds (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 318
clock_milliseconds (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
CLOCK_SPEED (spinn_machine.Processor attribute), 52
clock_speed (spinn_machine.Processor attribute), 52
CLOCK_SPEED (spinn_machine.processor.Processor attribute), 35
clock_speed (spinn_machine.processor.Processor attribute), 35
clock_speed (spinn_machine.Router attribute), 53
clock_speed (spinn_machine.router.Router attribute), 36
clone_as_system_processor() (spinn_machine.Processor method), 52
clone_as_system_processor() (spinn_machine.processor.Processor method), 35
close() (spinn_front_end_common.abstract_models.abstract_machine_allocatable method), 496
close() (spinn_front_end_common.abstract_models.AbstractMachineAllocatable

method), 499
close() (spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection method), 552
close() (spinn_front_end_common.utilities.connections.LiveEventConnection method), 554
close() (spinn_front_end_common.utilities.database.database_connection.DatabaseConnection method), 555
close() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 555
close() (spinn_front_end_common.utilities.database.DatabaseConnection method), 559
close() (spinn_front_end_common.utilities.database.DatabaseReader method), 559
close() (spinn_front_end_common.utilities.notification_protocol.notification_protocol.NotificationProtocol method), 562
close() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 563
close() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage method), 59
close() (spinn_storage_handlers.abstract_classes.abstract_context_manager.AbstractContextManager method), 63
close() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage method), 65
close() (spinn_storage_handlers.abstract_classes.AbstractContextManager method), 68
close() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedBytearrayDataStorage method), 69
close() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage method), 70
close() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage method), 70
close() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 72
close() (spinn_storage_handlers.BufferedFileDataStorage method), 73
close() (spinn_storage_handlers.BufferedTempfileDataStorage method), 73
close() (spinn_storage_handlers.file_data_reader.FileDataReader method), 71
close() (spinn_storage_handlers.file_data_writer.FileDataWriter method), 72
close() (spinn_storage_handlers.FileDataReader method), 74
close() (spinn_storage_handlers.FileDataWriter method), 74
close() (spinnman.connections.abstract_classes.Connection method), 225
close() (spinnman.connections.abstract_classes.connection.Connection method), 219
close() (spinnman.connections.connection_listener.ConnectionListener method), 243
close() (spinnman.connections.ConnectionListener method), 245
close() (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection method), 234
close() (spinnman.connections.udp_packet_connections.UDPConnection method), 239
close() (spinnman.transceiver.Transceiver method), 386
close() (spinnman.utilities.io.abstract_io.AbstractIO method), 372
close() (spinnman.utilities.io.AbstractIO method), 375
close() (spinnman.utilities.io.file_io.FileIO method), 373
close() (spinnman.utilities.io.FileIO method), 376
close() (spinnman.utilities.io.memory_io.MemoryIO method), 375
close() (spinnman.utilities.io.MemoryIO method), 377
close() (spynnaker.pyNN.connections.ethernet_control_connection.EthernetControlConnection method), 619
close() (spynnaker.pyNN.connections.EthernetControlConnection method), 620
close() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_protocol.AbstractBufferedDataStorage method), 631
close() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_protocol.AbstractContextManager method), 638
closed (spinnman.utilities.io.abstract_io.AbstractIO attribute), 372
closed (spinnman.utilities.io.AbstractIO attribute), 375
closed (spinnman.utilities.io.file_io.FileIO attribute), 373
closed (spinnman.utilities.io.FileIO attribute), 376
closed (spinnman.utilities.io.memory_io.MemoryIO attribute), 375
closed (spinnman.utilities.io.MemoryIO attribute), 377
CLR (spinnman.messages.scp.enums.iptag_command.IPTagCommand attribute), 279
CLR (spinnman.messages.scp.enums.IPTagCommand attribute), 279
cm (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAndFire attribute), 736
cm (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire attribute), 741
CMD_ALLOC (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_ALLOC (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_APLX (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_APLX (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_AR (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_AR (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_AS (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_AS (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_BMP_INFO (spinnman.messages.scp.enums.scp_command.SCPCommand

attribute), 275
CMD_BMP_INFO (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_BMP_POWER (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_BMP_POWER (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_DPRI (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_DPRI (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_FFD (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_FFD (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_FILL (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_FILL (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_FLASH_COPY (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_FLASH_COPY (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_FLASH_ERASE (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_FLASH_ERASE (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_FLASH_WRITE (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_FLASH_WRITE (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_INFO (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_INFO (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_IPTAG (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_IPTAG (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_LED (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_LED (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_LINK_READ (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_LINK_READ (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_LINK_WRITE (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_LINK_WRITE (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_NNP (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 275
CMD_NNP (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_P2PC (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_P2PC (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_READ (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_READ (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_REMAP (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_REMAP (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_RESET (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_RESET (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_RTR (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_RTR (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_RUN (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_RUN (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_SIG (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_SIG (spinnman.messages.scp.enums.SCPCommand attribute), 278
CMD_SROM (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_SROM (spinnman.messages.scp.enums.SCPCommand attribute), 279
CMD_TUBE (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_TUBE (spinnman.messages.scp.enums.SCPCommand attribute), 279
CMD_VER (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276

CMD_VER (spinnman.messages.scp.enums.SCPCommand attribute), 279
CMD_WRITE (spinnman.messages.scp.enums.scp_command.SCPCommand attribute), 276
CMD_WRITE (spinnman.messages.scp.enums.SCPCommand attribute), 279
command (spinnman.exceptions.SpinnmanUnexpectedResponseCodeException attribute), 383
command (spinnman.messages.eieio.command_messages.eieio_command_header.EIEIOCommandHeader attribute), 249
command (spinnman.messages.eieio.command_messages.EIEIOCommandHeader attribute), 253
command (spinnman.messages.scp.scp_request_header.SCPRequestHeader attribute), 307
command (spinnman.messages.scp.SCPRequestHeader attribute), 309
Commands (class in data_specification.enums), 417
Commands (class in data_specification.enums.commands), 414
COMMANDS_AT_START_RESUME (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 589
COMMANDS_AT_START_RESUME (spinn_front_end_common.utility_models.CommandSenderMachineVertex attribute), 605
COMMANDS_AT_STOP_PAUSE (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 589
COMMANDS_AT_STOP_PAUSE (spinn_front_end_common.utility_models.CommandSenderMachineVertex attribute), 605
COMMANDS_WITH_ARBITRARY_TIMES (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 589
COMMANDS_WITH_ARBITRARY_TIMES (spinn_front_end_common.utility_models.CommandSenderMachineVertex attribute), 605
CommandSender (class in spinn_front_end_common.utility_models), 603
CommandSender (class in spinn_front_end_common.utility_models.command_sender), 587
CommandSenderMachineVertex (class in spinn_front_end_common.utility_models), 605
CommandSenderMachineVertex (class in spinn_front_end_common.utility_models.command_sender_machine_vertex), 589
CommandSenderMachineVertex.DATA_REGIONS (class in spinn_front_end_common.utility_models), 605
CommandSenderMachineVertex.DATA_REGIONS (class in spinn_front_end_common.utility_models.command_sender_machine_vertex), 589
comment() (data_specification.data_specification_generator.DataSpecificati method), 428
comment() (data_specification.DataSpecificationGenerator method), 465
compress_bits_from_bit_array() (in module pacman.utilities.utility_calls), 208
compress_from_bit_array() (in module pacman.utilities.utility_calls), 209
CompressibleMallocBasedRoutingInfoAllocator (class in pacman.operations.routing_info_allocator_algorithms.malloc_based_ 192
Condition (class in data_specification.enums), 419
Condition (class in data_specification.enums.condition), 416
conditionals (data_specification.data_specification_generator.DataSpecificat attribute), 428
conditionals (data_specification.DataSpecificationGenerator attribute), 466
config (spinn_front_end_common.abstract_vertex.CommandSenderMachineVertex.DATA_REGIO attribute), 547
config (spinn_front_end_common.utilities.failed_state.FailedState attribute), 578
config (spinn_front_end_common.utilities.FailedState attribute), 583
config (spinn_front_end_common.utilities.simulator_interface.SimulatorInte attribute), 582
config (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
CONFIG (spinn_front_end_common.utility_models.chip_power_monitor_m attribute), 586
CONFIG_FILE_NAME (spinnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCom attribute), 833
CONFIG_SIZE_IN_BYTES (spinn_front_end_common.utility_models.chip_power_monitor_m attribute), 586
ConfigurationException, 578
configure_master_key() (spinnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic method), 818
configure_master_key() (spinnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
configure_master_key_key (spinnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 819
configure_master_key_key (spinnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824

ConfiguredFilter (class in spinn_utilities.log), 10
ConfiguredFormatter (class in spinn_utilities.log), 10
conn_list (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector attribute), 683
conn_list (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector attribute), 692
CONN_LIST_DTYPE (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector attribute), 683
CONN_LIST_DTYPE (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector attribute), 692
connected_chip_x (spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData attribute), 16
connected_chip_x (spinn_machine.link_data_objects.AbstractLinkData attribute), 17
connected_chip_y (spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData attribute), 16
connected_chip_y (spinn_machine.link_data_objects.AbstractLinkData attribute), 17
connected_link (spinn_machine.link_data_objects.abstract_link_data.AbstractLinkData attribute), 16
connected_link (spinn_machine.link_data_objects.AbstractLinkData attribute), 17
Connection (class in spinnman.connections.abstract_classes), 225
Connection (class in spinnman.connections.abstract_classes.connection), 219
ConnectionHolder (class in spynnaker.pyNN.models.neuron), 790
ConnectionHolder (class in spynnaker.pyNN.models.neuron.connection_holder), 783
ConnectionListener (class in spinnman.connections), 245
ConnectionListener (class in spinnman.connections.connection_listener), 243
connections (spynnaker.pyNN.models.neuron.connection_holder.ConnectionHolder attribute), 783
connections (spynnaker.pyNN.models.neuron.ConnectionHolder attribute), 791
ConnectiveBasedPlacer (class in pacman.operations.placer_algorithms), 188
ConnectiveBasedPlacer (class in pacman.operations.placer_algorithms.connective_based_placer), 187
connector (spynnaker.pyNN.models.neural_projections.synapse_information.SynapseInformation attribute), 698
connector (spynnaker.pyNN.models.neural_projections.SynapseInformation attribute), 701
ConstrainedObject (class in pacman.model.graphs.common), 121
ConstrainedObject (class in pacman.model.graphs.common.constrained_object), 118
constraint_class (pacman.utilities.ConstraintOrder attribute), 210
constraint_class (pacman.utilities.vertex_sorter.ConstraintOrder attribute), 210
ConstraintOrder (class in pacman.utilities), 210
ConstraintOrder (class in pacman.utilities.vertex_sorter), 210
constraints (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 138
constraints (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition attribute), 140
constraints (pacman.model.graphs.abstract_vertex.AbstractVertex attribute), 141
constraints (pacman.model.graphs.AbstractGraph attribute), 144
constraints (pacman.model.graphs.AbstractOutgoingEdgePartition attribute), 146
constraints (pacman.model.graphs.AbstractVertex attribute), 147
constraints (pacman.model.graphs.common.constrained_object.ConstrainedObject attribute), 119
constraints (pacman.model.graphs.common.ConstrainedObject attribute), 121
CONSTRUCT (data_specification.enums.Commands attribute), 418
CONSTRUCT (data_specification.enums.commands.Commands attribute), 414
construct_logging_parents() (spinn_utilities.log.ConfiguredFormatter static method), 10
consume() (spinnman.connections.token_bucket.TokenBucket method), 245
consume() (spinnman.connections.TokenBucket method), 247
ContiguousKeyRangeContraint (class in pacman.model.constraints.key_allocator_constraints), 100
ContiguousKeyRangeContraint (class in pacman.model.constraints.key_allocator_constraints.contiguous_key_99
CONTINUE (spinnman.messages.scp.enums.Signal attribute), 280
CONTINUE (spinnman.messages.scp.enums.signal.Signal attribute), 277
convert_from_rig_placements() (in module pacman.utilities.rig_converters), 208
convert_from_rig_routes() (in module pacman.utilities.rig_converters), 208
convert_mask_into_fields() (in module pacman.utilities.algorithm_utilities.field_based_system_utilities), 195
convert_param_to_numpy() (in module spynnaker.pyNN.utilities.utility_calls), 830

convert_per_connection_data_to_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
convert_per_connection_data_to_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 760
convert_routing_table_entry_to_spinnaker_route() (spinn_machine.Router static method), 53
convert_routing_table_entry_to_spinnaker_route() (spinn_machine.router.Router static method), 36
convert_string_into_chip_and_core_subset() (in module spinn_front_end_common.utilities.helpful_functions), 579
convert_time_diff_to_total_milliseconds() (in module spinn_front_end_common.utilities.helpful_functions), 579
convert_to_rig_graph() (in module pacman.utilities.rig_converters), 208
convert_to_rig_graph_pure_mc() (in module pacman.utilities.rig_converters), 208
convert_to_rig_machine() (in module pacman.utilities.rig_converters), 208
convert_to_rig_placements() (in module pacman.utilities.rig_converters), 208
ConvertToFileCoreAllocations (class in pacman.utilities.file_format_converters.convert_to_file_core_allocations), 197
ConvertToFileMachine (class in pacman.utilities.file_format_converters.convert_to_file_machine), 197
ConvertToFileMachineGraph (class in pacman.utilities.file_format_converters.convert_to_file_machine_graph), 197
ConvertToFileMachineGraphPureMulticast (class in pacman.utilities.file_format_converters.convert_to_file_machine_graph_pure_multicast), 198
ConvertToFilePlacement (class in pacman.utilities.file_format_converters.convert_to_file_placement), 198
ConvertToMemoryMultiCastRoutes (class in pacman.utilities.file_format_converters.convert_to_memory_multicast_routes), 198
ConvertToMemoryPlacements (class in pacman.utilities.file_format_converters.convert_to_memory_placements), 198
COPY_PARAM (data_specification.enums.Commands attribute), 418
COPY_PARAM (data_specification.enums.commands.Commands attribute), 414
COPY_STRUCT (data_specification.enums.Commands attribute), 418
COPY_STRUCT (data_specification.enums.commands.Commands attribute), 415
copy_structure() (data_specification.data_specification_generator.DataSpecification method), 428
copy_structure() (data_specification.DataSpecificationGenerator method), 466
copy_structure_parameter() (data_specification.data_specification_generator.DataSpecification method), 429
copy_structure_parameter() (data_specification.DataSpecificationGenerator method), 466
core_resources (pacman.model.resources.pre_allocated_resource_container.PreAllocatedResourceContainer attribute), 155
core_resources (pacman.model.resources.PreAllocatedResourceContainer attribute), 163
core_states (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
core_states (spinnman.model.ChipSummaryInfo attribute), 350
core_subsets (spinn_machine.core_subsets.CoreSubsets attribute), 26
core_subsets (spinn_machine.CoreSubsets attribute), 44
CoreResource (class in pacman.model.resources), 162
CoreResource (class in pacman.model.resources.core_resource), 152
cores (pacman.model.resources.specific_core_resource.SpecificCoreResource attribute), 159
cores (pacman.model.resources.SpecificCoreResource attribute), 164
cores_and_link_output_string() (spinn_machine.Machine method), 48
cores_and_link_output_string() (spinn_machine.machine.Machine method), 30
CoreSubset (class in spinn_machine), 43
CoreSubset (class in spinn_machine.core_subset), 25
CoreSubsets (class in spinn_machine), 44
CoreSubsets (class in spinn_machine.core_subsets), 26
count (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader attribute), 258
count (spinnman.messages.eieio.data_messages.EIEIODataHeader attribute), 262
count (spinnman.messages.scp.impl.count_state_response.CountStateResponse attribute), 282
count (spinnman.messages.scp.impl.CountStateResponse attribute), 282
count_used_regions() (data_specification.memory_region_collection.MemoryRegionCollection method), 456
count_used_regions() (data_specification.MemoryRegionCollection method), 488
CountState (class in spinnman.messages.scp.impl), 297
CountState (class in spinnman.messages.scp.impl.count_state), 282
CountStateResponse (class in spinnman.messages.scp.impl.count_state_response), 282
cpu_clock_mhz (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBootValues

attribute), 318
cpu_clock_mhz (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
cpu_clock_mhz (spinnman.model.chip_info.ChipInfo attribute), 333
cpu_clock_mhz (spinnman.model.ChipInfo attribute), 348
cpu_cycles (pacman.model.resources.resource_container.ResourceContainer attribute), 155
cpu_cycles (pacman.model.resources.ResourceContainer attribute), 161
cpu_cycles_available (spinn_machine.Processor attribute), 52
cpu_cycles_available (spinn_machine.processor.Processor attribute), 35
cpu_information_base_address (spinnman.messages.spinnaker_boot.system_variable_boot.SystemVariableDefinition attribute), 318
cpu_information_base_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
cpu_information_base_address (spinnman.model.chip_info.ChipInfo attribute), 333
cpu_information_base_address (spinnman.model.ChipInfo attribute), 348
cpu_infos (spinnman.model.cpu_infos.CPUInfos attribute), 338
cpu_infos (spinnman.model.CPUInfos attribute), 354
CPU_STATE_12 (spinnman.model.enums.cpu_state.CPUState attribute), 325
CPU_STATE_12 (spinnman.model.enums.CPUState attribute), 328
CPU_STATE_13 (spinnman.model.enums.cpu_state.CPUState attribute), 325
CPU_STATE_13 (spinnman.model.enums.CPUState attribute), 328
CPU_STATE_14 (spinnman.model.enums.cpu_state.CPUState attribute), 325
CPU_STATE_14 (spinnman.model.enums.CPUState attribute), 328
CPUCyclesPerTickResource (class in pacman.model.resources), 159
CPUCyclesPerTickResource (class in pacman.model.resources.cpu_cycles_per_tick_resource), 153
CPUInfo (class in spinnman.model), 351
CPUInfo (class in spinnman.model.cpu_info), 336
CPUInfos (class in spinnman.model), 353
CPUInfos (class in spinnman.model.cpu_infos), 338
CPUState (class in spinnman.model.enums), 328
CPUState (class in spinnman.model.enums.cpu_state), 325
create() (spinnman.messages.eieio.data_messages.eieio_data_message.EIEI static method), 260
create() (spinnman.messages.eieio.data_messages.EIEIODataMessage static method), 264
create() (spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.Fak method), 829
create_atom_to_event_id_mapping() (spinn_front_end_common.utilities.database.database_writer.Data method), 558
create_atom_to_event_id_mapping() (spinn_front_end_common.utilities.database.DatabaseWriter method), 561
create_cmd() (data_specification.data_specification_generator.DataSpecifica method), 429
create_cmd() (data_specification.DataSpecificationGenerator method), 467
create_from_bytearray() (spinn_front_end_common.interface.buffer_manag static method), 515
create_from_bytearray() (spinn_front_end_common.interface.buffer_manag static method), 523
create_label() (spynnaker.pyNN.models.pynn_population_common.PyNNP static method), 814
create_machine_edge() (pacman.model.graphs.application.application_edge.ApplicationEdge method), 109
create_machine_edge() (pacman.model.graphs.application.ApplicationEdge method), 114
create_machine_edge() (spynnaker.pyNN.models.neural_projections.delay_afferent_applicatio method), 696
create_machine_edge() (spynnaker.pyNN.models.neural_projections.DelayAfferentApplicatio method), 699
create_machine_edge() (spynnaker.pyNN.models.neural_projections.delayed_application_edge method), 696
create_machine_edge() (spynnaker.pyNN.models.neural_projections.DelayedApplicationEdge method), 699
create_machine_edge() (spynnaker.pyNN.models.neural_projections.projection_application_ed method), 697
create_machine_edge() (spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdg method), 700
create_machine_vertex() (pacman.model.graphs.application.application_fpga_vertex.Applicati method), 110
create_machine_vertex() (pacman.model.graphs.application.application_spinnaker_link_vertex

method), 112
create_machine_vertex() (pacman.model.graphs.application.application_vertex.ApplicationVertex method), 113
create_machine_vertex() (pacman.model.graphs.application.ApplicationFPGAVertex method), 115
create_machine_vertex() (pacman.model.graphs.application.ApplicationSpiNNakerLinkVertex method), 117
create_machine_vertex() (pacman.model.graphs.application.ApplicationVertex method), 116
create_machine_vertex() (spinn_front_end_common.utility_models.ChipPowerMonitorApplicationVertex method), 585
create_machine_vertex() (spinn_front_end_common.utility_models.CommandSender method), 588
create_machine_vertex() (spinn_front_end_common.utility_models.ExtraMonitorSupportApplicationVertex method), 604
create_machine_vertex() (spinn_front_end_common.utility_models.LivePacketGather method), 590
create_machine_vertex() (spinn_front_end_common.utility_models.LivePacketGatherApplicationVertex method), 594
create_machine_vertex() (spinn_front_end_common.utility_models.ReverseIPTagMultiCastSource method), 606
create_machine_vertex() (spinn_front_end_common.utility_models.TagMultiCastSource method), 600
create_machine_vertex() (spyn-naker.pyNN.external_devices_models.munich_spinnaker_link_interface.MunichMotorDevice method), 611
create_machine_vertex() (spyn-naker.pyNN.external_devices_models.MunichMotorDevice method), 655
create_machine_vertex() (spyn-naker.pyNN.external_devices_models.MunichMotorDevice method), 663
create_machine_vertex() (spyn-naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 779
create_machine_vertex() (spyn-naker.pyNN.models.neuron.AbstractPopulationVertex method), 787
create_machine_vertex() (spyn-naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 796
create_machine_vertex() (spyn-naker.pyNN.models.spike_source.SpikeSourcePoisson method), 802
create_machine_vertex() (spyn-naker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 807
create_machine_vertex() (spyn-naker.pyNN.models.utility_models.DelayExtensionVertex method), 811
create_rig_graph_constraints() (in module pacman.utilities.rig_converters), 208
create_rig_machine_constraints() (in module pacman.utilities.rig_converters), 208
create_schema() (spinn_front_end_common.utilities.database.database_writer.DatabaseWriter method), 558
create_schema() (spinn_front_end_common.utilities.database.DatabaseWriter method), 562
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector method), 678
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 681
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector method), 691
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 691
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 683
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.FromListConnector method), 692
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector method), 684
create_synaptic_block() (spyn-naker.pyNN.models.neural_projections.connectors.MultapseConnector

method), 693
create_synaptic_block() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685
create_synaptic_block() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
create_synaptic_block() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector method), 686
create_synaptic_block() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
create_transceiver_from_hostname() (in module spinnman.transceiver), 409
create_virtual_chip() (in module pacman.utilities.algorithm_utilities.machine_algorithm_utilities), 196
CreateConstraintsToFile (class in pacman.utilities.file_format_converters.create_file_constraints), 198
current_read (spinn_front_end_common.interface.buffer_management.storage_objects.sdp_buffered_state.ChannelBufferState attribute), 515
current_read (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState attribute), 523
current_region (data_specification.data_specification_executor.DataSpecificationExecutor attribute), 422
current_region (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 430
current_region (data_specification.DataSpecificationExecutorFunctions attribute), 459
current_region (data_specification.DataSpecificationGenerator attribute), 468
CURRENT_TIMER_TIC (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex attribute), 784
CURRENT_TIMER_TIC (spynnaker.pyNN.models.neuron.PopulationMachineVertex attribute), 792
current_timestamp (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region.BufferedSendingRegion attribute), 513
current_timestamp (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region attribute), 521
current_write (spinn_front_end_common.interface.buffer_management.storage_objects.sdp_buffered_state.ChannelBufferState attribute), 515
current_write (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState attribute), 523
cursor (spinn_front_end_common.utilities.database.database_connection.DatabaseConnection attribute), 555
cursor (spinn_front_end_common.utilities.database.DatabaseReader attribute), 559

D

d (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh attribute), 735
d (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 740
d_expression (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector attribute), 680
d_expression (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector attribute), 689
DABT (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
DABT (spinnman.model.enums.RunTimeError attribute), 329
data (spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage attribute), 249
data (spinnman.messages.eieio.command_messages.EIEIOCommandMessage attribute), 254
data (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 272
data (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest attribute), 270
data (spinnman.messages.sdp.sdp_message.SDPMessage attribute), 312
data (spinnman.messages.sdp.sdp_message.SDPMessage attribute), 315
data (spinnman.messages.spinnaker_boot.spinnaker_boot_message.SpinnakerBootMessage attribute), 315
data (spinnman.messages.spinnaker_boot.SpinnakerBootMessage attribute), 320
data_time_stamp (spinn_front_end_common.interface.provenance.pacman_provenance.PacmanProvenanceFunctions attribute), 544
data_time_stamp (spinn_front_end_common.interface.provenance.PacmanProvenanceFunctions attribute), 545
data_len (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 422
data_len (data_specification.DataSpecificationExecutorFunctions attribute), 459
data_specification (module), 413, 458
data_specification.constants (module), 420
data_specification.data_specification_executor_functions (module), 420
data_specification.data_specification_generator_functions (module), 421
data_specification.data_specification_executor (module), 424
data_specification.enums.arithemetic_operation (module), 414
data_specification.enums.commands (module), 414
data_specification.enums.condition (module), 416
data_specification.enums.data_type (module), 416
data_specification.enums.logic_operation (module), 417
data_specification.enums.random_number_generator (module), 417
data_specification.exceptions (module), 450
data_specification.memory_region (module), 455

data_specification.memory_region_collection (module), 456
data_specification.utility_calls (module), 456
data_type (spinnman.messages.spinnaker_boot.system_variable_definition.SystemVariableDefinition attribute), 318
data_type (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
DATABASE_CONFIRMATION (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
database_file_path (spinn_front_end_common.interface.interface_functions.database_interface.DatabaseInterface attribute), 531
database_file_path (spinn_front_end_common.interface.interface_functions.DatabaseInterface attribute), 538
database_path (spinn_front_end_common.utilities.database.database_connection.DatabaseConnection attribute), 558
database_path (spinn_front_end_common.utilities.database.DatabaseWriter attribute), 562
database_path (spinnman.messages.eieio.command_messages.database_confirmation.DatabaseConfirmation attribute), 248
database_path (spinnman.messages.eieio.command_messages.DatabaseConfirmation attribute), 253
DatabaseConfirmation (class in spinnman.messages.eieio.command_messages), 253
DatabaseConfirmation (class in spinnman.messages.eieio.command_messages.database_confirmation), 248
DatabaseConnection (class in spinn_front_end_common.utilities.database), 558
DatabaseConnection (class in spinn_front_end_common.utilities.database.database_connection), 555
DatabaseInterface (class in spinn_front_end_common.interface.interface_functions), 537
DatabaseInterface (class in spinn_front_end_common.interface.interface_functions.database_interface), 531
DatabaseReader (class in spinn_front_end_common.utilities.database), 559
DatabaseReader (class in spinn_front_end_common.utilities.database.database_reader), 555
DatabaseWriter (class in spinn_front_end_common.utilities.database), 560
DatabaseWriter (class in spinn_front_end_common.utilities.database.database_writer), 557
DataReadException, 71
DataSpecificationDuplicateParameterException, 450
DataSpecificationException, 450
DataSpecificationExecutor (class in data_specification), 458
DataSpecificationExecutorFunctions (class in data_specification.data_specification_executor), 420
DataSpecificationExecutorFunctions (class in data_specification), 459
DataSpecificationExecutorFunctions (class in data_specification.data_specification_executor_functions), 421
DataSpecificationFunctionInUse, 450
DataSpecificationGenerator (class in data_specification), 462
DataSpecificationGenerator (class in data_specification.data_specification_generator), 424
DataSpecificationInvalidCommandException, 450
DataSpecificationInvalidDatabaseException, 450
DataSpecificationInvalidSizeException, 450
DataSpecificationMoreException, 451
DataSpecificationNoRegionSelectedException, 451
DataSpecificationNotAllocatedException, 451
DataSpecificationParameterOutOfBoundsException, 451
DataSpecificationRandomNumberDistributionInUseException, 452
DataSpecificationRegionExhaustedException, 452
DataSpecificationRegionInUseException, 452
DataSpecificationRegionNotAllocated, 452
DataSpecificationRegionOutOfBoundsException, 452
DataSpecificationRegionUnfilledException, 453
DataSpecificationRNGInUseException, 452
DataSpecificationStructureInUseException, 453
DataSpecificationSyntaxError, 453
DataSpecificationTablePointerOutOfMemory, 453
DataSpecificationTypeMismatchException, 453
DataSpecificationUnknownConditionException, 454
DataSpecificationUnknownTypeException, 454
DataSpecificationUnknownTypeLengthException, 454
DataSpecificationWrongParameterNumberException, 454
DataType (class in data_specification.enums), 419
DataType (class in data_specification.enums.data_type), 416
DataUndefinedWriterException, 454
DataWriteException, 71
de_alloc_sdram() (spinnman.processes.de_alloc_sdram_process.DeAllocSDRAMProcess method), 362
de_alloc_sdram() (spinnman.processes.DeAllocSDRAMProcess method), 368
DEAD (spinnman.model.enums.cpu_state.CPUState attribute), 325

DEAD (spinnman.model.enums.CPUState attribute), 328
DeAllocSDRAMProcess (class in spinnman.processes), 368
DeAllocSDRAMProcess (class in spinnman.processes.de_alloc_sdram_process), 362
debug_x (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
debug_x (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
debug_y (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
debug_y (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
DECLARE_RANDOM_DIST (data_specification.enums.Commands attribute), 418
DECLARE_RANDOM_DIST (data_specification.enums.commands.Commands attribute), 415
declare_random_number_generator() (data_specification.data_specification_generator.DataSpecificationGenerator method), 430
declare_random_number_generator() (data_specification.DataSpecificationGenerator method), 468
DECLARE_RNG (data_specification.enums.Commands attribute), 418
DECLARE_RNG (data_specification.enums.commands.Commands attribute), 415
declare_uniform_random_distribution() (data_specification.data_specification_generator.DataSpecificationGenerator method), 431
declare_uniform_random_distribution() (data_specification.DataSpecificationGenerator method), 468
decode_algorithm_data_objects() (pacman.executor.algorithm_metadata_xml_reader.AlgorithmMetadataXmlReader method), 94
decode_algorithm_data_objects() (pacman.executor.AlgorithmMetadataXmlReader method), 97
deduce_types() (in module pacman.utilities.algorithm_utilities.field_based_system_utilities), 195
deepest_parent() (spinn_utilities.log.ConfiguredFormatter static method), 10
default (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
default (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
DEFAULT1 (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray attribute), 794
DEFAULT1 (spynnaker.pyNN.models.spike_source.SpikeSourceArray attribute), 800
DEFAULT_DEST_X_COORD (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 272
DEFAULT_DEST_X_COORD (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRe attribute), 270
DEFAULT_DEST_Y_COORD (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 272
DEFAULT_DEST_Y_COORD (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRe attribute), 270
DEFAULT_MALLOCS_USED (spinn_front_end_common.utility_models.chip_power_monitor_r attribute), 586
DEFAULT_MAX_ATOMS_PER_CORE (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSou attribute), 796
DEFAULT_MAX_ATOMS_PER_CORE (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 802
default_parameters (spynnaker.pyNN.external_devices_models.arbitrary_fpga_device.Arbi attribute), 652
default_parameters (spynnaker.pyNN.external_devices_models.ArbitraryFPGADevice attribute), 659
default_parameters (spynnaker.pyNN.external_devices_models.external_device_lif_control attribute), 653
default_parameters (spynnaker.pyNN.external_devices_models.external_spinnaker_link_co attribute), 653
default_parameters (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fp attribute), 654
default_parameters (spynnaker.pyNN.external_devices_models.ExternalCochleaDevice attribute), 661
default_parameters (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl attribute), 660
default_parameters (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevic attribute), 662
default_parameters (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_mo attribute), 655
default_parameters (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_ret attribute), 657
default_parameters (spynnaker.pyNN.external_devices_models.MunichMotorDevice

attribute), 663
default_parameters (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 665
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 642
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 643
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 644
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 644
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 645
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 646
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 647
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 647
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 648
default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol attribute), 648
default_parameters (spynnaker.pyNN.models.neuron.builds.if_cond_exp_base.IFCondExpBase attribute), 708
default_parameters (spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc.IFCondExpStoc attribute), 709
default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_delta.IFCurrDelta attribute), 711
default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base.IFCurrDualExpBase attribute), 712
default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.IFCurrExpBase attribute), 713
default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive.IFCurrExpCa2Adaptive attribute), 714
default_parameters (spynnaker.pyNN.models.neuron.builds.IFCondExpBase attribute), 719
default_parameters (spynnaker.pyNN.models.neuron.builds.IFCondExpStoc attribute), 722
default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrDelta attribute), 723
default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrDualExpBase attribute), 720
default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrExpBase attribute), 720
default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrExpCa2Adaptive attribute), 723
default_parameters (spynnaker.pyNN.models.neuron.builds.izk_cond_exp_basic.IzkCondExpBasic attribute), 716
default_parameters (spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base.IzkCurrExpBase attribute), 717
default_parameters (spynnaker.pyNN.models.neuron.builds.IzkCondExpBase attribute), 721
default_parameters (spynnaker.pyNN.models.neuron.builds.IzkCurrExpBase attribute), 722
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceRetinaDevice attribute), 746
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpeakerDevice attribute), 747
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceExponential attribute), 748
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceLookupExponential attribute), 750
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceLinear attribute), 750
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceNearestPairExponential attribute), 751
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceExponential attribute), 753
default_parameters (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceCa2Adaptive attribute), 755
default_parameters (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray

attribute), 794
default_parameters (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson attribute), 796
default_parameters (spynnaker.pyNN.models.spike_source.SpikeSourceArray attribute), 800
default_parameters (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 802
default_parameters (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector attribute), 809
default_parameters (spynnaker.pyNN.models.utility_models.SpikeInjector attribute), 813
DEFAULT_ROUTED (spinnman.model.enums.diagnostic_filter_default_routing_status.DiagnosticFilterDefaultRoutingStatus attribute), 325
DEFAULT_ROUTED (spinnman.model.enums.DiagnosticFilterDefaultRoutingStatus attribute), 329
default_routing_statuses (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
default_routing_statuses (spinnman.model.DiagnosticFilter attribute), 355
DEFAULT_SDRAM_BYTES (spinn_machine.SDRAM attribute), 54
DEFAULT_SDRAM_BYTES (spinn_machine.sdram.SDRAM attribute), 37
defaultable (pacman.model.routing_table_by_partition.multicast_routing_table_by_partition_entry.MulticastRoutingTableByPartitionEntry attribute), 171
defaultable (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartitionEntry attribute), 173
defaultable (pacman.operations.router_compressors.mundys_router_compressor.routing_table_condenser.MundyRouterCompressor.RoutingEntry attribute), 191
defaultable (spinn_machine.multicast_routing_entry.MulticastRoutingEntry attribute), 33
defaultable (spinn_machine.MulticastRoutingEntry attribute), 51
define_break() (data_specification.data_specification_generator.DataSpecificationGenerator method), 431
define_break() (data_specification.DataSpecificationGenerator method), 469
define_structure() (data_specification.data_specification_generator.DataSpecificationGenerator method), 432
define_structure() (data_specification.DataSpecificationGenerator method), 469
delay_between_repeats (spinn_front_end_common.utility_models.MultiCastCommand attribute), 598
delay_between_repeats (spinn_front_end_common.utility_models.MultiCastCommand attribute), 609
delay_block (spynnaker.pyNN.models.utility_models.delay_block.DelayBlock attribute), 806
delay_block (spynnaker.pyNN.models.utility_models.DelayBlock attribute), 810
delay_edge (spynnaker.pyNN.models.neural_projections.projection_applicationEdge attribute), 697
delay_edge (spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge attribute), 700
DelayAfferentApplicationEdge (class in spynnaker.pyNN.models.neural_projections), 699
DelayAfferentApplicationEdge (class in spynnaker.pyNN.models.neural_projections.delay_afferent_application_edge), 696
DelayAfferentMachineEdge (class in spynnaker.pyNN.models.neural_projections), 699
DelayAfferentMachineEdge (class in spynnaker.pyNN.models.neural_projections.delay_afferent_machine_edge), 696
DelayBlock (class in spynnaker.pyNN.models.utility_models), 810
DelayBlock (class in spynnaker.pyNN.models.utility_models.delay_block), 806
DelayedApplicationEdge (class in spynnaker.pyNN.models.neural_projections), 699
DelayedApplicationEdge (class in spynnaker.pyNN.models.neural_projections.delayed_application_edge), 696
DelayedMachineEdge (class in spynnaker.pyNN.models.neural_projections), 699
DelayedMachineEdge (class in spynnaker.pyNN.models.neural_projections.delayed_machine_edge), 697
DelayExtensionException, 834
DelayExtensionMachineVertex (class in spynnaker.pyNN.models.utility_models), 810
DelayExtensionMachineVertex (class in spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex), 806
DelayExtensionMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES (class in spynnaker.pyNN.models.utility_models), 810
DelayExtensionMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES (class in spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex), 806
DelayExtensionVertex (class in spynnaker.pyNN.models.utility_models), 810
DelayExtensionVertex (class in spynnaker.pyNN.models.utility_models.delay_extension_vertex),

807
DELTA_TIMESTAMPS (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichRetinaPayload attribute), 822
DELTA_TIMESTAMPS (spynnaker.pyNN.protocols.RetinaPayload attribute), 827
dendritic_delay_fraction (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP attribute), 759
dendritic_delay_fraction (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP attribute), 763
dependent_vertices() (spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.AbstractVertexWithEdgeTo method), 499
dependent_vertices() (spinn_front_end_common.abstract_models.AbstractVertexWithEdgeToDependentVertices method), 501
dependent_vertices() (spinn_front_end_common.utility_models.command_sender.CommandSender method), 588
dependent_vertices() (spinn_front_end_common.utility_models.CommandSender method), 604
dependent_vertices() (spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex.ExternalDeviceLifControl method), 653
dependent_vertices() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl method), 660
dependent_vertices() (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_interface.MunichMotorDevice method), 656
dependent_vertices() (spynnaker.pyNN.external_devices_models.MunichMotorDevice method), 663
describe() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 779
describe() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 787
describe() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray method), 794
describe() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 796
describe() (spynnaker.pyNN.models.spike_source.SpikeSourceArray method), 800
describe() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 802
describe() (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector method), 809
describe() (spynnaker.pyNN.models.utility_models.SpikeInjector method), 813
dest_reg (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 422
dest_reg (data_specification.DataSpecificationExecutorFunctions attribute), 459
destination_chip_x (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 310
destination_chip_x (spinnman.messages.sdp.SDPHeader attribute), 313
destination_chip_y (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 310
destination_chip_y (spinnman.messages.sdp.SDPHeader attribute), 313
destination_cpu (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
destination_cpu (spinnman.messages.sdp.SDPHeader attribute), 313
destination_p (spinn_machine.tags.reverse_iptag.ReverseIPTag attribute), 19
destination_p (spinn_machine.tags.ReverseIPTag attribute), 21
destination_port (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
destination_port (spinnman.messages.sdp.SDPHeader attribute), 313
destination_x (spinn_machine.Link attribute), 46
destination_x (spinn_machine.link.Link attribute), 28
destination_x (spinn_machine.tags.IPTag attribute), 20
destination_x (spinn_machine.tags.iptag.IPTag attribute), 19
destination_x (spinn_machine.tags.reverse_iptag.ReverseIPTag attribute), 19
destination_x (spinn_machine.tags.ReverseIPTag attribute), 21
destination_y (spinn_machine.Link attribute), 46
destination_y (spinn_machine.link.Link attribute), 28
destination_y (spinn_machine.tags.IPTag attribute), 20
destination_y (spinn_machine.tags.iptag.IPTag attribute), 19
destination_y (spinn_machine.tags.reverse_iptag.ReverseIPTag attribute), 19
destination_y (spinn_machine.tags.ReverseIPTag attribute), 21
DestinationBasedRoutingInfoAllocator (class in pacman.operations.routing_info_allocator_algorithms.destination_ba 193
destinations (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
destinations (spinnman.model.DiagnosticFilter attribute), 355
destroy() (spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.Fa method), 839
determine_flow_states() (in module spinn_front_end_common.utilities.helpful_functions), 579
device_control_key (spyn-

naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice attribute), 651
device_control_key (spyn-naker.pyNN.external_devices_models.AbstractMulticastControllableDevice attribute), 659
device_control_key (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 625
device_control_key (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 633
device_control_max_value (spyn-naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice attribute), 651
device_control_max_value (spyn-naker.pyNN.external_devices_models.AbstractMulticastControllableDevice attribute), 659
device_control_max_value (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 626
device_control_max_value (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 633
device_control_min_value (spyn-naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice attribute), 651
device_control_min_value (spyn-naker.pyNN.external_devices_models.AbstractMulticastControllableDevice attribute), 659
device_control_min_value (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 626
device_control_min_value (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 633
device_control_partition_id (spyn-naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice attribute), 651
device_control_partition_id (spyn-naker.pyNN.external_devices_models.AbstractMulticastControllableDevice attribute), 659
device_control_partition_id (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 626
device_control_partition_id (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 633
device_control_timesteps_between_sending (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 626
device_control_timesteps_between_sending (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 633
device_control_timesteps_between_sending() (spyn-naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice method), 651
device_control_timesteps_between_sending() (spyn-naker.pyNN.external_devices_models.AbstractMulticastControllableDevice method), 659
device_control_uses_payload (spyn-naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice attribute), 651
device_control_uses_payload (spyn-naker.pyNN.external_devices_models.AbstractMulticastControllableDevice attribute), 659
device_control_uses_payload (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 626
device_control_uses_payload (spyn-naker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 633
DiagnosticFilter (class in spinnman.model), 354
DiagnosticFilter (class in spinnman.model.diagnostic_filter), 339
DiagnosticFilterDefaultRoutingStatus (class in spinnman.model.enums), 329
DiagnosticFilterDefaultRoutingStatus (class in spinnman.model.enums.diagnostic_filter_default_routing_status), 325
DiagnosticFilterDestination (class in spinnman.model.enums), 329
DiagnosticFilterDestination (class in spinnman.model.enums.diagnostic_filter_destination), 326
DiagnosticFilterEmergencyRoutingStatus (class in spinnman.model.enums), 330
DiagnosticFilterEmergencyRoutingStatus (class in spinnman.model.enums.diagnostic_filter_emergency_routing_status), 326
DiagnosticFilterPacketType (class in spinnman.model.enums), 330
DiagnosticFilterPacketType (class in spinnman.model.enums.diagnostic_filter_packet_type), 326
DiagnosticFilterPayloadStatus (class in spinnman.model.enums), 330
DiagnosticFilterPayloadStatus (class in spinnman.model.enums.diagnostic_filter_payload_status), 327
DiagnosticFilterSource (class in spinnman.model.enums), 330
DiagnosticFilterSource (class in spinnman.model.enums.diagnostic_filter_source), 327
DictBasedMachinePartitionNKeysMap (class in pacman.model.routing_info), 168
DictBasedMachinePartitionNKeysMap (class in pacman.model.routing_info.dict_based_machine_partition_n_keys_m

165
disable_motor() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
disable_motor() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 822
disable_retina() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
disable_retina() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
disable_retina() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 822
disable_retina() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
disable_retina_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
disable_retina_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
discard() (spinn_machine.utilities.ordered_set.OrderedSet method), 21
discard() (spinn_utilities.ordered_set.OrderedSet method), 10
discover_scamp_connections() (spinnman.transceiver.Transceiver method), 386
DistanceDependentProbabilityConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 689
DistanceDependentProbabilityConnector (class in spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector), 679
DIVBY0 (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
DIVBY0 (spinnman.model.enums.RunTimeError attribute), 329
DMA_QUEUE_OVERLOADED (spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFr attribute), 544
DMA_QUEUE_OVERLOADED (spinn_front_end_common.interface.provenance.Provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFr attribute), 545
do_injection() (in module pacman.executor.injection_decorator), 94
DOWN_POLARITY (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARetinaDevice attribute), 654
DOWN_POLARITY (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
DOWN_POLARITY (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
DOWN_POLARITY (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
DOWNSAMPLE_16_X_16 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 639
DOWNSAMPLE_16_X_16 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 641
DOWNSAMPLE_16_X_16 (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Retina attribute), 822
DOWNSAMPLE_16_X_16 (spynnaker.pyNN.protocols.RetinaKey attribute), 827
DOWNSAMPLE_32_X_32 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 639
DOWNSAMPLE_32_X_32 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 641
DOWNSAMPLE_32_X_32 (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Retina attribute), 822
DOWNSAMPLE_32_X_32 (spynnaker.pyNN.protocols.RetinaKey attribute), 827
DOWNSAMPLE_64_X_64 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 639
DOWNSAMPLE_64_X_64 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 641
DOWNSAMPLE_64_X_64 (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Retina attribute), 822
DOWNSAMPLE_64_X_64 (spynnaker.pyNN.protocols.RetinaKey attribute), 827
DPRIFlags (class in spinn_front_end_common.utilities.utility_objs.dpri_flag 572
dsef (data_specification.DataSpecificationExecutor attribute), 421
dsef (data_specification.DataSpecificationExecutor attribute), 459
dsg_algorithm (spinn_front_end_common.interface.abstract_spinnaker_bas attribute), 547
DSGRegionReloader (class in spinn_front_end_common.interface.interface_functions), 538
DSGRegionReloader (class in spinn_front_end_common.interface.interface_functions.dsg_regio 531
dtcm (pacman.model.resources.resource_container.ResourceContainer attribute), 155
dtcm (pacman.model.resources.ResourceContainer at-

tribute), 161
DTCM_AVAILABLE (spinn_machine.Processor attribute), 52
dtcm_available (spinn_machine.Processor attribute), 52
DTCM_AVAILABLE (spinn_machine.processor.Processor attribute), 35
dtcm_available (spinn_machine.processor.Processor attribute), 35
DTCMResource (class in pacman.model.resources), 160
DTCMResource (class in pacman.model.resources.dtcm_resource), 153
du_th (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic attribute), 774
du_th (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic attribute), 777
DUMP (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
DUMP (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
DURATION (spinn_front_end_common.interface.profiling.profile_data_model.ProfileData attribute), 541
duration (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson attribute), 796
duration (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 802

E

e_rev_E (spynnaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance attribute), 725
e_rev_E (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance attribute), 727
e_rev_I (spynnaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance attribute), 725
e_rev_I (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance attribute), 727
EAST (pacman.utilities.constants.EDGES attribute), 207
EAST (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
EAST (spinnman.model.enums.P2PTableRoute attribute), 331
edge_partition_identifiers_for_dependent_vertex() (spinn_front_end_common.abstract_models.abstract_vertex_with_dependent_vertices.AbstractVertexWithEdgeToDependent method), 499
edge_partition_identifiers_for_dependent_vertex() (spinn_front_end_common.abstract_models.AbstractVertexWithEdgeToDependentVertices method), 502
edge_partition_identifiers_for_dependent_vertex() (spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl method), 653
edge_partition_identifiers_for_dependent_vertex() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl method), 660
edge_partition_identifiers_for_dependent_vertex() (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.MunichMotorDevice method), 656
edge_partition_identifiers_for_dependent_vertex() (spynnaker.pyNN.external_devices_models.MunichMotorDevice method), 663
EDGES (class in pacman.utilities.constants), 207
edges (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 138
edges (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOut attribute), 140
edges (pacman.model.graphs.AbstractGraph attribute), 144
edges (pacman.model.graphs.abstract.AbstractOutgoingEdgePartition attribute), 146
edges (pacman.model.graphs.impl.Graph attribute), 127
edges (pacman.model.graphs.impl.graph.Graph attribute), 124
edges (pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgeP attribute), 126
edges (pacman.model.graphs.impl.OutgoingEdgePartition attribute), 129
edges_and_partitions() (spinn_front_end_common.utility_models.command_ method), 588
edges_and_partitions() (spinn_front_end_common.utility_models.Comman method), 604
EdgeToNKeysMapper (class in spinn_front_end_common.interface.interface_functions), 538
EdgeToNKeysMapper (class in spinn_front_end_common.interface.interface_functions.edge_to_ 531
EdgeTrafficType (class in pacman.model.graphs.common), 131
EdgeTrafficType (class in pacman.model.graphs.common.edge_traffic_type), 119
EIEIO_COMMAND_IDS (class in spinnman.constants), 380
eieio_data_message (spinnman.messages.eieio.command_messages.host_send_sequenced_d attribute), 251
eieio_data_message (spinnman.messages.eieio.command_messages.HostSendSequencedDat attribute), 255
eieio_header (spinnman.messages.eieio.command_messages.eieio_comman attribute), 249
eieio_header (spinnman.messages.eieio.command_messages.EIEIOComma attribute), 254
eieio_header (spinnman.messages.eieio.data_messages.eieio_data_message attribute), 260
eieio_header (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 264
eieio_type (spinnman.messages.eieio.data_messages.eieio_data_header.EIE attribute), 258
eieio_type (spinnman.messages.eieio.data_messages.EIEIODataHeader

attribute), 262
EIEIOCommandHeader (class in spinnman.messages.eieio.command_messages), 253
EIEIOCommandHeader (class in spinnman.messages.eieio.command_messages.eieio_command_header), 248
EIEIOCommandMessage (class in spinnman.messages.eieio.command_messages), 254
EIEIOCommandMessage (class in spinnman.messages.eieio.command_messages.eieio_command_message), 249
EIEIOConnection (class in spinnman.connections.udp_packet_connections), 240
EIEIOConnection (class in spinnman.connections.udp_packet_connections.eieio_connection), 232
EIEIODataHeader (class in spinnman.messages.eieio.data_messages), 262
EIEIODataHeader (class in spinnman.messages.eieio.data_messages.eieio_data_header), 258
EIEIODataMessage (class in spinnman.messages.eieio.data_messages), 263
EIEIODataMessage (class in spinnman.messages.eieio.data_messages.eieio_data_message), 259
EIEIOPrefix (class in spinnman.messages.eieio), 267
EIEIOPrefix (class in spinnman.messages.eieio.eieio_prefix), 266
EIEIOReceiver (class in spinnman.connections.abstract_classes), 225
EIEIOReceiver (class in spinnman.connections.abstract_classes.eieio_receiver), 219
EIEIOSender (class in spinnman.connections.abstract_classes), 225
EIEIOSender (class in spinnman.connections.abstract_classes.eieio_sender), 220
EIEIOSpikeRecorder (class in spynnaker.pyNN.models.common), 675
EIEIOSpikeRecorder (class in spynnaker.pyNN.models.common.eieio_spike_recorder), 671
EIEIOType (class in spinnman.messages.eieio), 267
EIEIOType (class in spinnman.messages.eieio.eieio_type), 266
EIFConductanceAlphaPopulation (class in spynnaker.pyNN.models.neuron.builds), 718
EIFConductanceAlphaPopulation (class in spynnaker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_ista), 705
ElementAllocatorAlgorithm (class in pacman.utilities.algorithm_utilities), 197
ElementAllocatorAlgorithm (class in pacman.utilities.algorithm_utilities.element_allocator_algorithm), 195
ElementFreeSpace (class in pacman.model.resources), 160
ElementFreeSpace (class in pacman.model.resources.element_free_space), 153
ELSE (data_specification.enums.Commands attribute), 418
ELSE (data_specification.enums.commands.Commands attribute), 415
else_conditional() (data_specification.data_specification_generator.DataSpe method), 432
else_conditional() (data_specification.DataSpecificationGenerator method), 470
emergency_routing_enabled (spinn_machine.Router attribute), 53
emergency_routing_enabled (spinn_machine.router.Router attribute), 36
emergency_routing_statuses (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
emergency_routing_statuses (spinnman.model.DiagnosticFilter attribute), 355
ENABLE (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
ENABLE (spinnman.model.enums.RunTimeError attribute), 329
enable_disable_motor_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 819
enable_disable_motor_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
enable_interrupt_on_counter_event (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
enable_interrupt_on_counter_event (spinnman.model.DiagnosticFilter attribute), 355
enable_motor() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.M static method), 817
enable_motor() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
enable_recording() (spinn_front_end_common.utility_models.reverse_ip_ta method), 600
enable_recording() (spinn_front_end_common.utility_models.reverse_ip_ta method), 601
enable_recording() (spinn_front_end_common.utility_models.ReverseIpTag method), 611

enable_recording() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex method), 612
enable_retina() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
enable_retina() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
end() (spinn_machine.utilities.progress_bar.ProgressBar method), 22
end() (spinn_utilities.progress_bar.ProgressBar method), 12
end_address (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState attribute), 515
end_address (spinn_front_end_common.interface.buffer_management.storage_objects.ChannelBufferState attribute), 523
end_conditional() (data_specification.data_specification_generator.DataSpecificationGenerator method), 432
end_conditional() (data_specification.DataSpecificationGenerator method), 470
END_CONSTRUCTOR (data_specification.enums.Commands attribute), 418
END_CONSTRUCTOR (data_specification.enums.commands.Commands attribute), 415
end_function() (data_specification.data_specification_generator.DataSpecificationGenerator method), 433
end_function() (data_specification.DataSpecificationGenerator method), 470
END_IF (data_specification.enums.Commands attribute), 418
END_IF (data_specification.enums.commands.Commands attribute), 415
END_LOOP (data_specification.enums.Commands attribute), 418
END_LOOP (data_specification.enums.commands.Commands attribute), 415
end_loop() (data_specification.data_specification_generator.DataSpecificationGenerator method), 433
end_loop() (data_specification.DataSpecificationGenerator method), 471
END_SPEC (data_specification.enums.Commands attribute), 418
END_SPEC (data_specification.enums.commands.Commands attribute), 415
end_specification() (data_specification.data_specification_generator.DataSpecificationGenerator method), 433
end_specification() (data_specification.DataSpecificationGenerator method), 471
END_STRUCT (data_specification.enums.Commands attribute), 418
END_STRUCT (data_specification.enums.commands.Commands attribute), 415
EndBufferingState (class in spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState), 516
EndBufferingState (class in spinn_front_end_common.interface.buffer_management.storage_objects.ChannelBufferState), 523
ENERGY_DETAILED_FILENAME (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
ENERGY_SUMMARY_FILENAME (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
EnergyReport (class in spinn_front_end_common.utilities.report_functions.energy_report), 564
ensure_board_is_ready() (spinnman.transceiver.Transceiver method), 386
eof() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorageGenerator method), 66
eof() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage method), 65
eof() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedBytearrayDataStorage method), 69
eof() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage method), 70
eof() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorageGenerator method), 70
eof() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
eof() (spinn_storage_handlers.BufferedFileDataStorage method), 73
eof() (spinn_storage_handlers.BufferedTempfileDataStorage method), 73
EQUAL (data_specification.enums.Condition attribute), 419
EQUAL (data_specification.enums.condition.Condition attribute), 416
error_status (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 343
error_status (spinnman.model.RouterDiagnostics attribute), 358
ethernet_connected_chips (spinn_machine.Machine attribute), 48
ethernet_connected_chips (spinn_machine.machine.Machine attribute), 31
ethernet_ip_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBootValues attribute), 318
ethernet_ip_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
ethernet_ip_address (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
ethernet_ip_address (spinnman.model.ChipSummaryInfo attribute), 350
EthernetCommandConnection (class in spyn-

naker.pyNN.connections), 620
EthernetCommandConnection (class in spynnaker.pyNN.connections.ethernet_command_connection), 618
EthernetControlConnection (class in spynnaker.pyNN.connections), 620
EthernetControlConnection (class in spynnaker.pyNN.connections.ethernet_control_connection), 619
eval() (spinn_utilities.safe_eval.SafeEval method), 13
EVENT (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
EVENT (spinnman.model.enums.RunTimeError attribute), 330
EVENT_PADDING (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
EVENT_STOP (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
EVENTS_IN_PAYLOAD (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload attribute), 822
EVENTS_IN_PAYLOAD (spynnaker.pyNN.protocols.RetinaPayload attribute), 827
EventStopRequest (class in spinnman.messages.eieio.command_messages), 254
EventStopRequest (class in spinnman.messages.eieio.command_messages.event_stop_request), 250
exception (pacman.exceptions.PacmanAlgorithmFailedToCompleteException attribute), 211
exception (spinnman.exceptions.SpinnmanGenericProcessException attribute), 381
exception_handler() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 547
ExecutableFailedToStartException, 578
ExecutableFailedToStopException, 578
ExecutableFinder (class in spinn_front_end_common.utilities.utility_objs), 575
ExecutableFinder (class in spinn_front_end_common.utilities.utility_objs.executable_finder), 572
ExecutableFinder (class in spinn_utilities.executable_finder), 9
ExecutableNotFoundException, 578
ExecutableTargets (class in spinnman.model), 355
ExecutableTargets (class in spinnman.model.executable_targets), 340
ExecutableType (class in spinn_front_end_common.utilities.utility_objs), 575
ExecutableType (class in spinn_front_end_common.utilities.utility_objs.executable_type), 572
execute() (data_specification.data_specification_executor.DataSpecificationE method), 421
execute() (data_specification.DataSpecificationExecutor method), 459
execute() (spinnman.processes.send_single_command_process.SendSingleC method), 367
execute() (spinnman.processes.SendSingleCommandProcess method), 371
execute() (spinnman.transceiver.Transceiver method), 387
execute_align_wr_ptr() (data_specification.data_specification_executor_fun method), 422
execute_align_wr_ptr() (data_specification.DataSpecificationExecutorFunct method), 460
execute_application() (spinnman.transceiver.Transceiver method), 388
execute_arith_op() (data_specification.data_specification_executor_functior method), 422
execute_arith_op() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_block_copy() (data_specification.data_specification_executor_func method), 422
execute_block_copy() (data_specification.DataSpecificationExecutorFunctio method), 460
execute_break() (data_specification.data_specification_executor_functions.I method), 422
execute_break() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_break_loop() (data_specification.data_specification_executor_funct method), 422
execute_break_loop() (data_specification.DataSpecificationExecutorFunctio method), 460
execute_construct() (data_specification.data_specification_executor_functio method), 422
execute_construct() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_copy_param() (data_specification.data_specification_executor_func method), 422
execute_copy_param() (data_specification.DataSpecificationExecutorFunct method), 460
execute_copy_struct() (data_specification.data_specification_executor_func method), 422
execute_copy_struct() (data_specification.DataSpecificationExecutorFunctio method), 460
execute_declare_rng() (data_specification.data_specification_executor_func method), 422
execute_declare_rng() (data_specification.DataSpecificationExecutorFunctio method), 460
execute_else() (data_specification.data_specification_executor_functions.Da method), 422
execute_else() (data_specification.DataSpecificationExecutorFunctions

method), 460
execute_end_constructor() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_end_constructor() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_end_if() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_end_if() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_end_loop() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_end_loop() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_end_spec() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_end_spec() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_end_struct() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_end_struct() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_flood() (spinnman.transceiver.Transceiver method), 388
execute_free() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_free() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_get_random_rumber() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_get_random_rumber() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_get_wr_ptr() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_get_wr_ptr() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_if() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_if() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_logic_op() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_logic_op() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_loop() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_loop() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_mapping() (pacman.executor.pacman_algorithm_executor.PACMANAlgorithmExecutor method), 96
execute_mapping() (pacman.executor.PACMANAlgorithmExecutor method), 98
execute_mv() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 422
execute_mv() (data_specification.DataSpecificationExecutorFunctions method), 460
execute_nop() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_nop() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_print_struct() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_print_struct() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_print_val() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_print_val() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_random_dist() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_random_dist() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_read() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_read() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_read_param() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_read_param() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_reference() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_reference() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_reserve() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_reserve() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_set_wr_ptr() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_set_wr_ptr() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_start_constructor() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423

execute_start_constructor() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_start_struct() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_start_struct() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_struct_elem() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_struct_elem() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_switch_focus() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 423
execute_switch_focus() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_write() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 424
execute_write() (data_specification.DataSpecificationExecutorFunctions method), 461
execute_write_array() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 424
execute_write_array() (data_specification.DataSpecificationExecutorFunctions method), 462
execute_write_param() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 424
execute_write_param() (data_specification.DataSpecificationExecutorFunctions method), 462
execute_write_param_component() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 424
execute_write_param_component() (data_specification.DataSpecificationExecutorFunctions method), 462
execute_write_struct() (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions method), 424
execute_write_struct() (data_specification.DataSpecificationExecutorFunctions method), 462
ExecuteBreakInstruction, 455
EXIT (spinnman.messages.scp.enums.Signal attribute), 280
EXIT (spinnman.messages.scp.enums.signal.Signal attribute), 277
expand_to_bit_array() (in module pacman.utilities.utility_calls), 209
extend() (pacman.model.resources.pre_allocated_resource_container.PreAllocatedResourceContainer method), 155
extend() (pacman.model.resources.PreAllocatedResourceContainer method), 163
extend() (pacman.model.resources.resource_container.ResourceContainer method), 155
extend() (pacman.model.resources.ResourceContainer method), 161
extend_allocation() (spinn_front_end_common.abstract_models.abstractmachineallocationcontroller.AbstractMachineAllocationController method), 496
extend_allocation() (spinn_front_end_common.abstract_models.AbstractMachineAllocationController method), 500
extend_extra_load_algorithms() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
extend_extra_mapping_algorithms() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
extend_extra_post_run_algorithms() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
extend_extra_pre_run_algorithms() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
ExternalAlgorithmFunctions (class in pacman.executor.algorithm_classes), 85
ExternalAlgorithmFunctions (class in pacman.executor.algorithm_classes.external_algorithm), 83
ExternalCochleaDevice (class in spynnaker.pyNN.external_devices_models), 661
ExternalCochleaDevice (class in spynnaker.pyNN.external_devices_models.external_spinnaker_link_cochlea), 653
ExternalDeviceLifControl (class in spynnaker.pyNN.external_devices_models), 660
ExternalDeviceLifControlDataSpecificationExecutorFunctions (class in spynnaker.pyNN.external_devices_models.external_device_lif_control_data_specification), 653
ExternalFPGARetinaDevice (class in spynnaker.pyNN.external_devices_models), 661
ExternalFPGARetinaDevice (class in spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina), 654
EXTRA_MONITOR_CORE_RE_INJECTION (spinn_front_end_common.utilities.constants.SDP_PORTS attribute), 577
extract_provenance() (spinn_front_end_common.interface.provenance.pacman_provenance.PacmanProvenanceExtractor method), 544
extract_provenance() (spinn_front_end_common.interface.provenance.PacmanProvenanceExtractor method), 545
extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.abstractmasterpoptablegenerator.AbstractMasterPopTableGenerator method), 727
extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generator.MasterPopTableGenerator method), 729
extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generator.MasterPopTableGenerator method), 730
extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerator method), 731
extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerator method), 732
ExtraMonitorSupportApplicationVertex (class in

spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex), 590
ExtraMonitorSupportMachineVertex (class in spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex), 591

F

FailedState (class in spinn_front_end_common.utilities), 583
FailedState (class in spinn_front_end_common.utilities.failed_state), 578
FakeHBPPortalMachineProvider (class in spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider), 829
fan_0 (spinnman.model.adc_info.ADCInfo attribute), 331
fan_0 (spinnman.model.ADCInfo attribute), 347
fan_1 (spinnman.model.adc_info.ADCInfo attribute), 331
fan_1 (spinnman.model.ADCInfo attribute), 347
Field (class in pacman.utilities.utility_objs), 203
Field (class in pacman.utilities.utility_objs.field), 199
fields (pacman.model.constraints.key_allocator_constraints.fixed_key_field_constraint attribute), 100
fields (pacman.model.constraints.key_allocator_constraints.FixedKeyFieldConstraint attribute), 101
fields (pacman.model.constraints.key_allocator_constraints.flexi_key_field_constraint attribute), 100
fields (pacman.model.constraints.key_allocator_constraints.FlexiKeyFieldConstraint attribute), 102
file_name_type (pacman.executor.algorithm_decorators.Output attribute), 92
file_name_type (pacman.executor.algorithm_decorators.output_decorator attribute), 90
FileDataReader (class in spinn_storage_handlers), 74
FileDataReader (class in spinn_storage_handlers.file_data_reader), 71
FileDataWriter (class in spinn_storage_handlers), 74
FileDataWriter (class in spinn_storage_handlers.file_data_writer), 72
FileIO (class in spinnman.utilities.io), 376
FileIO (class in spinnman.utilities.io.file_io), 373
filename (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage attribute), 70
filename (spinn_storage_handlers.BufferedFileDataStorage attribute), 73
filename (spinn_storage_handlers.file_data_writer.FileDataWriter attribute), 72
filename (spinn_storage_handlers.FileDataWriter attribute), 75
fill() (spinnman.utilities.io.abstract_io.AbstractIO method), 372
fill() (spinnman.utilities.io.io.AbstractIO method), 376
fill() (spinnman.utilities.io.file_io.FileIO method), 374
fill() (spinnman.utilities.io.FileIO method), 376
fill() (spinnman.utilities.io.memory_io.MemoryIO method), 375
fill() (spinnman.utilities.io.MemoryIO method), 378
fill_memory() (spinnman.processes.fill_process.FillProcess method), 363
fill_memory() (spinnman.processes.FillProcess method), 369
fill_memory() (spinnman.transceiver.Transceiver method), 389
FillDataType (class in spinnman.processes), 369
FillDataType (class in spinnman.processes.fill_process), 363
FillProcess (class in spinnman.processes), 369
FillProcess (class in spinnman.processes.fill_process), 363
FillRequest (class in spinnman.messages.scp.impl), 298
FillRequest (class in spinnman.messages.scp.impl.fill_request), 283
filter() (spinn_utilities.log.ConfiguredFilter method), 10
filter_edge() (spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge method), 666
filter_edge() (spynnaker.pyNN.models.abstract_models.AbstractFilterableEdge method), 668
filter_edge() (spynnaker.pyNN.models.neural_projections.delay_afferent_machine_vertices.DelayAfferentMachineVertex method), 696
filter_edge() (spynnaker.pyNN.models.neural_projections.DelayAfferentMachineVertex method), 699
filter_edge() (spynnaker.pyNN.models.neural_projections.delayed_machine_edge.DelayedMachineEdge method), 697
filter_edge() (spynnaker.pyNN.models.neural_projections.DelayedMachineEdge method), 700
filter_edge() (spynnaker.pyNN.models.neural_projections.projection_machine_edge.ProjectionMachineEdge method), 698
filter_edge() (spynnaker.pyNN.models.neural_projections.ProjectionMachineEdge method), 701
filter_word (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
filter_word (spinnman.model.DiagnosticFilter attribute), 355
FilterableException, 834
finish() (spinnman.connections.scp_request_pipeline.SCPRequestPipeLine method), 244
finish() (spinnman.connections.SCPRequestPipeLine method), 246
finish() (spynnaker.pyNN.models.neuron.connection_holder.ConnectionHolder method), 783
finish() (spynnaker.pyNN.models.neuron.ConnectionHolder method), 791
finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_generator.AbstractMasterPopTableGenerator method), 728

finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAs2dArray method), 729
finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch method), 730
finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAs2dArray method), 731
finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch method), 732
FINISHED (spinn_front_end_common.interface.simulator_state.Simulator_State attribute), 550
FINISHED (spinnman.model.enums.cpu_state.CPUState attribute), 325
FINISHED (spinnman.model.enums.CPUState attribute), 329
FIQ (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
FIQ (spinnman.model.enums.RunTimeError attribute), 330
first_free_router_entry (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
first_free_router_entry (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
first_free_router_entry (spinnman.model.chip_info.ChipInfo attribute), 333
first_free_router_entry (spinnman.model.ChipInfo attribute), 348
first_id (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 814
first_key (pacman.model.routing_info.partition_routing_info.PartitionRoutingInfo attribute), 166
first_key (pacman.model.routing_info.PartitionRoutingInfo attribute), 169
first_key_and_mask (pacman.model.routing_info.partition_routing_info.PartitionRoutingInfo attribute), 166
first_key_and_mask (pacman.model.routing_info.PartitionRoutingInfo attribute), 169
first_mask (pacman.model.routing_info.partition_routing_info.PartitionRoutingInfo attribute), 166
first_mask (pacman.model.routing_info.PartitionRoutingInfo attribute), 169
FIRST_STAGE (spinnman.model.enums.diagnostic_filter_emergency_routing_status.DiagnosticFilterEmergencyRoutingStatus attribute), 326
FIRST_STAGE (spinnman.model.enums.DiagnosticFilterEmergencyRoutingStatus attribute), 330
FIRST_STAGE_COMBINED (spinnman.model.enums.diagnostic_filter_emergency_routing_status.DiagnosticFilterEmergencyRoutingStatus attribute), 326
FIRST_STAGE_COMBINED (spinnman.model.enums.DiagnosticFilterEmergencyRoutingStatus attribute), 330
FIXED (pacman.utilities.algorithm_utilities.field_based_system_utilities.FieldBasedSystemUtilities attribute), 195
FIXED_KEY (pacman.utilities.algorithm_utilities.field_based_system_utilities.FieldBasedSystemUtilities attribute), 195
FIXED_KEY (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpinnakerLinkProtocol attribute), 822
FIXED_KEY (spynnaker.pyNN.protocols.RetinaKey attribute), 827
FIXED_MASK (pacman.utilities.algorithm_utilities.field_based_system_utilities.FieldBasedSystemUtilities attribute), 195
FIXED_ROUTE (pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType attribute), 119
FIXED_ROUTE (pacman.model.graphs.common.EdgeTrafficType attribute), 121
FIXED_ROUTE (spinn_front_end_common.utilities.utility_objs.dpri_flags.DPRIFlags attribute), 572
FIXED_ROUTE (spinnman.model.enums.diagnostic_filter_packet_type.DiagnosticFilterPacketType attribute), 326
FIXED_ROUTE (spinnman.model.enums.DiagnosticFilterPacketType attribute), 330
fixed_route_copy (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
fixed_route_copy (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
fixed_size (spinnman.messages.scp.impl.iptag_get_info_response.IPTagGetInfoResponse attribute), 288
FixedKeyAndMaskConstraint (class in pacman.model.constraints.key_allocator_constraints), 101
FixedKeyAndMaskConstraint (class in pacman.model.constraints.key_allocator_constraints.fixed_key_and_mask_constraint), 99
FixedKeyFieldConstraint (class in pacman.model.constraints.key_allocator_constraints), 101
FixedKeyFieldConstraint (class in pacman.model.constraints.key_allocator_constraints.fixed_key_field_constraint), 100
FixedMaskConstraint (class in pacman.model.constraints.key_allocator_constraints), 101
FixedMaskConstraint (class in pacman.model.constraints.key_allocator_constraints.fixed_mask_constraint),

100
FixedNumberPostConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 690
FixedNumberPostConnector (class in spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector), 680
FixedNumberPreConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 690
FixedNumberPreConnector (class in spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector), 681
FixedProbabilityConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 691
FixedProbabilityConnector (class in spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector), 682
FixedRouteEntry (class in spinn_machine), 45
FixedRouteEntry (class in spinn_machine.fixed_route_entry), 28
FixedRouteFromMachineReport (class in spinn_front_end_common.utilities.report_functions.fixed_route_from_machine_report), 564
FixedRouteInit (class in spinnman.messages.scp.impl), 307
FixedRouteInit (class in spinnman.messages.scp.impl.fixed_route_init), 283
FixedRouteRead (class in spinnman.messages.scp.impl), 305
FixedRouteRead (class in spinnman.messages.scp.impl.fixed_route_read), 284
FixedRouteRouter (class in pacman.operations.fixed_route_router.fixed_route_router), 185
flags (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
flags (spinnman.messages.sdp.SDPHeader attribute), 313
FlexiField (class in pacman.utilities.utility_objs), 203
FlexiField (class in pacman.utilities.utility_objs.flexi_field), 199
FlexiKeyFieldConstraint (class in pacman.model.constraints.key_allocator_constraints), 102
FlexiKeyFieldConstraint (class in pacman.model.constraints.key_allocator_constraints.flexi_key_field_constraint), 100
FLOAT_32 (data_specification.enums.data_type.DataType attribute), 416
FLOAT_32 (data_specification.enums.DataType attribute), 419
FLOAT_64 (data_specification.enums.data_type.DataType attribute), 416
FLOAT_64 (data_specification.enums.DataType attribute), 419
float_to_fixed() (in module spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers), 743
FLOOD_FILL_BLOCK (spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.SpinnakerBootOpCode attribute), 317
FLOOD_FILL_BLOCK (spinnman.messages.spinnaker_boot.SpinnakerBootOpCode attribute), 321
FLOOD_FILL_CONTROL (spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.SpinnakerBootOpCode attribute), 317
FLOOD_FILL_CONTROL (spinnman.messages.spinnaker_boot.SpinnakerBootOpCode attribute), 321
FLOOD_FILL_START (spinnman.messages.spinnaker_boot.spinnaker_boot_op_code.SpinnakerBootOpCode attribute), 317
FLOOD_FILL_START (spinnman.messages.spinnaker_boot.SpinnakerBootOpCode attribute), 321
FloodFillData (class in spinnman.messages.scp.impl), 298
FloodFillData (class in spinnman.messages.scp.impl.flood_fill_data), 284
FloodFillEnd (class in spinnman.messages.scp.impl), 298
FloodFillEnd (class in spinnman.messages.scp.impl.flood_fill_end), 285
FloodFillStart (class in spinnman.messages.scp.impl), 299
FloodFillStart (class in spinnman.messages.scp.impl.flood_fill_start), 285
flush() (spinnman.utilities.io.abstract_io.AbstractIO method), 373
flush() (spinnman.utilities.io.AbstractIO method), 376
flush() (spinnman.utilities.io.file_io.FileIO method), 374
flush() (spinnman.utilities.io.FileIO method), 377
flush() (spinnman.utilities.io.memory_io.MemoryIO method), 375
flush() (spinnman.utilities.io.MemoryIO method), 378
flushing_data_from_region() (spinn_front_end_common.interface.buffer_management.storage_method), 598
flushing_data_from_region() (spinn_front_end_common.interface.buffer_management.storage_method), 517
fpga_id (pacman.model.graphs.abstract_fpga_vertex.AbstractFPGAVertex attribute), 137

fpga_id (pacman.model.graphs.AbstractFPGAVertex attribute), 143
fpga_id (pacman.model.graphs.application.application_fpga_vertex.ApplicationFPGAVertex attribute), 110
fpga_id (pacman.model.graphs.application.ApplicationFPGAVertex attribute), 115
fpga_id (pacman.model.graphs.machine.machine_fpga_vertex.MachineFPGAVertex attribute), 131
fpga_id (pacman.model.graphs.machine.MachineFPGAVertex attribute), 134
fpga_id (spinn_machine.link_data_objects.fpga_link_data.FPGALinkData attribute), 17
fpga_id (spinn_machine.link_data_objects.FPGALinkData attribute), 18
fpga_link_id (pacman.model.graphs.abstract_fpga_vertex.AbstractFPGAVertex attribute), 137
fpga_link_id (pacman.model.graphs.AbstractFPGAVertex attribute), 143
fpga_link_id (pacman.model.graphs.application.application_fpga_vertex.ApplicationFPGAVertex attribute), 110
fpga_link_id (pacman.model.graphs.application.ApplicationFPGAVertex attribute), 115
fpga_link_id (pacman.model.graphs.machine.machine_fpga_vertex.MachineFPGAVertex attribute), 131
fpga_link_id (pacman.model.graphs.machine.MachineFPGAVertex attribute), 134
fpga_link_id (spinn_machine.link_data_objects.fpga_link_data.FPGALinkData attribute), 17
fpga_link_id (spinn_machine.link_data_objects.FPGALinkData attribute), 18
FPGALinkData (class in spinn_machine.link_data_objects), 17
FPGALinkData (class in spinn_machine.link_data_objects.fpga_link_data), 17
frame (spinnman.connections.udp_packet_connections.bmp_connection.BMPConnection attribute), 231
frame (spinnman.connections.udp_packet_connections.BMPConnection attribute), 237
frame (spinnman.model.bmp_connection_data.BMPConnectionData attribute), 332
frame (spinnman.model.BMPConnectionData attribute), 348
FREE (data_specification.enums.Commands attribute), 418
FREE (data_specification.enums.commands.Commands attribute), 415
FREE (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES attribute), 818
FREE (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES attribute), 823
free_id() (spinnman.utilities.appid_tracker.AppIdTracker method), 379
free_memory_region() (data_specification.data_specification_generator.DataSpecificationGenerator method), 433
free_memory_region() (data_specification.DataSpecificationGenerator method), 471
FREE_ROUTING_BY_APP_ID (spinnman.messages.scp.enums.alloc_free.AllocFree attribute), 274
FREE_ROUTING_BY_APP_ID (spinnman.messages.scp.enums.AllocFree attribute), 277
FREE_ROUTING_BY_POINTER (spinnman.messages.scp.enums.alloc_free.AllocFree attribute), 274
FREE_ROUTING_BY_POINTER (spinnman.messages.scp.enums.AllocFree attribute), 277
free_sdram() (spinnman.transceiver.Transceiver method), 389
FREE_SDRAM_BY_APP_ID (spinnman.messages.scp.enums.alloc_free.AllocFree attribute), 274
FREE_SDRAM_BY_APP_ID (spinnman.messages.scp.enums.AllocFree attribute), 278
free_sdram_by_app_id() (spinnman.transceiver.Transceiver method), 389
FREE_SDRAM_BY_POINTER (spinnman.messages.scp.enums.alloc_free.AllocFree attribute), 274
FREE_SDRAM_BY_POINTER (spinnman.messages.scp.enums.AllocFree attribute), 278
from_bytestring() (spinnman.messages.eieio.command_messages.database_confirmation.DatabaseConfirmation static method), 248
from_bytestring() (spinnman.messages.eieio.command_messages.DatabaseConfirmation static method), 253
from_bytestring() (spinnman.messages.eieio.command_messages.eieio_command_header.EIEIOCommandHeader static method), 249
from_bytestring() (spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage static method), 249
from_bytestring() (spinnman.messages.eieio.command_messages.EIEIOCommandHeader static method), 254
from_bytestring() (spinnman.messages.eieio.command_messages.EIEIOCommandMessage static method), 254
from_bytestring() (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead static method), 250
from_bytestring() (spinnman.messages.eieio.command_messages.host_data_read_ack.HostDataReadAck

static method), 250
from_bytestring() (spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData static method), 251
from_bytestring() (spinnman.messages.eieio.command_messages.HostDataRead static method), 255
from_bytestring() (spinnman.messages.eieio.command_messages.HostDataRead2Ack static method), 256
from_bytestring() (spinnman.messages.eieio.command_messages.HostSendSequencedData static method), 255
from_bytestring() (spinnman.messages.eieio.command_messages.notification_protocol_pause_stop.NotificationProtocolPauseStop static method), 251
from_bytestring() (spinnman.messages.eieio.command_messages.notification_protocol_start_resume.NotificationProtocolStartResume static method), 251
from_bytestring() (spinnman.messages.eieio.command_messages.NotificationProtocolPauseStop static method), 255
from_bytestring() (spinnman.messages.eieio.command_messages.NotificationProtocolStartResume static method), 255
from_bytestring() (spinnman.messages.eieio.command_messages.spinnaker_request_buffers.SpinnakerRequestBuffers static method), 252
from_bytestring() (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData static method), 252
from_bytestring() (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffers static method), 256
from_bytestring() (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData static method), 256
from_bytestring() (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader static method), 258
from_bytestring() (spinnman.messages.eieio.data_messages.EIEIODataHeader static method), 262
from_bytestring() (spinnman.messages.scp.scp_response_header.SCPResponseHeader static method), 308
from_bytestring() (spinnman.messages.scp.SCPResponseHeader static method), 309
from_bytestring() (spinnman.messages.sdp.sdp_header.SDPHeader static method), 311
from_bytestring() (spinnman.messages.sdp.sdp_message.SDPMessage static method), 312
from_bytestring() (spinnman.messages.sdp.SDPHeader static method), 314
from_bytestring() (spinnman.messages.sdp.SDPMessage static method), 315
from_bytestring() (spinnman.messages.spinnaker_boot.spinnaker_boot_message.SpinnakerBootMessage static method), 316
from_bytestring() (spinnman.messages.spinnaker_boot.SpinnakerBootMessage static method), 320
FromFileConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 692
FromFileConnector (class in spynnaker.pyNN.models.neural_projections.connectors.from_file_connector), 683
FromListConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 692
FromListConnector (class in spynnaker.pyNN.models.neural_projections.connectors.from_list_connector), 683
function (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 434
function (data_specification.DataSpecificationGenerator attribute), 472

G

generate_application_data_specification() (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.MunichMotorDevice method), 656
generate_application_data_specification() (spynnaker.pyNN.external_devices_models.MunichMotorDevice method), 663
generate_comparison_router_report() (in module pacman.operations.algorithm_reports.reports), 180
generate_data_specification() (spinn_front_end_common.abstract_models.abstract_generates_data_specification.AbstractGeneratesDataSpecification method), 495
generate_data_specification() (spinn_front_end_common.abstract_models.AbstractGeneratesDataSpecification method), 499
generate_data_specification() (spinn_front_end_common.abstract_models.impl.machine_data_specification.MachineDataSpecificationGenerator method), 493
generate_data_specification() (spinn_front_end_common.abstract_models.impl.MachineDataSpecificationGenerator method), 494
generate_data_specification() (spinn_front_end_common.utility_models.chip_power_monitor_application_vertex.ChipPowerMonitorApplicationVertex method), 585
generate_data_specification() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex

method), 586
generate_data_specification() (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex method), 588
generate_data_specification() (spinn_front_end_common.utility_models.command_sender_machine_vertex_with_dependent_vertices.CommandSenderMachineVertexWithDependentVertices method), 590
generate_data_specification() (spinn_front_end_common.utility_models.CommandSender method), 604
generate_data_specification() (spinn_front_end_common.utility_models.CommandSenderMachineVertex method), 605
generate_data_specification() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupportMachineVertex method), 590
generate_data_specification() (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex method), 591
generate_data_specification() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 594
generate_data_specification() (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex method), 596
generate_data_specification() (spinn_front_end_common.utility_models.LivePacketGather method), 606
generate_data_specification() (spinn_front_end_common.utility_models.LivePacketGatherMachineVertex method), 608
generate_data_specification() (spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.ReverseIpTagMultiCastSource method), 600
generate_data_specification() (spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source_machine_vertex.ReverseIpTagMultiCastSourceMachineVertex method), 602
generate_data_specification() (spinn_front_end_common.utility_models.ReverseIpTagMultiCastSource method), 611
generate_data_specification() (spinn_front_end_common.utility_models.ReverseIpTagMultiCastSourceMachineVertex method), 613
generate_data_specification() (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.MunichMotorDevice method), 656
generate_data_specification() (spynnaker.pyNN.external_devices_models.MunichMotorDevice method), 663
generate_data_specification() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 779
generate_data_specification() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 787
generate_data_specification() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 796
generate_data_specification() (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex method), 802
generate_data_specification() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 807
generate_data_specification() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 811
generate_file_machine() (spinn_front_end_common.interface.abstract_spinnaker_protocol.AbstractSpinnakerProtocol method), 548
generate_machine_data_specification() (spinn_front_end_common.abstract_models.impl.machine_data_specifiable.MachineDataSpecifiableMachineVertex method), 491
generate_machine_data_specification() (spinn_front_end_common.abstract_models.impl.MachineDataSpecifiable method), 495
generate_machine_edges() (in module pacman.utilities.algorithm_utilities.partition_algorithm_utilities), 196
generate_machine_report() (in module spinnman.utilities.reports), 379
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector method), 678
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 680
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 681
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
generate_on_machine() (spyn-

naker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector method), 691
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 691
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 683
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector method), 692
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector method), 684
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector method), 686
generate_on_machine() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
generate_unique_folder_name() (in module spinn_front_end_common.utilities.helpful_functions), 580
generic_motor0_raw_output_leak_to_0() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
generic_motor0_raw_output_leak_to_0() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
generic_motor0_raw_output_leak_to_0_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
generic_motor0_raw_output_leak_to_0_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
generic_motor0_raw_output_permanent() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
generic_motor0_raw_output_permanent() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
generic_motor0_raw_output_permanent_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
generic_motor0_raw_output_permanent_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
generic_motor1_raw_output_leak_to_0() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
generic_motor1_raw_output_leak_to_0() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
generic_motor1_raw_output_leak_to_0_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
generic_motor1_raw_output_leak_to_0_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
generic_motor1_raw_output_permanent() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
generic_motor1_raw_output_permanent() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
generic_motor1_raw_output_permanent_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
generic_motor1_raw_output_permanent_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
generic_motor_disable() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
generic_motor_disable() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
generic_motor_enable() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
generic_motor_enable() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
generic_motor_total_period() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
generic_motor_total_period() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
generic_motor_total_period_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
generic_motor_total_period_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
GET (spinnman.messages.scp.enums.iptag_command.IPTagCommand

attribute), 274
GET (spinnman.messages.scp.enums.IPTagCommand attribute), 279
get() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 814
get_algorithms() (in module pacman.executor.algorithm_decorators), 94
get_algorithms() (in module pacman.executor.algorithm_decorators.algorithm_decorator), 88
get_allowed_row_length() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array.MasterPopTableAs2dArray method), 729
get_allowed_row_length() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAs2dArray method), 730
get_allowed_row_length() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAs2dArray method), 731
get_allowed_row_length() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch method), 732
get_application_edge() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 120
get_application_edge() (pacman.model.graphs.common.GraphMapper method), 122
get_application_vertex() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 120
get_application_vertex() (pacman.model.graphs.common.GraphMapper method), 122
get_atom_id_to_key_mapping() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 555
get_atom_id_to_key_mapping() (spinn_front_end_common.utilities.database.DatabaseReader method), 559
get_binary_file_name() (spinn_front_end_common.abstract_models.AbstractHasAssociatedBinary method), 496
get_binary_file_name() (spinn_front_end_common.abstract_models.AbstractHasAssociatedBinary method), 499
get_binary_file_name() (spinn_front_end_common.utility_models.chip_power_monitor.ChipPowerMonitorMachineVertex method), 585
get_binary_file_name() (spinn_front_end_common.utility_models.chip_power_monitor.ChipPowerMonitorMachineVertex method), 586
get_binary_file_name() (spinn_front_end_common.utility_models.command_sender.CommandSender method), 588
get_binary_file_name() (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex method), 590
get_binary_file_name() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 604
get_binary_file_name() (spinn_front_end_common.utility_models.CommandSender method), 605
get_binary_file_name() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupportMachineVertex method), 590
get_binary_file_name() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupportMachineVertex method), 592
get_binary_file_name() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 594
get_binary_file_name() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 596
get_binary_file_name() (spinn_front_end_common.utility_models.LivePacketGather method), 606
get_binary_file_name() (spinn_front_end_common.utility_models.LivePacketGather method), 608
get_binary_file_name() (spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.ReverseIPTagMultiCastSource method), 600
get_binary_file_name() (spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source.ReverseIPTagMultiCastSource method), 605
get_binary_file_name() (spinn_front_end_common.utility_models.ReverseIPTagMultiCastSource method), 611
get_binary_file_name() (spinn_front_end_common.utility_models.ReverseIPTagMultiCastSource method), 613
get_binary_file_name() (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.MunichSpinnakerLinkMotorDevice method), 656
get_binary_file_name() (spynnaker.pyNN.external_devices_models.MunichMotorDevice method), 663
get_binary_file_name() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_binary_file_name() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_binary_file_name() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 796
get_binary_file_name() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 803
get_binary_file_name() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 808
get_binary_file_name() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 811
get_binary_start_type() (spinn_machine.abstract_models.AbstractHasBinary method), 496
get_binary_start_type() (spinn_front_end_common.abstract_models.AbstractHasAssociatedBinary method), 499
get_binary_start_type() (spinn_front_end_common.utility_models.chip_power_monitor.ChipPowerMonitorMachineVertex method), 585
get_binary_start_type() (spinn_front_end_common.utility_models.chip_power_monitor.ChipPowerMonitorMachineVertex method), 586

get_binary_start_type() (spinn_front_end_common.utility_models.command_sender.CommandSender method), 588
get_binary_start_type() (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex method), 590
get_binary_start_type() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 604
get_binary_start_type() (spinn_front_end_common.utility_models.LivePacketGatherMachineVertex method), 605
get_binary_start_type() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupport method), 591
get_binary_start_type() (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex method), 592
get_binary_start_type() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 595
get_binary_start_type() (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex method), 596
get_binary_start_type() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 606
get_binary_start_type() (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex method), 608
get_binary_start_type() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source.ReverseIPTagMulticastSource method), 600
get_binary_start_type() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex method), 602
get_binary_start_type() (spinn_front_end_common.utility_models.ReverseIpTagMultiCastSourceMachineVertex method), 611
get_binary_start_type() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex method), 613
get_binary_start_type() (spynaker.pyNN.external_devices_models.munich_spinnaker_interface.MunichMotorDevice method), 656
get_binary_start_type() (spynaker.pyNN.external_devices_models.MunichMotorDevice method), 663
get_binary_start_type() (spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_binary_start_type() (spynaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_binary_start_type() (spynaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 796
get_binary_start_type() (spynaker.pyNN.models.spike_source.SpikeSourcePoisson method), 803
get_binary_start_type() (spynaker.pyNN.models.utility_models.delay_extension.DelayExtension method), 808
get_binary_start_type() (spynaker.pyNN.models.utility_models.DelayExtensionVertex method), 811
get_block_n_bytes() (spynaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO method), 765
get_block_n_bytes() (spynaker.pyNN.models.neuron.synapse_io.AbstractSynapseIO method), 765
get_block_n_bytes() (spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 765
get_block_n_bytes() (spynaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 765
get_block_n_bytes() (spinn_front_end_common.utility_models.command_sender.CommandSender method), 766
get_block_n_bytes() (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex method), 766
get_board_mask() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupport static method), 269
get_board_mask() (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex static method), 271
get_body() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 3
get_buffer_size() (spynaker.pyNN.models.common.recording_utils.RecordingRegion method), 676
get_buffer_size() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 678
get_buffer_size() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 680
get_buffer_size() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 600
get_buffer_size() (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex method), 602
get_bytestring() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source.ReverseIPTagMulticastSource method), 611
get_bytestring() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex method), 613
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 656
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 663
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 780
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegion method), 788
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 796
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 803
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 808
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 811
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 765
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 765
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 766
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 766
get_bytestring() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupport static method), 271
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 3
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegion method), 676
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 678
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 680
get_bytestring() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather method), 600
get_bytestring() (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex method), 602
get_bytestring() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source.ReverseIPTagMulticastSource method), 611
get_bytestring() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex method), 613
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 656
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 663
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 780
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegion method), 788
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 796
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 803
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 808
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 811
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 765
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 765
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 766
get_bytestring() (spynaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 766
get_bytestring() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupport static method), 271
get_bytestring() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 3
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegion method), 676
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 678
get_bytestring() (spynaker.pyNN.models.common.recording_utils.RecordingRegionMachineVertex method), 680
get_cast_source() (in module spynaker.pyNN.models.common.recording_utils), 672
get_bytestring() (spinn_messages.abstract_data_element.AbstractDataElement method), 257
get_bytestring() (spinnman.messages.eieio.data_messages.AbstractDataElement method), 262
get_bytestring() (spinnman.messages.eieio.data_messages.key_data_element.KeyDataElement method), 261
get_bytestring() (spinnman.messages.eieio.data_messages.key_payload_data_element.KeyPayloadDataElement method), 261
get_bytestring() (spinnman.messages.eieio.data_messages.KeyDataElement method), 265
get_bytestring() (spinnman.messages.eieio.data_messages.KeyPayloadDataElement method), 265
get_chip_and_core() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker static method), 201
get_chip_and_core() (pacman.utilities.utility_objs.ResourceTracker static method), 206
get_chip_at() (spinn_machine.machine.Machine method), 48
get_chip_at() (spinn_machine.machine.Machine method), 31
get_chip_at() (spinn_machine.virtual_machine.VirtualMachine method), 40
get_chip_at() (spinn_machine.VirtualMachine method), 56

get_chip_info() (spinnman.processes.get_machine_process.GetMachineProcess method), 363
get_chip_info() (spinnman.processes.GetMachineProcess method), 369
get_chip_over_link() (spinn_machine.Machine static method), 48
get_chip_over_link() (spinn_machine.machine.Machine static method), 31
get_chips_on_board() (spinn_machine.Machine method), 48
get_chips_on_board() (spinn_machine.machine.Machine method), 31
get_colour() (in module spynnaker.plot_utils), 838
get_column_offset() (spinnman.model.p2p_table.P2PTable static method), 342
get_column_offset() (spinnman.model.P2PTable static method), 357
get_configuration_parameter_value() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_configuration_parameter_value() (spinn_front_end_common.utilities.database.DatabaseReader method), 559
get_connections() (spinnman.transceiver.Transceiver method), 389
get_connections_from_machine() (spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses method), 666
get_connections_from_machine() (spynnaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses method), 667
get_connections_from_machine() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_connections_from_machine() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_connections_from_machine() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
get_connections_from_machine() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
get_constructed_data_size() (data_specification.data_specification_executor.DataSpecificationExecutor method), 421
get_constructed_data_size() (data_specification.DataSpecificationExecutor method), 459
get_core_state_count() (spinnman.transceiver.Transceiver method), 389
get_core_status_string() (spinnman.transceiver.Transceiver method), 390
get_core_subset_for_chip() (spinn_machine.core_subsets.CoreSubsets method), 26
get_core_subset_for_chip() (spinn_machine.CoreSubsets method), 44
get_cores_and_link_count() (spinn_machine.Machine method), 48
get_cores_and_link_count() (spinn_machine.machine.Machine method), 31
get_cores_and_link_count() (spinn_machine.virtual_machine.VirtualMachine method), 40
get_cores_and_link_count() (spinn_machine.VirtualMachine method), 56
get_cores_for_binary() (spinnman.model.executable_targets.ExecutableTargets method), 340
get_cores_for_binary() (spinnman.model.ExecutableTargets method), 355
get_cores_in_state() (spinnman.transceiver.Transceiver method), 390
get_cores_not_in_state() (spinnman.transceiver.Transceiver method), 390
get_cpu_info() (spinnman.processes.get_cpu_info_process.GetCPUInfoProcess method), 363
get_cpu_info() (spinnman.processes.GetCPUInfoProcess method), 368
get_cpu_information() (spinnman.transceiver.Transceiver method), 390
get_cpu_information_from_core() (spinnman.transceiver.Transceiver method), 391
get_cpu_usage() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather static method), 597
get_cpu_usage() (spinn_front_end_common.utility_models.LivePacketGather static method), 608
get_cpu_usage() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source.ReverseIPTagMulticastSource static method), 602
get_cpu_usage() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSource static method), 613
get_cpu_usage_for_atoms() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_cpu_usage_for_atoms() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_cpu_usage_for_atoms() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson static method), 797
get_cpu_usage_for_atoms() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson static method), 803

get_cpu_usage_for_atoms() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 808
get_cpu_usage_for_atoms() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 811
get_current_time() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
get_current_time() (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface method), 837
get_current_time() (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState method), 829
get_data() (in module spynnaker.pyNN.models.common), 676
get_data() (in module spynnaker.pyNN.models.common.recording_utils), 672
get_data() (spynnaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable method), 669
get_data() (spynnaker.pyNN.models.common.abstract_uint32_recorder.AbstractUInt32Recorder static method), 671
get_data() (spynnaker.pyNN.models.common.AbstractNeuronRecordable method), 673
get_data() (spynnaker.pyNN.models.common.AbstractUInt32Recorder static method), 674
get_data() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_data() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_data_for_vertex() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 525
get_data_for_vertex() (spinn_front_end_common.interface.buffer_management.BufferManager method), 529
get_data_spec_and_file_writer_filename() (in module data_specification.utility_calls), 456
get_data_spec_file_path() (in module data_specification.utility_calls), 457
get_database_connection() (spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor method), 650
get_database_connection() (spynnaker.pyNN.external_devices_models.AbstractEthernetSensor method), 658
get_database_connection() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice method), 629
get_database_connection() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice method), 635
get_dataspec_datatype() (spynnaker.pyNN.models.neural_properties.neural_parameter.NeuronParameter method), 701
get_dataspec_datatype() (spynnaker.pyNN.models.neural_properties.NeuronParameter method), 701
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector method), 678
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 680
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 681
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 691
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 691
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 683
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector method), 692
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector method), 684
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685

get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector method), 686
get_delay_maximum() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
get_delay_maximum() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
get_delay_maximum() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 760
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector method), 678
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 680
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector method), 681
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector method), 691
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 691
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 683
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector method), 692
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector method), 684
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector method), 686
get_delay_variance() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
get_delay_variance() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
get_delay_variance() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 760
get_distribution_to_stats() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 837
get_distribution_to_stats() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 829
get_dtcm_usage() (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGather static method), 587
get_dtcm_usage() (spinn_front_end_common.utility_models.LivePacketGather static method), 608
get_dtcm_usage() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source.ReverseIPTagMulticastSource static method), 602
get_dtcm_usage() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSource static method), 613
get_dtcm_usage_for_atoms() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_dtcm_usage_for_atoms() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_dtcm_usage_for_atoms() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson static method), 797
get_dtcm_usage_for_atoms() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson static method), 803
get_dtcm_usage_for_atoms() (spyn-

naker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 808
get_dtcm_usage_for_atoms() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 811
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.eieio_spike_recorder.EIEIOSpikeRecorder method), 671
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.EIEIOSpikeRecorder method), 675
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.gsyn_inhibitory_recorder.GsynInhibitoryRecorder method), 671
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder method), 672
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.MultiSpikeRecorder method), 676
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder method), 672
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.NeuronRecorder method), 675
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.spike_recorder.SpikeRecorder method), 673
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.common.SpikeRecorder method), 676
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
get_dtcm_usage_in_bytes() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput method), 702
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.AbstractAdditionalInput method), 703
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.AdditionalInputCa2Adaptive method), 703
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Adaptive method), 704
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType method), 724
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 733
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 738
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 767
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 770
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType method), 773
get_dtcm_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.threshold_types.AbstractThresholdType method), 775
get_edge_constraints() (spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract method), 728
get_edge_constraints() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ method), 729
get_edge_constraints() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ method), 730
get_edge_constraints() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterP method), 731
get_edge_constraints() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterP method), 732
get_edge_groups() (in module pacman.utilities.algorithm_utilities.routing_info_allocator_utilities), 196
get_edges_ending_at_vertex() (pacman.model.graphs.abstract_graph.AbstractGraph method), 139
get_edges_ending_at_vertex() (pacman.model.graphs.AbstractGraph method), 144
get_edges_ending_at_vertex() (pacman.model.graphs.application.ApplicationGraph method), 127
get_edges_ending_at_vertex() (pacman.model.graphs.impl.graph.Graph method), 124
get_edges_ending_at_vertex_with_partition_name() (pacman.model.graphs.abstract_graph.AbstractGraph method), 139
get_edges_ending_at_vertex_with_partition_name()

(pacman.model.graphs.AbstractGraph method), 144
get_edges_ending_at_vertex_with_partition_name() (pacman.model.graphs.impl.Graph method), 127
get_edges_ending_at_vertex_with_partition_name() (pacman.model.graphs.impl.graph.Graph method), 124
get_edges_starting_at_vertex() (pacman.model.graphs.abstract_graph.AbstractGraph method), 139
get_edges_starting_at_vertex() (pacman.model.graphs.AbstractGraph method), 144
get_edges_starting_at_vertex() (pacman.model.graphs.impl.Graph method), 127
get_edges_starting_at_vertex() (pacman.model.graphs.impl.graph.Graph method), 124
get_end_buffering_sequence_number() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 509
get_end_buffering_sequence_number() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 517
get_end_buffering_state() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 509
get_end_buffering_state() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 517
get_entries_for_router() (pacman.model.routing_table_by_partition.multicast_routing_table_by_partition.MulticastRoutingTableByPartition method), 171
get_entries_for_router() (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartition method), 172
get_entry_on_coords_for_edge() (pacman.model.routing_table_by_partition.multicast_routing_table_by_partition.MulticastRoutingTableByPartition method), 171
get_entry_on_coords_for_edge() (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartition method), 172
get_ethernet_chip() (in module spinn_front_end_common.utilities.helpful_functions), 580
get_ethernet_chip_coordinates() (spinn_machine.spinnaker_triad_geometry.SpiNNakerTriadGeometry method), 38
get_ethernet_chip_coordinates() (spinn_machine.SpiNNakerTriadGeometry method), 55
get_exact_master_population_table_size() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators.MasterPopTableAsBinarySearch method), 730
get_exact_master_population_table_size() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch method), 732
get_executable_path() (spinn_utilities.executable_finder.ExecutableFinder method), 9
get_exponential_decay_and_init() (in module spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential), 770
get_external_devices() (spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController method), 650
get_external_devices() (spynnaker.pyNN.external_devices_models.AbstractEthernetController method), 658
get_external_devices() (spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex.ExternalDeviceLifControlVertex method), 653
get_external_devices() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControlVertex method), 661
get_first_board() (spinnman.messages.scp.abstract_messages.bmp_request.BMPRequest static method), 269
get_first_board() (spinnman.messages.scp.abstract_messages.BMPRequest static method), 271
get_first_key_for_edge() (pacman.model.routing_info.routing_info.RoutingInfo method), 166
get_first_key_for_edge() (pacman.model.routing_info.RoutingInfo method), 169
get_first_key_from_partition() (pacman.model.routing_info.routing_info.RoutingInfo method), 167
get_first_key_from_partition() (pacman.model.routing_info.RoutingInfo method), 170
get_first_key_from_pre_vertex() (pacman.model.routing_info.routing_info.RoutingInfo method), 167
get_first_key_from_pre_vertex() (pacman.model.routing_info.RoutingInfo method), 170
get_first_none_monitor_processor() (spinn_machine.Chip method), 41
get_first_none_monitor_processor() (spinn_machine.chip.Chip method), 23
get_fixed_mask() (in module pacman.utilities.algorithm_utilities.routing_info_allocator_utilities), 197
get_float() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser

method), 4
get_fpga_link_with_id() (spinn_machine.Machine method), 48
get_fpga_link_with_id() (spinn_machine.machine.Machine method), 31
get_front_end_common_pacman_xml_paths() (in module spinn_front_end_common.utilities.function_list), 579
get_global_parameter_types() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 733
get_global_parameter_types() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronMethod method), 738
get_global_parameter_types() (spynnaker.pyNN.models.neuron.neuron_models.neuronmodelizhikevich.NeuronModelIzh method), 735
get_global_parameter_types() (spynnaker.pyNN.models.neuron.neuron_models.neuronmodelleaky_integrate.NeuronModelLeakyIntegrate method), 736
get_global_parameter_types() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_global_parameter_types() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate method), 741
get_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 733
get_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronMethod method), 738
get_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.neuronmodelizhikevich.NeuronModelIzh method), 735
get_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.neuronmodelleaky_integrate.NeuronModelLeakyIntegrate method), 736
get_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate method), 741
get_global_weight_scale() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType method), 724
get_global_weight_scale() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
get_global_weight_scale() (spynnaker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent method), 725
get_global_weight_scale() (spynnaker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent method), 725
get_global_weight_scale() (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance method), 727
get_global_weight_scale() (spynnaker.pyNN.models.neuron.input_types.InputTypeCurrent method), 727
get_gsyn_inhibitory() (spynnaker.pyNN.models.neuron.recording.gsyn_inhibitory_recorder.GsynInhibitoryRecorder method), 671
get_header() (data_specification.data_specification_executor.DataSpecificationExecutor method), 421
get_header() (data_specification.DataSpecificationExecutor method), 459
get_header_size() (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader static method), 258
get_header_size() (spinnman.messages.eieio.data_messages.EIEIODataHeader static method), 263
get_heap() (spinnman.processes.get_heap_process.GetHeapProcess method), 363
get_heap() (spinnman.processes.GetHeapProcess method), 368
get_heap() (spinnman.transceiver.Transceiver method), 391
get_incoming_partitions() (spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraints.AbstractProvidesIncomingPartitionConstraints method), 496
get_incoming_partition_constraints() (spinn_front_end_common.abstract_models.AbstractProvidesIncomingPartitionConstraints method), 500
get_incoming_partition_constraints() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_incoming_partition_constraints() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_incoming_partition_constraints() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
get_incoming_partition_constraints() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
get_input_type_parameter_names() (spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor method), 650
get_input_type_parameter_names() (spynnaker.pyNN.external_devices_models.AbstractEthernetSensor method), 658
get_input_type_parameter_names() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.InputTypeConductance method), 629

get_injector_label() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetRetinaDevice method), 636
get_injector_parameters() (spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor method), 650
get_injector_parameters() (spynnaker.pyNN.external_devices_models.AbstractEthernetSensor method), 658
get_injector_parameters() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetRetinaDevice method), 629
get_injector_parameters() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetRetinaDevice method), 636
get_input_chips() (spinnman.connections.abstract_classes.multicast_receiver.MulticastReceiver method), 220
get_input_chips() (spinnman.connections.abstract_classes.multicast_sender.MulticastSender method), 221
get_input_chips() (spinnman.connections.abstract_classes.MulticastReceiver method), 226
get_input_chips() (spinnman.connections.abstract_classes.MulticastSender method), 226
get_input_type_parameter_types() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType method), 724
get_input_type_parameter_types() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
get_input_type_parameter_types() (spynnaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance method), 725
get_input_type_parameter_types() (spynnaker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent method), 725
get_input_type_parameter_types() (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance method), 727
get_input_type_parameter_types() (spynnaker.pyNN.models.neuron.input_types.InputTypeCurrent method), 727
get_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType method), 724
get_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
get_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance method), 725
get_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent method), 725
get_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance method), 727
get_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.InputTypeCurrent method), 727
get_inputs_by_name() (pacman.executor.algorithm_decorators.all_of_input.AllOfInput method), 87
get_inputs_by_name() (pacman.executor.algorithm_decorators.AbstractInput method), 91
get_inputs_by_name() (pacman.executor.algorithm_decorators.all_of_input.AllOfInput method), 89
get_inputs_by_name() (pacman.executor.algorithm_decorators.AllOfInput method), 91
get_inputs_by_name() (pacman.executor.algorithm_decorators.one_of_input.OneOfInput method), 89
get_inputs_by_name() (pacman.executor.algorithm_decorators.OneOfInput method), 91
get_inputs_by_name() (pacman.executor.algorithm_decorators.single_input.SingleInput method), 90
get_inputs_by_name() (pacman.executor.algorithm_decorators.SingleInput method), 92
get_int() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConf method), 4
get_iobuf() (spinnman.transceiver.Transceiver method), 391
get_iobuf() (spinnman.transceiver.Transceiver method), 392
get_ip_tag_info() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker static method), 202
get_ip_tag_info() (pacman.utilities.utility_objs.ResourceTracker static method), 206
get_ip_tags_for_vertex() (pacman.model.tags.Tags method), 179
get_ip_tags_for_vertex() (pacman.model.tags.tags.Tags method), 178
get_item() (pacman.executor.pacman_algorithm_executor.PACMANAlgorit method), 96
get_item() (pacman.executor.PACMANAlgorithmExecutor method), 98
get_items() (pacman.executor.pacman_algorithm_executor.PACMANAlgori

method), 96
get_items() (pacman.executor.PACMANAlgorithmExecutor method), 98
get_key_from_coords() (in module spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array), 729
get_key_to_atom_id_mapping() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_key_to_atom_id_mapping() (spinn_front_end_common.utilities.database.DatabaseReader method), 559
get_keys() (pacman.model.routing_info.base_key_and_mask.BaseKeyAndMask method), 164
get_keys() (pacman.model.routing_info.BaseKeyAndMask method), 168
get_keys() (pacman.model.routing_info.partition_routing_info.PartitionRoutingInfo method), 166
get_keys() (pacman.model.routing_info.PartitionRoutingInfo method), 169
get_last_sequence_number() (in module spinn_front_end_common.interface.buffer_management.buffer_utility), 525
get_link() (spinn_machine.Router method), 53
get_link() (spinn_machine.router.Router method), 36
get_live_input_details() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_live_input_details() (spinn_front_end_common.utilities.database.DatabaseReader method), 559
get_live_output_details() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_live_output_details() (spinn_front_end_common.utilities.database.DatabaseReader method), 560
get_local_chip_coordinate() (spinn_machine.spinnaker_triad_geometry.SpiNNakerTriadGeometry method), 38
get_local_chip_coordinate() (spinn_machine.SpiNNakerTriadGeometry method), 55
get_local_provenance_data() (spinn_front_end_common.interface.provenance.abstract_provides_local_provenance_data.AbstractProvidesLocalProvenanceData method), 543
get_local_provenance_data() (spinn_front_end_common.interface.provenance.AbstractProvidesLocalProvenanceData method), 544
get_local_provenance_data() (spynnaker.pyNN.models.neural_projections.projection_machine_edge_instance.ProjectionMachineEdgeInstance method), 698
get_local_provenance_data() (spynnaker.pyNN.models.neural_projections.ProjectionMachineEdge method), 701
get_lut_provenance() (in module spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers), 743
get_machine_details() (spinnman.processes.get_machine_process.GetMachineProcess method), 364
get_machine_details() (spinnman.processes.GetMachineProcess method), 369
get_machine_details() (spinnman.transceiver.Transceiver method), 392
get_machine_dimensions() (spinnman.transceiver.Transceiver method), 393
get_machine_edges() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 120
get_machine_edges() (pacman.model.graphs.common.GraphMapper method), 122
get_machine_info() (spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.FakeHBPPortalMachineProvider method), 829
get_machine_live_input_details() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_machine_live_input_details() (spinn_front_end_common.utilities.database.DatabaseReader method), 560
get_machine_live_input_key() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_machine_live_input_key() (spinn_front_end_common.utilities.database.DatabaseReader method), 560
get_machine_live_output_details() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_machine_live_output_details() (spinn_front_end_common.utilities.database.DatabaseReader method), 560
get_machine_live_output_key() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_machine_live_output_key() (spinn_front_end_common.utilities.database.DatabaseReader method), 560
get_machine_provenance_data() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 120
get_machine_edge_provenance() (pacman.model.graphs.common.GraphMapper method), 122
get_machine_edge_vertices() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 120

get_machine_vertices() (pacman.model.graphs.common.GraphMapper method), 122
get_master_population_table_size() (spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_generator.AbstractMasterPopTableFactory method), 728
get_master_population_table_size() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_array.MasterPopTableAsArray method), 729
get_master_population_table_size() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch method), 730
get_master_population_table_size() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopNoneAddressArray method), 731
get_master_population_table_size() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopNoneAddressBinarySearch method), 732
get_max_atoms_per_core() (pacman.model.abstract_classes.abstract_has_global_max_atoms.AbstractHasGlobalMaxAtoms static method), 98
get_max_atoms_per_core() (pacman.model.abstract_classes.AbstractHasGlobalMaxAtoms static method), 98
get_max_atoms_per_core() (spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex.ExternalDeviceLIFControlVertex static method), 653
get_max_atoms_per_core() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl static method), 661
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_cond_exp_base.IFCondExpBase static method), 708
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc.IFCondExpStoc static method), 709
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_delta.IFCurrDelta static method), 711
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base.IFCurrDualExpBase static method), 712
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.IFCurrExpBase static method), 713
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive_base.IFCurrExpCa2Adaptive static method), 714
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCondExpBase static method), 719
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCondExpStoc static method), 722
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrDelta static method), 723
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrDualExpBase static method), 720
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrExpBase static method), 720
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrExpCa2Adaptive static method), 723
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base.IzkCondExpBase static method), 716
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base.IzkCurrExpBase static method), 717
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IzkCondExpBase static method), 721
get_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IzkCurrExpBase static method), 722
get_max_atoms_per_core() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray static method), 794
get_max_atoms_per_core() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson static method), 797
get_max_atoms_per_core() (spynnaker.pyNN.models.spike_source.SpikeSourceArray static method), 800
get_max_atoms_per_core() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson static method), 803
get_max_buffer_size_possible() (spinn_front_end_common.interface.buffer_management.buffer_m method), 503
get_max_buffer_size_possible() (spinn_front_end_common.interface.buffer_management.buffer_m method), 506
get_max_buffer_size_possible() (spinn_front_end_common.interface.buffer_management.buffer_m method), 504
get_max_buffer_size_possible() (spinn_front_end_common.interface.buffer_management.buffer_m method), 507
get_max_synapses() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_ method), 757
get_max_synapses() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseD method), 760

get_max_synapses() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
get_max_synapses() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 759
get_max_synapses() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 762
get_max_synapses() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 763
get_maximum_constrained_resources_available() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 202
get_maximum_constrained_resources_available() (pacman.utilities.utility_objs.ResourceTracker method), 206
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses method), 666
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses method), 668
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 780
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 788
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO method), 765
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.synapse_io.AbstractSynapseIO method), 766
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 765
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 766
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
get_maximum_delay_supported_in_ms() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
get_maximum_probable_value() (in module spynnaker.pyNN.utilities.utility_calls), 830
get_maximum_resources_available() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 202
get_maximum_resources_available() (pacman.utilities.utility_objs.ResourceTracker method), 206
get_mean() (in module spynnaker.pyNN.utilities.utility_calls), 830
get_mean() (spinn_front_end_common.interface.profiling.profile_data.ProfileData method), 541
get_mean_ms_per_ts() (spinn_front_end_common.interface.profiling.profile_data.ProfileData method), 541
get_mean_n_calls_per_ts() (spinn_front_end_common.interface.profiling.profile_data.ProfileData method), 541
get_message_translator() (spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController method), 650
get_message_translator() (spynnaker.pyNN.external_devices_models.AbstractEthernetController method), 658
get_message_translator() (spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex.ExternalDeviceLifControlVertex method), 653
get_message_translator() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControlVertex method), 661
get_min_packet_length() (spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage static method), 249
get_min_packet_length() (spinnman.messages.eieio.command_messages.EIEIOCommandMessage static method), 254
get_min_packet_length() (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead static method), 250
get_min_packet_length() (spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData static method), 251
get_min_packet_length() (spinnman.messages.eieio.command_messages.HostDataRead static method), 255
get_min_packet_length() (spinnman.messages.eieio.command_messages.HostSendSequencedData static method), 255
get_min_packet_length() (spinnman.messages.eieio.command_messages.padding_request.PaddingRequest static method), 251
get_min_packet_length() (spinnman.messages.eieio.command_messages.PaddingRequest static method), 255
get_min_packet_length() (spinnman.messages.eieio.command_messages.spinnaker_request_buffer.SpinnakerRequestBuffer static method), 252
get_min_packet_length() (spinnman.messages.eieio.command_messages.spinnaker_request_read_buffer.SpinnakerRequestReadBuffer static method), 252
get_min_packet_length() (spinn-

man.messages.eieio.command_messages.SpinnakerRequestBuffers static method), 256
get_min_packet_length() (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData static method), 257
get_min_packet_length() (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage method), 260
get_min_packet_length() (spinnman.messages.eieio.data_messages.EIEIODataMessage method), 264
get_minimum_buffer_sdram() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 526
get_minimum_buffer_sdram_usage() (spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost method), 502
get_minimum_buffer_sdram_usage() (spinn_front_end_common.interface.buffer_management.recording_utilities.AbstractRecordable method), 505
get_minimum_buffer_sdram_usage() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex method), 586
get_minimum_buffer_sdram_usage() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source.ReverseIPTagMulticastSourceMachineVertex method), 602
get_minimum_buffer_sdram_usage() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex method), 613
get_minimum_buffer_sdram_usage() (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex method), 784
get_minimum_buffer_sdram_usage() (spynnaker.pyNN.models.neuron.PopulationMachineVertex method), 792
get_minimum_buffer_sdram_usage() (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex method), 799
get_minimum_buffer_sdram_usage() (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVertex method), 805
get_minimum_probable_value() (in module spynnaker.pyNN.utilities.utility_calls), 830
get_missing_info_for_region() (spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost method), 516
get_missing_info_for_region() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedTableWithSizeAndAddress method), 524
get_mpt_sb_mem_addrs_from_coords() (in module spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search), 729
get_multicast_routes() (spinnman.transceiver.Transceiver Buffers), 393
get_multicast_routing_entry_by_routing_entry_key() (pacman.model.routing_tables.multicast_routing_table.MulticastRoutingTable method), 174
get_multicast_routing_entry_by_routing_entry_key() (pacman.model.routing_tables.MulticastRoutingTable method), 176
get_munich_d() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 822
get_munich_f() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 822
get_munich_i() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 822
get_n_atoms() (spinn_front_end_common.utilities.database.database_reader.DatabaseReader method), 556
get_n_atoms() (spinn_front_end_common.utilities.database.DatabaseReader method), 560
get_n_bytes() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedTableWithSizeAndAddress method), 513
get_n_bytes() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedTableWithSizeAndAddress method), 521
get_n_bytes_per_connection() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure method), 743
get_n_bytes_per_connection() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.AbstractSynapseStructure method), 744
get_n_bytes_per_connection() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseStructure method), 744
get_n_bytes_per_connection() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseStructure method), 744
get_n_bytes_per_connection() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseStructure method), 744
get_n_bytes_per_connection() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseStructure method), 744
get_n_bytes_written() (spinn_storage_handlers.abstract_classes.abstract_byte_writer.AbstractByteWriter method), 62
get_n_bytes_written() (spinn_storage_handlers.ByteWriter method), 67
get_n_calls() (spinn_front_end_common.interface.profiling.profile_data.ProfileData method), 542
get_n_column_bytes() (spinnman.model.p2p_table.P2PTable static method), 343
get_n_column_bytes() (spinnman.model.p2p_table.P2PTable static method), 357
get_n_command_bytes() (spinn_front_end_common.utility_models.comma

static method), 590
get_n_command_bytes() (spinn_front_end_common.utility_models.CommandSendingMachineVertex static method), 605
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector method), 679
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 680
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector method), 681
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector method), 691
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 691
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 683
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector method), 692
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector method), 684
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector method), 686
get_n_connections_from_pre_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 679
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 680
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector method), 681
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector method), 691
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 691
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 683
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector method), 692
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector method), 684
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 684

get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector method), 686
get_n_connections_to_post_vertex_maximum() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
get_n_cpu_cycles() (spynnaker.pyNN.models.common.eieio_spike_recorder.EIEIOSpikeRecorder method), 671
get_n_cpu_cycles() (spynnaker.pyNN.models.common.EIEIOSpikeRecorder method), 675
get_n_cpu_cycles() (spynnaker.pyNN.models.common.gsyn_inhibitory_recorder.GsynInhibitoryRecorder method), 671
get_n_cpu_cycles() (spynnaker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder method), 672
get_n_cpu_cycles() (spynnaker.pyNN.models.common.MultiSpikeRecorder method), 676
get_n_cpu_cycles() (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder method), 672
get_n_cpu_cycles() (spynnaker.pyNN.models.common.NeuronRecorder method), 675
get_n_cpu_cycles() (spynnaker.pyNN.models.common.spike_recorder.SpikeRecorder method), 673
get_n_cpu_cycles() (spynnaker.pyNN.models.common.SpikeRecorder method), 676
get_n_cpu_cycles() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
get_n_cpu_cycles() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.external_devices_models.threshold_type_multicast_device.ThresholdTypeMulticastDevice method), 658
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.external_devices_models.ThresholdTypeMulticastDevice method), 665
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_ method), 702
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.additional_inputs.AbstractAdditional method), 703
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca method), 703
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2 method), 704
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.Abst method), 724
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.input_types.input_type_conductance. method), 725
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.input_types.input_type_current.InputT method), 725
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance method), 727
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.input_types.InputTypeCurrent method), 727
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_mod method), 733
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronMode method), 738
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.N method), 735
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_ method), 736
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_ method), 738
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyI method), 741
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyI method), 742

get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 767
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 770
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta method), 768
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential method), 769
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential method), 769
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 773
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential method), 772
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential method), 772
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType method), 774
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.threshold_types.AbstractThresholdType method), 776
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic.ThresholdTypeMaassStochastic method), 774
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static.ThresholdTypeStatic method), 775
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic method), 777
get_n_cpu_cycles_per_neuron() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic method), 776
get_n_fixed_plastic_words_per_row() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics method), 756
get_n_fixed_plastic_words_per_row() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics method), 762
get_n_fixed_plastic_words_per_row() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 759
get_n_fixed_plastic_words_per_row() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_n_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 734
get_n_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 739
get_n_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh method), 735
get_n_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_exponential.NeuronModelLeakyIntegrateAndFireExponential method), 736
get_n_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_n_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFireExponential method), 741
get_n_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType method), 724
get_n_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
get_n_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance method), 725
get_n_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent method), 725
get_n_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance method), 727
get_n_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.InputTypeCurrent method), 727
get_n_items() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics method), 757
get_n_items() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics method), 761
get_n_keys() (spinn_front_end_common.interface.buffer_management.storage_control_model.AbstractProvidesNKeys method), 513
get_n_keys() (spinn_front_end_common.interface.buffer_management.storage_control_model.AbstractProvidesNKeys method), 521
get_n_keys_for_partition() (spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_partition.AbstractProvidesNKeys method), 497
get_n_keys_for_partition() (spinn_front_end_common.abstract_models.AbstractProvidesNKeys method), 500
get_n_keys_for_partition() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionSTDP method), 398
get_n_keys_for_partition() (spyn-

naker.pyNN.models.utility_models.DelayExtensionVertex method), 811
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 734
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 739
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_izh_cerebellum.NeuronModelIzhCerebellum method), 735
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAndFire method), 737
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire_mars.NeuronModelLeakyIntegrateAndFireMars method), 738
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate method), 741
get_n_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire method), 742
get_n_neurons() (spynaker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor method), 650
get_n_neurons() (spynaker.pyNN.external_devices_models.AbstractEthernetSensor method), 659
get_n_neurons() (spynaker.pyNN.external_devices_models.external_spinnaker_link_fpga.ExternalFPGARetinaDevice static method), 654
get_n_neurons() (spynaker.pyNN.external_devices_models.ExternalFPGARetinaDevice static method), 662
get_n_neurons() (spynaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetRetinaDevice method), 629
get_n_neurons() (spynaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetRetinaDevice method), 636
get_n_parameters() (spynaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput method), 702
get_n_parameters() (spynaker.pyNN.models.neuron.additional_inputs.AbstractAdditionalInput method), 703
get_n_parameters() (spynaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.AdditionalInputCa2Adaptive method), 703
get_n_parameters() (spynaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2 method), 704
get_n_plastic_plastic_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics method), 756
get_n_plastic_plastic_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics method), 762
get_n_plastic_plastic_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics method), 759
get_n_plastic_plastic_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 764
get_n_static_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamics method), 756
get_n_static_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamics method), 761
get_n_static_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics method), 758
get_n_static_words_per_row() (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 762
get_n_synapse_type_bits() (spynaker.pyNN.models.neuron.synapse_types.abstract_synapse_type method), 767
get_n_synapse_type_bits() (spynaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 770
get_n_synapse_type_parameters() (spynaker.pyNN.models.neuron.synapse_types.abstract_synapse_type method), 767
get_n_synapse_type_parameters() (spynaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 770
get_n_synapse_type_parameters() (spynaker.pyNN.models.neuron.synapse_types.PushBotEthernetRetinaSynapseType method), 768
get_n_synapse_type_parameters() (spynaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential method), 769
get_n_synapse_type_parameters() (spynaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential method), 770
get_n_synapse_type_parameters() (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 773
get_n_synapse_type_parameters() (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential method), 772
get_n_synapse_type_parameters() (spyn-

naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential (spynnaker.pyNN.external_devices_models.ThresholdTypeMulticastDe
method), 772 method), 665
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 767 get_n_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_ty method), 774
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 771 get_n_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.AbstractThresholdTy method), 776
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta method), 768 get_n_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maas method), 774
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential method), 769 get_n_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static method), 775
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential method), 770 get_n_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass method), 777
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 773 get_n_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic method), 776
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential method), 772 get_n_timesteps_in_buffer_space() (in module spinn_front_end_common.interface.buffer_management.recordin 526
get_n_synapse_types() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential method), 772 get_n_timesteps_in_buffer_space() (spinn_front_end_common.interface.buffer_management.buffer_m method), 502
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics method), 756 get_n_timesteps_in_buffer_space() (spinn_front_end_common.interface.buffer_management.buffer_m method), 505
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamics method), 756 get_n_timesteps_in_buffer_space() (spinn_front_end_common.interface.buffer_management.recording_monitor_m method), 586
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics method), 762 get_n_timesteps_in_buffer_space() (spinn_front_end_common.utility_models.reverse_ip_tag_multica method), 602
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamics method), 761 get_n_timesteps_in_buffer_space() (spinn_front_end_common.utility_models.ReverseIPTagMulticas method), 613
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758 get_n_timesteps_in_buffer_space() (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulatioMachineVertex method), 784
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 759 get_n_timesteps_in_buffer_space() (spynnaker.pyNN.models.neuron.PopulationMachineVertex method), 792
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 762 get_n_timesteps_in_buffer_space() (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine method), 799
get_n_synapses_in_rows() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764 get_n_timesteps_in_buffer_space() (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVe method), 805
get_n_threshold_parameters() (spynnaker.pyNN.external_devices_models.threshold_type_multicast_delay.ThresholdTypeMulticastDelay method), 658 get_n_words_for_plastic_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_s method), 756
get_n_threshold_parameters() (spyn- get_n_words_for_plastic_connections() (spyn-

naker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics method), 762
get_n_words_for_plastic_connections() (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 759
get_n_words_for_plastic_connections() (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_n_words_for_static_connections() (spynaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics method), 756
get_n_words_for_static_connections() (spynaker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamics method), 761
get_n_words_for_static_connections() (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 758
get_n_words_for_static_connections() (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 762
get_neighbouring_chips_coords() (spinn_machine.Router method), 54
get_neighbouring_chips_coords() (spinn_machine.router.Router method), 37
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 734
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 739
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh method), 735
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate method), 737
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAndFire method), 738
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate method), 741
get_neural_parameter_types() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire method), 742
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 734
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 739
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh method), 735
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate method), 737
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAndFire method), 738
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate method), 741
get_neural_parameters() (spynaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire method), 742
get_new() (spinnman.utilities.appid_tracker.AppIdTracker method), 379
get_next_allowed_address() (spynaker.pyNN.models.neuron.master_pop_table_generators.master_ method), 729
get_next_allowed_address() (spynaker.pyNN.models.neuron.master_pop_table_generators.master_ method), 730
get_next_allowed_address() (spynaker.pyNN.models.neuron.master_pop_table_generators.MasterP method), 731
get_next_allowed_address() (spynaker.pyNN.models.neuron.master_pop_table_generators.MasterP method), 732
get_next_connection() (spinnman.processes.abstract_multi_connection_process_connection_se method), 362
get_next_connection() (spinnman.processes.most_direct_connection_selector.MostDirectConn method), 365
get_next_connection() (spinnman.processes.MostDirectConnectionSelector method), 370
get_next_connection() (spinnman.processes.round_robin_connection_selector.RoundRobinCon method), 367
get_next_connection() (spinnman.processes.RoundRobinConnectionSelector method), 370
get_next_key() (spinn_front_end_common.interface.buffer_management.bu method), 503
get_next_key() (spinn_front_end_common.interface.buffer_management.bu method), 506
get_next_key() (spinn_front_end_common.interface.buffer_management.bu method), 504
get_next_key() (spinn_front_end_common.interface.buffer_management.bu

method), 507
get_next_timestamp() (spinn_front_end_common.interface.buffer_management.buffer_models.abstract_sends_buffers_from_host.Abstr
method), 503
get_next_timestamp() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractSendsBuffersFromHost
method), 506
get_next_timestamp() (spinn_front_end_common.interface.buffer_management.buffer_models.sends_buffers_from_host_pre_buffered_
method), 504
get_next_timestamp() (spinn_front_end_common.interface.buffer_management.buffer_models.SendsBuffersFromHostPreBufferedImpl
method), 507
get_nid_from_key() (in module spyn-
naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array),
729
get_not_running_simulator() (in module
spinn_front_end_common.utilities.globals_variables),
579
get_number_of_available_cores_on_machine
(spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase
attribute), 548
get_number_of_mallocs_used_by_dsg()
(spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex
static method), 590
get_number_of_mallocs_used_by_dsg()
(spinn_front_end_common.utility_models.CommandSenderMachineVertex
static method), 605
get_number_of_mallocs_used_by_dsg() (spyn-
naker.pyNN.models.neuron.synaptic_manager.SynapticManager
method), 785
get_number_of_mallocs_used_by_dsg() (spyn-
naker.pyNN.models.neuron.SynapticManager
method), 791
get_outgoing_edge_partition_starting_at_vertex() (pac-
man.model.graphs.abstract_graph.AbstractGraph
method), 139
get_outgoing_edge_partition_starting_at_vertex() (pac-
man.model.graphs.AbstractGraph method),
144
get_outgoing_edge_partition_starting_at_vertex() (pac-
man.model.graphs.impl.Graph method), 128
get_outgoing_edge_partition_starting_at_vertex() (pac-
man.model.graphs.impl.graph.Graph method),
124
get_outgoing_edge_partitions_starting_at_vertex() (pac-
man.model.graphs.abstract_graph.AbstractGraph
method), 139
get_outgoing_edge_partitions_starting_at_vertex() (pac-
man.model.graphs.AbstractGraph method),
145
get_outgoing_edge_partitions_starting_at_vertex() (pac-
man.model.graphs.impl.Graph method), 128
get_outgoing_edge_partitions_starting_at_vertex() (pac-
man.model.graphs.impl.graph.Graph method),
124
get_outgoing_partition_constraints()
(spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartition
method), 497
get_outgoing_partition_constraints()
(spinn_front_end_common.abstract_models.AbstractProvidesOut
method), 501
get_outgoing_partition_constraints()
(spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex
method), 588
get_outgoing_partition_constraints()
(spinn_front_end_common.utility_models.CommandSender
method), 604
get_outgoing_partition_constraints()
(spinn_front_end_common.utility_models.reverse_ip_tag_multi_
method), 600
get_outgoing_partition_constraints()
(spinn_front_end_common.utility_models.reverse_ip_tag_multica
method), 602
get_outgoing_partition_constraints()
(spinn_front_end_common.utility_models.ReverseIpTagMultiCas
method), 611
get_outgoing_partition_constraints()
(spinn_front_end_common.utility_models.ReverseIPTagMulticas
method), 613
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.external_device_lif_contro
method), 653
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.external_spinnaker_link_fp
method), 654
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.ExternalDeviceLifControl
method), 661
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.ExternalFPGARetinaDevic
method), 662
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.munich_spinnaker_link_m
method), 656
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.munich_spinnaker_link_re
method), 657
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.MunichMotorDevice
method), 663
get_outgoing_partition_constraints() (spyn-
naker.pyNN.external_devices_models.MunichRetinaDevice
method), 665
get_outgoing_partition_constraints() (spyn-
naker.pyNN.models.neuron.abstract_population_vertex.AbstractP
method), 780
get_outgoing_partition_constraints() (spyn-
naker.pyNN.models.neuron.AbstractPopulationVertex
method), 788
get_outgoing_partition_constraints() (spyn-
naker.pyNN.models.spike_source.abstract_provides_outgoing_partition_constraints.AbstractProvidesOutgoingPartition

method), 797
get_outgoing_partition_constraints() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 803
get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 808
get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 811
get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector method), 809
get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.SpikeInjector method), 813
get_outgoing_partition_ids() (spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController method), 650
get_outgoing_partition_ids() (spynnaker.pyNN.external_devices_models.AbstractEthernetController method), 658
get_outgoing_partition_ids() (spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl method), 653
get_outgoing_partition_ids() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl method), 661
get_p_from_key() (in module spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_array), 729
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence method), 745
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence method), 748
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_spike_triplet_timing_dependence.PfisterSpikeTripletTimingDependence method), 745
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.recurrent_timing_dependence.AbstractRecurrentTimingDependence method), 746
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.spike_nearest_pair_timing_dependence.SpikeNearestPairTimingDependence method), 747
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.spike_pair_timing_dependence.SpikePairTimingDependence method), 747
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.vogels_2011_timing_dependence.Vogels2011TimingDependence method), 748
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_spike_triplet_timing_dependence.PfisterSpikeTripletTimingDependence method), 749
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependence method), 750
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependence method), 750
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependence method), 749
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependence method), 751
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence method), 751
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence method), 754
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence.AdditiveTwoPairWeightDependence method), 752
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence.MultiplicativeTwoPairWeightDependence method), 753
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence.AdditiveTwoPairWeightDependence method), 753
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependence method), 754
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence method), 755
get_parameter_names() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependence method), 754
get_parameter_names() (spynnaker.pyNN.models.neuron.synapse_dynamics.pfister_spike_triplet_timing_dependence.PfisterSpikeTripletTimingDependenceSynapse method), 757
get_parameter_names() (spynnaker.pyNN.models.neuron.synapse_dynamics.recurrent_timing_dependence.AbstractRecurrentTimingDependenceSynapse method), 761
get_parameter_names() (spynnaker.pyNN.models.neuron.synapse_dynamics.spike_nearest_pair_timing_dependence.SpikeNearestPairTimingDependenceSynapse method), 758
get_parameter_names() (spynnaker.pyNN.models.neuron.synapse_dynamics.spike_pair_timing_dependence.SpikePairTimingDependenceSynapseDynamics method), 759
get_parameter_names() (spynnaker.pyNN.models.neuron.synapse_dynamics.vogels_2011_timing_dependence.Vogels2011TimingDependenceSynapseDynamics method), 762
get_parameter_names() (spynnaker.pyNN.models.neuron.synapse_dynamics.pfister_spike_triplet_timing_dependence.PfisterSpikeTripletTimingDependenceSynapseDynamics

method), 764
get_parameter_types() (spynaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput method), 702
get_parameter_types() (spynaker.pyNN.models.neuron.additional_inputs.AbstractAdditionalInput method), 703
get_parameter_types() (spynaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.AdditionalInputCa2Adaptive method), 703
get_parameter_types() (spynaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Adaptive method), 704
get_parameters() (spynaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput method), 702
get_parameters() (spynaker.pyNN.models.neuron.additional_inputs.AbstractAdditionalInput method), 704
get_parameters() (spynaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.AdditionalInputCa2Adaptive method), 703
get_parameters() (spynaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Adaptive method), 704
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence method), 745
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence method), 748
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_spike_triplet.PfisterSpikeTriplet method), 745
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.recurrent_timing_dependence.RecurrentTimingDependence method), 746
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.spike_nearest_pair_timing_dependence.SpikeNearestPair method), 747
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.spike_pair_timing_dependence.SpikePair method), 747
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.vogels_2011_timing_dependence.Vogels2011 method), 748
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_spike_triplet method), 749
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.recurrent_timing_dependence.RecurrentTimingDependence method), 750
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependence method), 750
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependence method), 749
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependence method), 751
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence method), 751
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence method), 754
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence.WeightDependence method), 752
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependence method), 753
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence.WeightDependence method), 753
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependence method), 754
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence method), 755
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence method), 755
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_spike_triplet.PfisterSpikeTriplet method), 757
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.recurrent_timing_dependence.RecurrentTimingDependence method), 761
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.spike_nearest_pair_timing_dependence.SpikeNearestPair method), 758
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.spike_pair_timing_dependence.SpikePair method), 759
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.vogels_2011_timing_dependence.Vogels2011 method), 763
get_parameters_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_spike_triplet method), 764
get_parameters_size_in_bytes() (in module spynaker.pyNN.utilities.utility_calls), 830
get_params_bytes() (spynaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 707

get_params_bytes() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson static method), 803
get_placed_processors() (pacman.model.placements.Placements method), 151
get_placed_processors() (pacman.model.placements.placements.Placements method), 149
get_placement_of_vertex() (pacman.model.placements.Placements method), 151
get_placement_of_vertex() (pacman.model.placements.placements.Placements method), 149
get_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics method), 756
get_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics method), 762
get_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamicsSTDP method), 759
get_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_pointer_table() (data_specification.data_specification_executor.DataSpecificationExecutor method), 421
get_pointer_table() (data_specification.DataSpecificationExecutor method), 459
get_probability_within_range() (in module spynnaker.pyNN.utilities.utility_calls), 830
get_probable_maximum_selected() (in module spynnaker.pyNN.utilities.utility_calls), 831
get_processor_with_id() (spinn_machine.Chip method), 41
get_processor_with_id() (spinn_machine.chip.Chip method), 23
get_profile_data() (spinn_front_end_common.interface.profiling.abstract_has_profile_data.AbstractHasProfileData method), 541
get_profile_data() (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex method), 784
get_profile_data() (spynnaker.pyNN.models.neuron.PopulationMachineVertex method), 792
get_profile_region_size() (in module spinn_front_end_common.interface.profiling.profile_utils), 542
get_profiling_data() (in module spinn_front_end_common.interface.profiling.profile_utils), 542
get_provenance_data() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
get_provenance_data() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence method), 745
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence method), 748
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_pair_rule.TimingDependencePairRule method), 746
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent.TimingDependenceRecurrent method), 747
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels_2011.TimingDependenceVogels2011 method), 747
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSTDP method), 749
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSTDP method), 750
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSTDP method), 749
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence method), 751
get_provenance_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence method), 754
get_provenance_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
get_provenance_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 761
get_provenance_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamicsSTDP method), 759
get_provenance_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_provenance_data_from_machine() (spinn_front_end_common.interface.provenance.abstract_provides_provenance_data_from_machine.AbstractProvidesProvenanceDataFromMachine method), 543
get_provenance_data_from_machine() (spinn_front_end_common.interface.provenance.AbstractProvidesProvenanceDataFromMachine method), 545
get_provenance_data_from_machine()

(spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFr
method), 544
get_provenance_data_from_machine()
(spinn_front_end_common.interface.provenance.ProvidesProvenanceDataFromMachineImpl
method), 545
get_provenance_data_from_machine()
(spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex
method), 597
get_provenance_data_from_machine()
(spinn_front_end_common.utility_models.LivePacketGatherMachineVertex
method), 608
get_provenance_data_from_machine()
(spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceM
method), 602
get_provenance_data_from_machine()
(spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex
method), 613
get_provenance_data_from_machine() (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex
method), 784
get_provenance_data_from_machine() (spynnaker.pyNN.models.neuron.PopulationMachineVertex
method), 793
get_provenance_data_from_machine() (spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex.DelayExtensionMachineVertex
method), 807
get_provenance_data_from_machine() (spynnaker.pyNN.models.utility_models.DelayExtensionMachineVertex
method), 810
get_provenance_data_size()
(spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFr
static method), 544
get_provenance_data_size()
(spinn_front_end_common.interface.provenance.ProvidesProvenanceDataFromMachineImpl
static method), 545
get_push_bot_laser_led_speaker_frequency_i()
(in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol),
822
get_push_bot_motor_i() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol),
822
get_pushbot_wifi_connection() (in module spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_protocol),
637
get_pushbot_wifi_connection() (in module spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_protocol),
632
get_pynn_NumpyRNG() (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface
method), 837
get_pynn_NumpyRNG() (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState
method), 829
GET_RANDOM_NUMBER
(data_specification.enums.Commands attribute), 418
GET_RANDOM_NUMBER
(data_specification.enums.commands.Commands attribute), 415
get_reader() (spynnaker.pyNN.models.neural_projections.connectors.from_
method), 683
get_reader() (spynnaker.pyNN.models.neural_projections.connectors.FromF
method), 692
get_receive_method() (spinnman.connections.abstract_classes.Listenable
method), 236
get_receive_method() (spinnman.connections.abstract_classes.listenable.Listenable
method), 239
get_receive_method() (spinnman.connections.udp_packet_connections.eieio_connection.EIEI
method), 238
get_receive_method() (spinnman.connections.udp_packet_connections.EIEIOConnection
method), 240
get_receive_method() (spinnman.connections.udp_packet_connections.sdp_connection.SDPC
method), 241
get_receive_method() (spinnman.connections.udp_packet_connections.SDPConnection
method), 242
get_receive_method() (spinnman.connections.udp_packet_connections.udp_listenable_connec
method), 236
get_receive_method() (spinnman.connections.udp_packet_connections.UDPListenableConnec
method), 241
get_receive_method() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_etherne
method), 631
get_receive_method() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_etherne
method), 638
get_recordable_variables() (spynnaker.pyNN.models.common.abstract_neuron_recordable.Abstrac
method), 669
get_recordable_variables() (spynnaker.pyNN.models.common.AbstractNeuronRecordable
method), 674
get_recordable_variables() (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder
method), 672
get_recordable_variables() (spynnaker.pyNN.models.common.NeuronRecorder
method), 675
get_recorded_state_variables() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP

method), 781
get_recordable_variables() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
get_recorded_data() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex method), 587
get_recorded_region_ids() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 526
get_recorded_region_ids() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractReceiveBuffersToHost method), 502
get_recorded_region_ids() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractReceiveBuffersToHostMachineVertex method), 505
get_recorded_region_ids() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex method), 587
get_recorded_region_ids() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex method), 602
get_recorded_region_ids() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex method), 613
get_recorded_region_ids() (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex method), 784
get_recorded_region_ids() (spynnaker.pyNN.models.neuron.PopulationMachineVertex method), 793
get_recorded_region_ids() (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex method), 799
get_recorded_region_ids() (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVertex method), 805
get_recorded_region_sizes() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 526
get_recording_data_size() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 526
get_recording_header_array() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 527
get_recording_header_size() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 527
get_recording_region_base_address() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractReceiveBuffersToHost method), 502
get_recording_region_base_address() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractReceiveBuffersToHostMachineVertex method), 506
get_recording_region_base_address() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex method), 587
get_recording_region_base_address() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex method), 603
get_recording_region_base_address() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex method), 613
get_recording_region_base_address() (spynnaker.pyNN.models.neuron.buffer_models.AbstractReceiveBuffersToHost method), 785
get_recording_region_base_address() (spynnaker.pyNN.models.neuron.buffer_models.AbstractReceiveBuffersToHostMachineVertex method), 793
get_recording_region_base_address() (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex method), 799
get_recording_region_base_address() (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVertex method), 806
get_recording_region_size_in_bytes() (in module spynnaker.pyNN.models.common.recording_utils), 676
get_recording_region_size_in_bytes() (in module spynnaker.pyNN.models.common.recording_utils), 672
get_recording_region_sizes() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 527
get_recording_resources() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 528
get_region() (data_specification.data_specification_executor.DataSpecificationExecutor method), 421
get_region() (data_specification.DataSpecificationExecutor method), 459
get_region_base_address_offset() (in module data_specification.utility_calls), 457
get_region_buffer_size() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractReceiveBuffersToHost method), 503
get_region_buffer_size() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractReceiveBuffersToHostMachineVertex method), 506
get_region_buffer_size() (spinn_front_end_common.interface.buffer_management.recording_utilities), 505
get_region_buffer_size() (spinn_front_end_common.interface.buffer_management.recording_utilities), 507
get_region_data() (spinn_front_end_common.interface.buffer_management.recording_utilities), 509
get_region_data() (spinn_front_end_common.interface.buffer_management.buffer_models.abstract_receive_buffers_to_host.AbstractReceiveBuffersToHost method), 517
get_region_data_pointer() (spinn_front_end_common.interface.buffer_management.storage_objects.AbstractReceiveBuffersToHost method), 529
get_region_data_pointer()

(spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 517
get_region_pointer() (in module spinn_front_end_common.interface.buffer_management.recording_utilities), 528
get_regions() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractSendsBuffersFromHost method), 503
get_regions() (spinn_front_end_common.interface.buffer_management.buffer_models.AbstractSendsBuffersFromHostPreBufferedImpl method), 506
get_regions() (spinn_front_end_common.interface.buffer_management.buffer_models.SendsBuffersFromHostPreBufferedImpl method), 505
get_regions() (spinn_front_end_common.interface.buffer_management.buffer_models.SendsBuffersFromHostPreBufferedMultiCastPartitionableVertex method), 508
get_reinjection_status() (spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.read_status_process.ReadStatusProcess method), 570
get_reinjection_status() (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex method), 592
get_reinjection_status_for_core_subsets() (spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.read_status_process.ReadStatusProcess method), 570
get_reinjection_status_for_vertices() (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex method), 592
get_remaining_constraints() (in module pacman.utilities.algorithm_utilities.partition_algorithm_utilities), 196
get_resources() (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex static method), 587
get_resources_used_by_atoms() (pacman.model.graphs.application.application_fpga_vertex.ApplicationFPGAVertex method), 111
get_resources_used_by_atoms() (pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex method), 112
get_resources_used_by_atoms() (pacman.model.graphs.application.application_vertex.ApplicationVertex method), 113
get_resources_used_by_atoms() (pacman.model.graphs.application.ApplicationFPGAVertex method), 115
get_resources_used_by_atoms() (pacman.model.graphs.application.ApplicationSpiNNakerLinkVertex method), 118
get_resources_used_by_atoms() (pacman.model.graphs.application.ApplicationVertex method), 117
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.chip_power_monitor_vertex.ChipPowerMonitorVertex method), 585
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex method), 589
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.CommandSender method), 604
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.extra_monitor_support_utilities), 595
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.extra_monitor_sends_buffers_from_host_pre_buffered_impl.ExtraMonitorSendsBuffersFromHostPreBufferedImpl method), 595
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.LivePacketGather method), 607
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.sends_buffers_from_host_pre_buffered_impl.SendsBuffersFromHostPreBufferedImpl method), 601
get_resources_used_by_atoms() (spinn_front_end_common.utility_models.ReverseIpTagMultiCastVertex method), 613
get_resources_used_by_atoms() (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_models.MunichMotorDevice method), 656
get_resources_used_by_atoms() (spynnaker.pyNN.external_devices_models.MunichMotorDevice method), 661
get_resources_used_by_atoms() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 781
get_resources_used_by_atoms() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
get_resources_used_by_atoms() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 797
get_resources_used_by_atoms() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 803
get_resources_used_by_atoms() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 808
get_resources_used_by_atoms() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 812
get_retina_i() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 822
GET_RETINA_KEY_VALUE() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 818
GET_RETINA_PAYLOAD_VALUE() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 818
get_reverse_ip_tags_for_vertex() (pacman.model.tags.Tags method), 179
get_reverse_ip_tags_for_vertex() (pacman.model.tags.tags.Tags method), 178
get_route() (spinnman.model.p2p_table.P2PTable

method), 343
get_route() (spinnman.model.P2PTable method), 357
get_router_diagnostic_filter() (spinnman.transceiver.Transceiver method), 393
get_router_diagnostics() (spinnman.processes.read_router_diagnostics_process.ReadRouterDiagnosticsProcess method), 366
get_router_diagnostics() (spinnman.processes.ReadRouterDiagnosticsProcess method), 370
get_router_diagnostics() (spinnman.transceiver.Transceiver method), 394
get_routers() (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartition method), 171
get_routers() (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartition method), 172
get_routes() (spinnman.processes.get_routes_process.GetMultiCastRoutesProcess method), 364
get_routes() (spinnman.processes.GetMultiCastRoutesProcess method), 369
get_routing_info_for_edge() (pacman.model.routing_info.routing_info.RoutingInfo method), 167
get_routing_info_for_edge() (pacman.model.routing_info.RoutingInfo method), 170
get_routing_info_from_partition() (pacman.model.routing_info.routing_info.RoutingInfo method), 167
get_routing_info_from_partition() (pacman.model.routing_info.RoutingInfo method), 170
get_routing_info_from_pre_vertex() (pacman.model.routing_info.routing_info.RoutingInfo method), 167
get_routing_info_from_pre_vertex() (pacman.model.routing_info.RoutingInfo method), 170
get_routing_table_for_chip() (pacman.model.routing_tables.multicast_routing_tables.MulticastRoutingTables method), 175
get_routing_table_for_chip() (pacman.model.routing_tables.MulticastRoutingTables method), 177
get_same_chip_vertex_groups() (in module pacman.utilities.algorithm_utilities.placer_algorithm_utilities), 196
get_same_size_vertex_groups() (in module pacman.utilities.algorithm_utilities.partition_algorithm_utilities), 196
get_scamp_version() (spinnman.transceiver.Transceiver method), 394
get_scp_data() (spinnman.connections.abstract_classes.scp_sender.SCPSender method), 222
get_scp_data() (spinnman.connections.abstract_classes.SCPSender method), 228
get_scp_data() (spinnman.connections.udp_packet_connections.bmp_conne method), 231
get_scp_data() (spinnman.connections.udp_packet_connections.BMPConne method), 235
get_scp_data() (spinnman.connections.udp_packet_connections.scamp_con method), 233
get_scp_data() (spinnman.connections.udp_packet_connections.SCAMPCo method), 241
get_scp_response() (spinn_front_end_common.utilities.scp.scp_clear_iobuf method), 566
get_scp_response() (spinn_front_end_common.utilities.scp.scp_update_run method), 566
get_scp_response() (spinn_front_end_common.utilities.scp.SCPClearIOBU method), 567
get_scp_response() (spinn_front_end_common.utilities.scp.SCPUpdateRun method), 567
get_scp_response() (spinn_front_end_common.utilities.utility_objs.extra_m method), 568
get_scp_response() (spinn_front_end_common.utilities.utility_objs.extra_m method), 568
get_scp_response() (spinn_front_end_common.utilities.utility_objs.extra_m method), 569
get_scp_response() (spinn_front_end_common.utilities.utility_objs.extra_m method), 569
get_scp_response() (spinn_front_end_common.utilities.utility_objs.extra_m method), 570
get_scp_response() (spinnman.messages.scp.abstract_messages.AbstractSCPRequest method), 273
get_scp_response() (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRe method), 270
get_scp_response() (spinnman.messages.scp.impl.app_stop.AppStop method), 280
get_scp_response() (spinnman.messages.scp.impl.application_run.ApplicationRun method), 281
get_scp_response() (spinnman.messages.scp.impl.ApplicationRun method), 296
get_scp_response() (spinnman.messages.scp.impl.AppStop method), 296
get_scp_response() (spinnman.messages.scp.impl.bmp_get_version.BMPGetVersion method), 281
get_scp_response() (spinnman.messages.scp.impl.bmp_set_led.BMPSetLed method), 282
get_scp_response() (spinnman.messages.scp.impl.BMPGetVersion

method), 297
get_scp_response() (spinnman.messages.scp.impl.BMPSetLed method), 297
get_scp_response() (spinnman.messages.scp.impl.count_state.CountState method), 282
get_scp_response() (spinnman.messages.scp.impl.CountState method), 298
get_scp_response() (spinnman.messages.scp.impl.fill_request.FillRequest method), 283
get_scp_response() (spinnman.messages.scp.impl.FillRequest method), 298
get_scp_response() (spinnman.messages.scp.impl.fixed_route_init.FixedRouteInit method), 283
get_scp_response() (spinnman.messages.scp.impl.fixed_route_read.FixedRouteRead method), 284
get_scp_response() (spinnman.messages.scp.impl.FixedRouteInit method), 307
get_scp_response() (spinnman.messages.scp.impl.FixedRouteRead method), 306
get_scp_response() (spinnman.messages.scp.impl.flood_fill_data.FloodFillData method), 285
get_scp_response() (spinnman.messages.scp.impl.flood_fill_end.FloodFillEnd method), 285
get_scp_response() (spinnman.messages.scp.impl.flood_fill_start.FloodFillStart method), 285
get_scp_response() (spinnman.messages.scp.impl.FloodFillData method), 298
get_scp_response() (spinnman.messages.scp.impl.FloodFillEnd method), 299
get_scp_response() (spinnman.messages.scp.impl.FloodFillStart method), 299
get_scp_response() (spinnman.messages.scp.impl.get_chip_info.GetChipInfo method), 286
get_scp_response() (spinnman.messages.scp.impl.get_version.GetVersion method), 286
get_scp_response() (spinnman.messages.scp.impl.GetChipInfo method), 297
get_scp_response() (spinnman.messages.scp.impl.GetVersion method), 305
get_scp_response() (spinnman.messages.scp.impl.iptag_clear.IPTagClear method), 287
get_scp_response() (spinnman.messages.scp.impl.iptag_get.IPTagGet method), 287
get_scp_response() (spinnman.messages.scp.impl.iptag_get_info.IPTagGetInfo method), 288
get_scp_response() (spinnman.messages.scp.impl.iptag_set.IPTagSet method), 289
get_scp_response() (spinnman.messages.scp.impl.iptag_set_tto.IPTagSetTTO method), 289
get_scp_response() (spinnman.messages.scp.impl.IPTagClear method), 299
get_scp_response() (spinnman.messages.scp.impl.IPTagGet method), 299
get_scp_response() (spinnman.messages.scp.impl.IPTagGetInfo method), 300
get_scp_response() (spinnman.messages.scp.impl.IPTagSet method), 300
get_scp_response() (spinnman.messages.scp.impl.IPTagSetTTO method), 300
get_scp_response() (spinnman.messages.scp.impl.read_adc.ReadADC method), 289
get_scp_response() (spinnman.messages.scp.impl.read_fpga_register.ReadFPGARegister method), 290
get_scp_response() (spinnman.messages.scp.impl.read_link.ReadLink method), 290
get_scp_response() (spinnman.messages.scp.impl.read_memory.ReadMemory method), 290
get_scp_response() (spinnman.messages.scp.impl.ReadADC method), 301
get_scp_response() (spinnman.messages.scp.impl.ReadFPGARegister method), 301
get_scp_response() (spinnman.messages.scp.impl.ReadLink method),

302
get_scp_response() (spinnman.messages.scp.impl.ReadMemory method), 302
get_scp_response() (spinnman.messages.scp.impl.reverse_iptag_set.ReverseIPTagSet method), 291
get_scp_response() (spinnman.messages.scp.impl.ReverseIPTagSet method), 303
get_scp_response() (spinnman.messages.scp.impl.router_alloc.RouterAlloc method), 291
get_scp_response() (spinnman.messages.scp.impl.router_clear.RouterClear method), 292
get_scp_response() (spinnman.messages.scp.impl.router_init.RouterInit method), 292
get_scp_response() (spinnman.messages.scp.impl.RouterAlloc method), 303
get_scp_response() (spinnman.messages.scp.impl.RouterClear method), 303
get_scp_response() (spinnman.messages.scp.impl.RouterInit method), 304
get_scp_response() (spinnman.messages.scp.impl.sdram_alloc.SDRAMAlloc method), 293
get_scp_response() (spinnman.messages.scp.impl.sdram_de_alloc.SDRAMDeAlloc method), 293
get_scp_response() (spinnman.messages.scp.impl.SDRAMAlloc method), 304
get_scp_response() (spinnman.messages.scp.impl.SDRAMDeAlloc method), 304
get_scp_response() (spinnman.messages.scp.impl.send_signal.SendSignal method), 293
get_scp_response() (spinnman.messages.scp.impl.SendSignal method), 305
get_scp_response() (spinnman.messages.scp.impl.set_led.SetLED method), 294
get_scp_response() (spinnman.messages.scp.impl.set_power.SetPower method), 294
get_scp_response() (spinnman.messages.scp.impl.SetLED method), 300
get_scp_response() (spinnman.messages.scp.impl.SetPower method), 301
get_scp_response() (spinnman.messages.scp.impl.write_fpga_register.WriteFPGARegister method), 295
get_scp_response() (spinnman.messages.scp.impl.write_link.WriteLink method), 295
get_scp_response() (spinnman.messages.scp.impl.write_memory.WriteMemory method), 296
get_scp_response() (spinnman.messages.scp.impl.WriteFPGARegister method), 305
get_scp_response() (spinnman.messages.scp.impl.WriteLink method), 306
get_scp_response() (spinnman.messages.scp.impl.WriteMemory method), 306
get_sdram_usage() (spinn_front_end_common.utility_models.live_packet_g static method), 597
get_sdram_usage() (spinn_front_end_common.utility_models.LivePacketGa static method), 608
get_sdram_usage() (spinn_front_end_common.utility_models.reverse_ip_ta static method), 603
get_sdram_usage() (spinn_front_end_common.utility_models.ReverseIPTag static method), 614
get_sdram_usage_for_atoms() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP method), 781
get_sdram_usage_for_atoms() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
get_sdram_usage_for_atoms() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSou method), 797
get_sdram_usage_for_atoms() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 803
get_sdram_usage_for_atoms() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.Delay method), 808
get_sdram_usage_for_atoms() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 812
get_sdram_usage_for_global_parameters_in_bytes() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron method), 734
get_sdram_usage_for_global_parameters_in_bytes() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuron method), 739

get_sdram_usage_in_bytes() (spynnaker.pyNN.models.common.gsyn_inhibitory_recorder.GsynInhibitoryRecorder method), 671
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder method), 672
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.common.MultiSpikeRecorder method), 676
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder method), 672
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.common.NeuronRecorder method), 675
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.common.spike_recorder.SpikeRecorder method), 673
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.common.SpikeRecorder method), 676
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO method), 765
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_io.AbstractSynapseIO method), 766
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 765
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 766
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
get_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.SynapticManager method), 791
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput method), 702
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.AbstractAdditionalInput method), 704
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.AdditionalInputCa2Adaptive method), 703
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Adaptive method), 704
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType method), 724
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 734
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 739
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 767
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 771
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType method), 774
get_sdram_usage_per_neuron_in_bytes() (spynnaker.pyNN.models.neuron.threshold_types.AbstractThresholdType method), 776
get_simulation_header_array() (in module spinn_front_end_common.interface.simulation.simulation_utilities), 546
get_simulator() (in module spinn_front_end_common.utilities.globals_variables), 579
get_slice() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 120
get_slice() (pacman.model.graphs.common.GraphMapper method), 122
get_slice() (pacman.model.graphs.common.graph_mapper.GraphMapper method), 120
get_slices() (pacman.model.graphs.common.GraphMapper method), 122
get_spike_counts() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 814
get_spike_value_from_fpga_retina() (in module spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina), 655
get_spike_value_from_robot_retina() (in module spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina), 658
get_spikes() (spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable method), 670
get_spikes() (spynnaker.pyNN.models.common.AbstractSpikeRecordable method), 675
get_spikes() (spynnaker.pyNN.models.common.eieio_spike_recorder.EIEIOSpikeRecorder method), 671
get_spikes() (spynnaker.pyNN.models.common.EIEIOSpikeRecorder method), 675
get_spikes() (spynnaker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder method), 672

get_spikes() (spynnaker.pyNN.models.common.MultiSpikeRecorder method), 676
get_spikes() (spynnaker.pyNN.models.common.spike_recorder.SpikeRecorder method), 673
get_spikes() (spynnaker.pyNN.models.common.SpikeRecorder method), 676
get_spikes() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 781
get_spikes() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
get_spikes() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray method), 794
get_spikes() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 797
get_spikes() (spynnaker.pyNN.models.spike_source.SpikeSourceArray method), 800
get_spikes() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 803
get_spikes() (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector method), 809
get_spikes() (spynnaker.pyNN.models.utility_models.SpikeInjector method), 813
get_spinn5_geometry() (spinn_machine.spinnaker_triad_geometry.SpiNNakerTriadGeometry static method), 39
get_spinn5_geometry() (spinn_machine.SpiNNakerTriadGeometry static method), 55
get_spinnaker_link_with_id() (spinn_machine.Machine method), 49
get_spinnaker_link_with_id() (spinn_machine.machine.Machine method), 32
get_standard_deviation() (in module spynnaker.pyNN.utilities.utility_calls), 831
get_state_for_region() (spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state.EndBufferingState method), 516
get_state_for_region() (spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state.EndBufferingState method), 524
get_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamics method), 756
get_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamics method), 761
get_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
get_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 763
get_str() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser method), 4
get_structure_value() (data_specification.data_specification_generator.DataSpecificationGenerator method), 434
get_structure_value() (data_specification.DataSpecificationGenerator method), 472
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 767
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 771
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta method), 768
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exp.SynapseTypeDualExp method), 769
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential method), 770
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 773
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExp method), 772
get_synapse_id_by_target() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential method), 772
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 767
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 771
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta method), 768
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exp.SynapseTypeDualExp method), 769
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential method), 770
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 773
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExp method), 772
get_synapse_targets() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential method), 772
get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 768
get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 771

get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta method), 768
get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential method), 769
get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential method), 770
get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 773
get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential method), 772
get_synapse_type_parameter_types() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential method), 773
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 768
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType method), 771
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta method), 768
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential method), 769
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential method), 770
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 773
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential method), 772
get_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential method), 773
get_synapses() (spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO method), 765
get_synapses() (spynnaker.pyNN.models.neuron.synapse_io.AbstractSynapseIO method), 766
get_synapses() (spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased method), 765
get_synapses() (spynnaker.pyNN.models.neuron.synapse_io.SynapseIORowBased method), 766
get_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure method), 743
get_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure method), 744
get_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_delta.SynapseTypeDelta method), 744
get_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_dual_exponential.SynapseTypeDualExponential method), 744
get_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure_exponential.SynapseTypeExponential method), 744
get_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseTypeDelta method), 744
get_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseTypeDualExponential method), 744
get_tags() (spinnman.processes.get_tags_process.GetTagsProcess method), 364
get_tags() (spinnman.processes.GetTagsProcess method), 369
get_tags() (spinnman.transceiver.Transceiver method), 395
get_threshold_parameter_types() (spynnaker.pyNN.external_devices_models.threshold_type_multicast_d method), 658
get_threshold_parameter_types() (spynnaker.pyNN.external_devices_models.ThresholdTypeMulticastDe method), 665
get_threshold_parameter_types() (spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_ty method), 774
get_threshold_parameter_types() (spynnaker.pyNN.models.neuron.threshold_types.AbstractThresholdTy method), 776
get_threshold_parameter_types() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maas method), 774
get_threshold_parameter_types() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static method), 775
get_threshold_parameter_types() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass method), 777
get_threshold_parameter_types() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic method), 776
get_threshold_parameters() (spynnaker.pyNN.external_devices_models.threshold_type_multicast_d method), 658
get_threshold_parameters() (spynnaker.pyNN.external_devices_models.ThresholdTypeMulticastDe method), 665
get_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_ty method), 774
get_threshold_parameters() (spyn-

naker.pyNN.models.neuron.threshold_types.AbstractThresholdType method), 776
get_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic method), 775
get_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static method), 775
get_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic method), 777
get_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic method), 777
get_timed_commands_bytes() (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex static method), 590
get_timed_commands_bytes() (spinn_front_end_common.utility_models.CommandSenderMachineVertex static method), 606
get_translator() (spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor method), 650
get_translator() (spynnaker.pyNN.external_devices_models.AbstractEthernetSensor method), 659
get_translator() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_resource.AbstractPushBotEthernetResource method), 629
get_translator() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_resource.PushBotEthernetResource method), 636
get_units() (spynnaker.pyNN.models.abstract_models.abstract_contains_units_vertex.AbstractContainsUnits method), 666
get_units() (spynnaker.pyNN.models.abstract_models.AbstractContainsUnits method), 669
get_units() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 781
get_units() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
get_units() (spynnaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance method), 725
get_units() (spynnaker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent method), 725
get_units() (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance method), 727
get_units() (spynnaker.pyNN.models.neuron.input_types.InputTypeCurrent method), 727
get_units() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh method), 736
get_units() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate method), 737
get_units() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAnd method), 738
get_units() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
get_units() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate method), 742
get_units() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelL method), 742
get_units() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_d method), 769
get_units() (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_e method), 770
get_units() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDu method), 772
get_units() (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeEx method), 773
get_units() (spynnaker.pyNN.models.neuron.threshold_types.threshold_type method), 775
get_units() (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTyp method), 777
get_user_0_register_address_from_core() (spinnman.transceiver.Transceiver method), 395
get_user_1_register_address_from_core() (spinnman.transceiver.Transceiver method), 396
get_user_2_register_address_from_core() (spinnman.transceiver.Transceiver method), 396
get_valid_component_sensor.AbstractEthernetSensor module spinn_utilities.helpful_functions), 9
get_value() (pacman.model.resources.abstract_resource.AbstractResource method), 152
get_value() (pacman.model.resources.abstract_resource.PushBotEthe method), 159
get_value() (pacman.model.resources.CoreResource method), 152
get_value() (pacman.model.resources.CoreResource method), 163
get_value() (pacman.model.resources.cpu_cycles_per_tick_resource.CPUC method), 153
get_value() (pacman.model.resources.CPUCyclesPerTickResource method), 160
get_value() (pacman.model.resources.dtcm_resource.DTCMResource method), 153
get_value() (pacman.model.resources.DTCMResource method), 160
get_value() (pacman.model.resources.iptag_resource.IPtagResource method), 154
get_value() (pacman.model.resources.IPtagResource method), 160
get_value() (pacman.model.resources.reverse_iptag_resource.ReverseIPtagR method), 156
get_value() (pacman.model.resources.ReverseIPtagResource method), 162
get_value() (pacman.model.resources.sdram_resource.SDRAMResource method), 157
get_value() (pacman.model.resources.SDRAMResource method), 162
get_value() (pacman.model.resources.specific_board_iptag_resource.Specif method), 157
get_value() (pacman.model.resources.specific_board_reverse_iptag_resourc method), 158

get_value() (pacman.model.resources.specific_chip_sdram_resource.SpecificChipSDRAMResource method), 159
get_value() (pacman.model.resources.specific_core_resource.SpecificCoreResource method), 159
get_value() (pacman.model.resources.SpecificChipSDRAMResource method), 163
get_value() (pacman.model.resources.SpecificCoreResource method), 164
get_value() (spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable method), 667
get_value() (spynnaker.pyNN.models.abstract_models.AbstractPopulationSettable method), 668
get_value() (spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable method), 673
get_value() (spynnaker.pyNN.models.common.SimplePopulationSettable method), 676
get_value() (spynnaker.pyNN.models.neural_properties.neuron_parameter.NeuronParameter method), 701
get_value() (spynnaker.pyNN.models.neural_properties.NeuronParameter method), 701
get_value() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 781
get_value() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
get_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
get_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 759
get_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 763
get_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_variance() (in module spynnaker.pyNN.utilities.utility_calls), 831
get_vcpu_address() (in module spinnman.utilities.utility_functions), 379
get_version() (spinnman.processes.get_version_process.GetVersionProcess method), 364
get_version() (spinnman.processes.GetVersionProcess method), 369
get_vertex_executable_suffix() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
get_vertex_executable_suffix() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 761
get_vertex_executable_suffix() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
get_vertex_executable_suffix() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 759
get_vertex_executable_suffix() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 763
get_vertex_executable_suffix() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_vertex_on_processor() (pacman.model.placements.Placements method), 151
get_vertex_on_processor() (pacman.model.placements.abstract_placements.AbstractPlacements method), 149
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connectable_connector.AbstractPopulationSettable method), 677
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnectable method), 687
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector_parameter.AllToAllConnectorParameter method), 679
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnectorAbstractPopulationVertex method), 688
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnectorStatic method), 681
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnectorSTDP method), 681
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnectorStatic method), 682
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnectorAbstractSynapseDynamics method), 691
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 692
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnectorStatic method), 683
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnectorSTDP method), 693
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector

method), 684
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector method), 685
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector method), 686
get_weight_maximum() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
get_weight_maximum() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
get_weight_maximum() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 761
get_weight_maximum() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 759
get_weight_maximum() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 677
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector method), 679
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 689
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 681
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector method), 681
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector method), 691
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 692
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 684
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector method), 693
get_weight_mean() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 684
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 693
get_weight_mean() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 685
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector method), 686
get_weight_mean() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
get_weight_mean() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
get_weight_mean() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 761
get_weight_mean() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 760
get_weight_mean() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 678
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector

method), 679
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector method), 688
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector method), 680
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector method), 690
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector method), 681
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector method), 682
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector method), 682
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector method), 690
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector method), 691
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector method), 692
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector method), 684
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.FromListConnector method), 693
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.multapse_connector.MultapseConnector method), 684
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector method), 694
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector method), 686
get_weight_variance() (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldConnector method), 695
get_weight_variance() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
get_weight_variance() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseD method), 761
get_weight_variance() (spynnaker.pyNN.models.neuron.synapse_dynamics.DistanceDependentProbability method), 760
get_weight_variance() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 764
get_words() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_s method), 757
get_words() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractS method), 761
GET_WR_PTR (data_specification.enums.commands.Commands attribute), 418
GET_WR_PTR (data_specification.enums.commands.Commands attribute), 415
get_x_from_fpga_retina() (in module spynnaker.pyNN.external_devices_models.external_spinnaker_link_fp 655
get_x_from_key() (in module spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ 729
get_x_from_robot_retina() (in module spynnaker.pyNN.external_devices_models.munich_spinnaker_link_ret 658
get_y_from_fpga_retina() (in module spynnaker.pyNN.external_devices_models.external_spinnaker_link_fp 655
get_y_from_key() (in module spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ 729
get_y_from_robot_retina() (in module spynnaker.pyNN.external_devices_models.munich_spinnaker_link_ret 658
GetChipInfo (class in spinnman.messages.scp.impl), 297
GetChipInfo (class in spinnman.messages.scp.impl.get_chip_info), 285
GetChipInfoResponse (class in spinnman.messages.scp.impl.get_chip_info_response), 286
GetCPUInfoProcess (class in spinnman.processes), 368
GetCPUInfoProcess (class in spinnman.processes.get_cpu_info_process), 363
GetHeapProcess (class in spinnman.processes), 368
GetHeapProcess (class in spinnman.processes.get_heap_process), 363
GetMachineProcess (class in spinnman.processes), 368
GetMachineProcess (class in spinnman.processes.get_machine_process), 363
GetMultiCastRoutesProcess (class in spinnman.processes), 369
GetMultiCastRoutesProcess (class in spinn-

man.processes.get_routes_process), 364
GetReinjectionStatusMessage (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.get_reinjection_status_message), 567
GetReinjectionStatusMessageResponse (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.get_reinjection_status_response), 568
GetTagsProcess (class in spinnman.processes), 369
GetTagsProcess (class in spinnman.processes.get_tags_process), 364
GetVersion (class in spinnman.messages.scp.impl), 305
GetVersion (class in spinnman.messages.scp.impl.get_version), 286
GetVersionProcess (class in spinnman.processes), 369
GetVersionProcess (class in spinnman.processes.get_version_process), 364
GetVersionResponse (class in spinnman.messages.scp.impl.get_version_response), 287
Graph (class in pacman.model.graphs.impl), 126
Graph (class in pacman.model.graphs.impl.graph), 123
graph_mapper (spinn_front_end_common.interface.abstract_spinnaker_common.AbstractSpinnakerBase attribute), 548
graph_mapper (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
graph_mapper (spinn_front_end_common.utilities.FailedState attribute), 583
graph_mapper (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 582
graph_mapper (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
graph_mapper() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol.PushBotSpiNNakerLinkRetinaDevice method), 644
graph_mapper() (spynnaker.pyNN.external_devices_models.push_bot.PushBotSpiNNakerLinkRetinaDevice method), 648
GraphBinaryGatherer (class in spinn_front_end_common.interface.interface_functions), 538
GraphBinaryGatherer (class in spinn_front_end_common.interface.interface_functions.graph_binary_gatherer), 531
GraphDataSpecificationWriter (class in spinn_front_end_common.interface.interface_functions), 538
GraphDataSpecificationWriter (class in spinn_front_end_common.interface.interface_functions.graph_data_specification_writer), 532
GraphEdgeFilter (class in spynnaker.pyNN.overridden_pacman_functions.graph_edge_filter), 817
GraphEdgeWeightUpdater (class in spynnaker.pyNN.overridden_pacman_functions.graph_edge_weight_updater), 817
GraphMapper (class in pacman.model.graphs.common), 121
GraphMapper (class in pacman.model.graphs.common.impl.graph_mapper), 119
GraphMeasurer (class in spinn_front_end_common.interface.interface_functions), 538
GraphMeasurer (class in spinn_front_end_common.interface.interface_functions.graph_me 532
GraphProvenanceGatherer (class in spinn_front_end_common.interface.interface_functions), 538
GraphProvenanceGatherer (class in spinn_front_end_common.interface.interface_functions.graph_pr 532
GREATER_THAN (data_specification.enums.Condition attribute), 419
GREATER_THAN (data_specification.enums.condition.Condition attribute), 416
GREATER_THAN_OR_EQUAL (data_specification.enums.Condition attribute), 419
GREATER_THAN_OR_EQUAL (data_specification.enums.condition.Condition attribute), 416
grid() (in module spynnaker.plot_utils), 838
GSYN_EXCITATORY_RECORDING_REGION (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP attribute), 779
GSYN_EXCITATORY_RECORDING_REGION (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 786
GSYN_INHIBITORY_RECORDING_REGION (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP attribute), 779
GSYN_INHIBITORY_RECORDING_REGION (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 786
GsynInhibitoryRecorder (class in spynnaker.pyNN.models.common.gsyn_inhibitory_recorder), 671

H

HALF_WORD (spinnman.processes.fill_process.FillDataType attribute), 363
HALF_WORD (spinnman.processes.FillDataType attribute), 369
handle_control_register_response() (spinnman.processes.read_router_diagnostics_process.ReadRouterDiag method), 366
handle_control_register_response() (spinnman.processes.ReadRouterDiagnosticsProcess method), 370

handle_error_status_response() (spinnman.processes.read_router_diagnostics_process.ReadRouterDiagnosticsProcess method), 366
handle_error_status_response() (spinnman.processes.ReadRouterDiagnosticsProcess method), 370
handle_extra_iobuf_response() (spinnman.processes.read_iobuf_process.ReadIOBufProcess method), 366
handle_extra_iobuf_response() (spinnman.processes.ReadIOBufProcess method), 370
handle_first_iobuf_response() (spinnman.processes.read_iobuf_process.ReadIOBufProcess method), 366
handle_first_iobuf_response() (spinnman.processes.ReadIOBufProcess method), 370
handle_flexi_field() (in module pacman.utilities.algorithm_utilities.field_based_system_utilities), 195
handle_get_tag_response() (spinnman.processes.get_tags_process.GetTagsProcess method), 364
handle_get_tag_response() (spinnman.processes.GetTagsProcess method), 369
handle_iobuf_address_response() (spinnman.processes.read_iobuf_process.ReadIOBufProcess method), 366
handle_iobuf_address_response() (spinnman.processes.ReadIOBufProcess method), 370
handle_read_response() (spinnman.processes.get_routes_process.GetMultiCastRoutesProcess method), 364
handle_read_response() (spinnman.processes.GetMultiCastRoutesProcess method), 369
handle_read_response() (spinnman.processes.read_fixed_route_routing_entry_process.ReadFixedRouteRoutingEntryProcess method), 366
handle_read_response() (spinnman.processes.ReadFixedRouteRoutingEntryProcess method), 371
handle_read_vcpu_offset() (spinnman.processes.get_cpu_info_process.GetCPUInfoProcess method), 363
handle_read_vcpu_offset() (spinnman.processes.GetCPUInfoProcess method), 368
handle_register_response() (spinnman.processes.read_router_diagnostics_process.ReadRouterDiagnosticsProcess method), 367
handle_register_response() (spinnman.processes.ReadRouterDiagnosticsProcess method), 370
handle_reinjection_status_response() (spinn_front_end_common.utilities.utility_objs.extra_monitor_sc method), 570
handle_response() (spinnman.processes.get_cpu_info_process.GetCPUInfoProcess method), 363
handle_response() (spinnman.processes.GetCPUInfoProcess method), 368
handle_response() (spinnman.processes.read_memory_process.ReadMemoryProcess method), 366
handle_response() (spinnman.processes.ReadMemoryProcess method), 370
handle_response() (spinnman.processes.send_single_command_process.SendSingleComm method), 367
handle_response() (spinnman.processes.SendSingleCommandProcess method), 371
handle_router_alloc_response() (spinnman.processes.load_routes_process.LoadMultiCastRoutesProcess method), 365
handle_router_alloc_response() (spinnman.processes.LoadMultiCastRoutesProcess method), 369
handle_sdram_alloc_response() (spinnman.processes.de_alloc_sdram_process.DeAllocSDRAMProcess method), 362
handle_sdram_alloc_response() (spinnman.processes.DeAllocSDRAMProcess method), 368
handle_sdram_alloc_response() (spinnman.processes.malloc_sdram_process.MallocSDRAMProcess method), 365
handle_sdram_alloc_response() (spinnman.processes.MallocSDRAMProcess method), 370
handle_tag_info_response() (spinnman.processes.get_tags_process.GetTagsProcess method), 364
handle_tag_info_response() (spinnman.processes.GetTagsProcess method), 369
hardware (spinnman.model.version_info.VersionInfo attribute), 346
hardware (spinnman.model.VersionInfo attribute), 360
hardware_version (spinnman.messages.spinnaker_boot.system_variable_boot_values.Syst attribute), 318

hardware_version (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
has_ran (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
has_ran (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
has_ran (spinn_front_end_common.utilities.FailedState attribute), 583
has_ran (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 582
has_ran (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
has_reset_last (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
has_reset_last (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface attribute), 837
has_reset_last (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState attribute), 829
has_simulator() (in module spinn_front_end_common.utilities.globals_variables), 579
has_wrap_arounds (spinn_machine.Machine attribute), 49
has_wrap_arounds (spinn_machine.machine.Machine attribute), 32
HBPAllocator (class in spinn_front_end_common.interface.interface_functions), 538
HBPAllocator (class in spinn_front_end_common.interface.interface_functions.hbp_allocator), 532
HBPMaxMachineGenerator (class in spinn_front_end_common.interface.interface_functions), 538
HBPMaxMachineGenerator (class in spinn_front_end_common.interface.interface_functions.hbp_max_machine_generator), 532
HeapElement (class in spinnman.model), 356
HeapElement (class in spinnman.model.heap_element), 341
heat_plot() (in module spynnaker.plot_utils), 838
height (spinnman.model.machine_dimensions.MachineDimensions attribute), 342
height (spinnman.model.MachineDimensions attribute), 357
height (spinnman.model.p2p_table.P2PTable attribute), 343
height (spinnman.model.P2PTable attribute), 357
HELLO (spinnman.messages.spinnaker_boot.spinnaker_boot_opcodes.SpinnakerBootOpCodes attribute), 317
HELLO (spinnman.messages.spinnaker_boot.SpinnakerBootOpCode attribute), 321
HHCondExp (class in spynnaker.pyNN.models.neuron.builds), 718
HHCondExp (class in spynnaker.pyNN.models.neuron.builds.hh_cond_exp), 706
hi (pacman.utilities.utility_objs.Field attribute), 203
hi (pacman.utilities.utility_objs.field.Field attribute), 199
high() (in module spynnaker.pyNN.utilities.utility_calls), 831
high() (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats method), 827
high() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
HilbertPlacer (class in pacman.operations.rigged_algorithms.hilbert_placer), 189
HilbertState (class in pacman.operations.rigged_algorithms.hilbert_state), 189
HOST_DATA_READ (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
HOST_DATA_READ_ACK (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
HOST_SEND_SEQUENCED_DATA (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
HostDataRead (class in spinnman.messages.eieio.command_messages), 254
HostDataRead (class in spinnman.messages.eieio.command_messages.host_data_read), 250
HostDataReadAck (class in spinnman.messages.eieio.command_messages), 256
HostDataReadAck (class in spinnman.messages.eieio.command_messages.host_data_read_ack), 250
HostExecuteDataSpecification (class in spinn_front_end_common.interface.interface_functions), 538
HostExecuteDataSpecification (class in spinn_front_end_common.interface.interface_functions.host_execute_data_specification), 532
hostname (spinn_front_end_common.utilities.utility_objs.live_packet_gather.LivePacketGather attribute), 573
hostname (spinn_front_end_common.utilities.utility_objs.LivePacketGather attribute), 576
hostn_core (spinnman.connections.socket_address_with_chip.SocketAddressWithChip attribute), 244
HostCore (spinnman.connections.SocketAddressWithChip attribute), 247
HostSendSequencedData (class in spinn-

man.messages.eieio.command_messages), 255
HostSendSequencedData (class in spinnman.messages.eieio.command_messages.host_send_sequenced_data), 251

I

i_alpha (spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.AdditionalInputCa2Adaptive attribute), 703
i_alpha (spynnaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Adaptive attribute), 704
i_ca2 (spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.AdditionalInputCa2Adaptive attribute), 703
i_ca2 (spynnaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Adaptive attribute), 704
i_offset (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh attribute), 736
i_offset (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate attribute), 737
i_offset (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 740
i_offset (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate attribute), 742
id_counter (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon attribute), 833
id_to_index() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 814
id_to_local_index() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 814
identifier (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition attribute), 140
identifier (pacman.model.graphs.AbstractOutgoingEdgePartition attribute), 146
identifier (pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition attribute), 126
identifier (pacman.model.graphs.impl.OutgoingEdgePartition attribute), 129
IDLE (spinnman.model.enums.cpu_state.CPUState attribute), 325
IDLE (spinnman.model.enums.CPUState attribute), 329
IF (data_specification.enums.Commands attribute), 418
IF (data_specification.enums.commands.Commands attribute), 415
IFCondAlpha (class in spynnaker.pyNN.models.neuron.builds), 719
IFCondAlpha (class in spynnaker.pyNN.models.neuron.builds.if_cond_alpha), 707
IFCondExpBase (class in spynnaker.pyNN.models.neuron.builds), 719
IFCondExpBase (class in spynnaker.pyNN.models.neuron.builds.if_cond_exp_base), 708
IFCondExpStoc (class in spynnaker.pyNN.models.neuron.builds), 722
IFCondExpStoc (class in spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc), 709
IFCurrAlpha (class in spynnaker.pyNN.models.neuron.builds), 719
IFCurrAlpha (class in spynnaker.pyNN.models.neuron.builds.if_curr_alpha), 710
IFCurrDelta (class in spynnaker.pyNN.models.neuron.builds), 722
IFCurrDelta (class in spynnaker.pyNN.models.neuron.builds.if_curr_delta), 711
IFCurrDualExpBase (class in spynnaker.pyNN.models.neuron.builds), 720
IFCurrDualExpBase (class in spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base), 712
IFCurrExpBase (class in spynnaker.pyNN.models.neuron.builds), 720
IFCurrExpBase (class in spynnaker.pyNN.models.neuron.builds.if_curr_exp_base), 713
IFCurrExpCa2Adaptive (class in spynnaker.pyNN.models.neuron.builds), 723
IFCurrExpCa2Adaptive (class in spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive), 714
IFFacetsConductancePopulation (class in spynnaker.pyNN.models.neuron.builds), 721
IFFacetsConductancePopulation (class in spynnaker.pyNN.models.neuron.builds.if_facets_hardware1), 715
IN_RUN (spinn_front_end_common.interface.simulator_state.Simulator_St attribute), 550
incoming_link (pacman.model.routing_table_by_partition.multicast_routing attribute), 171
incoming_link (pacman.model.routing_table_by_partition.MulticastRouting attribute), 173
incoming_processor (pacman.model.routing_table_by_partition.multicast_routing_table_b attribute), 172
incoming_processor (pacman.model.routing_table_by_partition.MulticastRoutingTableBy attribute), 173
increment_count() (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIOData method), 259
increment_count() (spinnman.messages.eieio.data_messages.EIEIODataHeader method), 263
increment_none_labelled_edge_count()

(spinn_front_end_common.interface.abstract_spinnaker_base method), 548
increment_none_labelled_vertex_count (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
increment_none_labelled_vertex_count (spinn_front_end_common.utilities.FailedState attribute), 583
increment_none_labelled_vertex_count (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 583
increment_none_labelled_vertex_count (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
increment_none_labelled_vertex_count() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
increment_write_pointer() (data_specification.memory_region.MemoryRegion method), 455
increment_write_pointer() (data_specification.MemoryRegion method), 488
index (spynnaker.pyNN.models.neural_projections.synapse_information.SynapseInformation attribute), 698
index (spynnaker.pyNN.models.neural_projections.SynapseInformation attribute), 701
INIT (spinn_front_end_common.interface.simulator_state.Simulator_State attribute), 550
INITIALISE (spinnman.messages.scp.enums.Signal attribute), 280
INITIALISE (spinnman.messages.scp.enums.signal.Signal attribute), 277
initialise() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage class method), 70
initialise() (spinn_storage_handlers.BufferedTempfileDataStorage class method), 73
initialise_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAs2dArray method), 729
initialise_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch method), 730
initialise_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAs2dArray method), 731
initialise_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch method), 732
INITIALISING (spinnman.model.enums.cpu_state.CPUState attribute), 325
INITIALISING (spinnman.model.enums.CPUState attribute), 329
initialize() (spynnaker.pyNN.models.abstract_models.abstract_population_i method), 666
initialize() (spynnaker.pyNN.models.abstract_models.AbstractPopulationIni method), 668
initialize() (spynnaker.pyNN.models.neuron.abstract_population_vertex.Ab method), 781
initialize() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
initialize() (spynnaker.pyNN.models.pynn_population_common.PyNNPopu method), 814
initialize() (spynnaker.pyNN.models.neuron.neuron_models.neuron_mod method), 736
initialize_u() (spynnaker.pyNN.models.neuron.neuron_models.NeuronMod method), 740
initialize_v() (spynnaker.pyNN.models.neuron.neuron_models.neuron_mod method), 736
initialize_v() (spynnaker.pyNN.models.neuron.neuron_models.neuron_mod method), 737
initialize_v() (spynnaker.pyNN.models.neuron.neuron_models.NeuronMod method), 740
initialize_v() (spynnaker.pyNN.models.neuron.neuron_models.NeuronMod method), 742
inject() (in module pacman.executor.injection_decorator), 94
inject() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulati method), 814
inject_items() (in module pacman.executor.injection_decorator), 95
injection_context (class in pacman.executor.injection_decorator), 95
InjectionException, 94
INPUT_BUFFERING_SDP_PORT (spinn_front_end_common.utilities.constants.SDP_PORTS attribute), 578
input_matches() (pacman.executor.algorithm_decorators.abstract_input.Abs method), 87
input_matches() (pacman.executor.algorithm_decorators.AbstractInput method), 91
input_matches() (pacman.executor.algorithm_decorators.all_of_input.AllOf method), 89
input_matches() (pacman.executor.algorithm_decorators.AllOfInput method), 91
input_matches() (pacman.executor.algorithm_decorators.one_of_input.OneO method), 89
input_matches() (pacman.executor.algorithm_decorators.OneOfInput method), 92
input_matches() (pacman.executor.algorithm_decorators.single_input.Single method), 90
input_matches() (pacman.executor.algorithm_decorators.SingleInput method), 92
input_type (spynnaker.pyNN.models.neuron.abstract_population_vertex.Ab attribute), 781
input_type (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 789

InputTypeConductance (class in spynnaker.pyNN.models.neuron.input_types), 726
InputTypeConductance (class in spynnaker.pyNN.models.neuron.input_types.input_type_conductance), 725
InputTypeCurrent (class in spynnaker.pyNN.models.neuron.input_types), 727
InputTypeCurrent (class in spynnaker.pyNN.models.neuron.input_types.input_type_current), 725
InsertChipPowerMonitorsToGraphs (class in spinn_front_end_common.interface.interface_functions.insert_chip_power_monitors_to_graphs), 532
InsertEdgesToLivePacketGatherers (class in spinn_front_end_common.interface.interface_functions), 538
InsertEdgesToLivePacketGatherers (class in spinn_front_end_common.interface.interface_functions.insert_edges_to_live_packet_gatherers), 533
InsertExtraMonitorVerticesToGraphs (class in spinn_front_end_common.interface.interface_functions.insert_extra_monitor_vertices_to_graphs), 533
InsertLivePacketGatherersToGraphs (class in spinn_front_end_common.interface.interface_functions), 538
InsertLivePacketGatherersToGraphs (class in spinn_front_end_common.interface.interface_functions.insert_live_packet_gatherers_to_graphs), 533
install_cfg_and_IOError() (in module spinn_utilities.conf_loader), 7
instance_key (spynnaker.pyNN.protocols.munich_io_spinnaker_protocol.MunichIoSpiNNakerProtocol attribute), 819
instance_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLiveSpikesProtocol attribute), 824
instance_n_keys (pacman.utilities.utility_objs.flexi_field.FlexiField attribute), 199
instance_n_keys (pacman.utilities.utility_objs.FlexiField attribute), 203
instruction_counter (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 434
instruction_counter (data_specification.DataSpecificationGenerator attribute), 472
INT16 (data_specification.enums.data_type.DataType attribute), 416
INT16 (data_specification.enums.DataType attribute), 419
INT32 (data_specification.enums.data_type.DataType attribute), 416
INT32 (data_specification.enums.DataType attribute), 419
INT64 (data_specification.enums.data_type.DataType attribute), 416
INT64 (data_specification.enums.DataType attribute), 419
INT8 (data_specification.enums.data_type.DataType attribute), 416
INT8 (data_specification.enums.DataType attribute), 419
InvalidParameterType, 834
IOBUF (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
IOBUF (spinnman.model.enums.RunTimeError attribute), 330
iobuf (spinnman.model.io_buffer.IOBuffer attribute), 341
iobuf (spinnman.model.IOBuffer attribute), 356
iobuf_address (spinnman.model.cpu_info.CPUInfo attribute), 336
iobuf_address (spinnman.model.CPUInfo attribute), 351
iobuf_size (spinnman.messages.spinnaker_boot.system_variable_boot_valu attribute), 318
iobuf_size (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
iobuf_size (spinnman.model.chip_info.ChipInfo attribute), 333
iobuf_size (spinnman.model.ChipInfo attribute), 348
IOBuffer (class in spinnman.model.io_buffer), 341
IOBuffer (class in spinnman.model), 356
IP_ADDR (spinnman.messages.scp.enums.bmp_info.BMPInfo attribute), 274
IP_ADDR (spinnman.messages.scp.enums.BMPInfo attribute), 278
ip_address (pacman.model.resources.iptag_resource.IPtagResource attribute), 154
ip_address (pacman.model.resources.IPtagResource attribute), 160
ip_address (pacman.model.resources.iptag_resource.SpecificIPtag attribute), 157
ip_address (spinn_machine.Chip attribute), 41
ip_address (spinn_machine.chip.Chip attribute), 23
ip_address (spinn_machine.tags.IPTag attribute), 20
ip_address (spinn_machine.tags.iptag.IPTag attribute), 19
ip_address (spinnman.model.bmp_connection_data.BMPConnectionData attribute), 332
ip_address (spinnman.model.BMPConnectionData attribute), 348
ip_address (spinnman.model.chip_info.ChipInfo attribute), 333
ip_address (spinnman.model.ChipInfo attribute), 348
ip_tags (pacman.model.tags.Tags attribute), 180
ip_tags (pacman.model.tags.tags.Tags attribute), 178
IPAddressesConnection (class in spinnman.connections.udp_packet_connections), 240
IPAddressesConnection (class in spinnman.connections.udp_packet_connections.ip_address_connection 232
IPTag (class in spinn_machine.tags), 20

IPTag (class in spinn_machine.tags.iptag), 18
IPTAG_IDS (spinn_machine.Chip attribute), 41
IPTAG_IDS (spinn_machine.chip.Chip attribute), 23
IPTAG_TIME_OUT_WAIT_TIMES (class in spinnman.constants), 380
IPTagClear (class in spinnman.messages.scp.impl), 299
IPTagClear (class in spinnman.messages.scp.impl.iptag_clear), 287
IPTagCommand (class in spinnman.messages.scp.enums), 279
IPTagCommand (class in spinnman.messages.scp.enums.iptag_command), 274
IPTagGet (class in spinnman.messages.scp.impl), 299
IPTagGet (class in spinnman.messages.scp.impl.iptag_get), 287
IPTagGetInfo (class in spinnman.messages.scp.impl), 299
IPTagGetInfo (class in spinnman.messages.scp.impl.iptag_get_info), 288
IPTagGetInfoResponse (class in spinnman.messages.scp.impl.iptag_get_info_response), 288
IPtagResource (class in pacman.model.resources), 160
IPtagResource (class in pacman.model.resources.iptag_resource), 153
iptags (pacman.model.resources.resource_container.ResourceContainer attribute), 156
iptags (pacman.model.resources.ResourceContainer attribute), 161
IPTagSet (class in spinnman.messages.scp.impl), 300
IPTagSet (class in spinnman.messages.scp.impl.iptag_set), 288
IPTagSetTTO (class in spinnman.messages.scp.impl), 300
IPTagSetTTO (class in spinnman.messages.scp.impl.iptag_set_tto), 289
IRQ (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
IRQ (spinnman.model.enums.RunTimeError attribute), 330
is_a_pynn_random() (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface method), 837
is_a_pynn_random() (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState method), 829
is_at_end() (spinn_storage_handlers.abstract_classes.abstract_byte_reader method), 61
is_at_end() (spinn_storage_handlers.abstract_classes.AbstractByteReader method), 66
is_chip() (spinn_machine.core_subsets.CoreSubsets method), 26
is_chip() (spinn_machine.CoreSubsets method), 44
is_chip_at() (spinn_machine.Machine method), 49
is_chip_at() (spinn_machine.machine.Machine method), 32
is_chip_at() (spinn_machine.virtual_machine.VirtualMachine method), 40
is_chip_at() (spinn_machine.VirtualMachine method), 56
is_chip_available() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 202
is_chip_available() (pacman.utilities.utility_objs.ResourceTracker method), 207
is_connected() (spinnman.connections.abstract_classes.Connection method), 225
is_connected() (spinnman.connections.abstract_classes.connection.Connect method), 219
is_connected() (spinnman.connections.udp_packet_connections.udp_conne method), 235
is_connected() (spinnman.connections.udp_packet_connections.UDPConne method), 239
is_connected() (spinnman.transceiver.Transceiver method), 397
is_connected() (spynnaker.pyNN.external_devices_models.push_bot.push_b method), 631
is_connected() (spynnaker.pyNN.external_devices_models.push_bot.push_b method), 638
is_core() (spinn_machine.core_subsets.CoreSubsets method), 26
is_core() (spinn_machine.CoreSubsets method), 45
is_data_from_region_flushed() (spinn_front_end_common.interface.buffer_management.storage_ method), 510
is_data_from_region_flushed() (spinn_front_end_common.interface.buffer_management.storage_ method), 518
is_empty() (data_specification.memory_region_collection.MemoryRegionC method), 456
is_empty() (data_specification.MemoryRegionCollection method), 488
is_empty() (spinn_front_end_common.interface.buffer_management.buffer_ method), 503
is_empty() (spinn_front_end_common.interface.buffer_management.buffer_ method), 506
is_empty() (spinn_front_end_common.interface.buffer_management.buffer_ method), 505
is_empty() (spinn_front_end_common.interface.buffer_management.buffer_ method), 508
is_empty() (spinn_front_end_common.interface.buffer_management.storage method), 514
is_empty() (spinn_front_end_common.interface.buffer_management.storage method), 522
is_end_buffering_sequence_number_stored() (spinn_front_end_common.interface.buffer_management.storage_ method), 510

is_end_buffering_sequence_number_stored() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 518
is_end_buffering_state_recovered() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 510
is_end_buffering_state_recovered() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 518
is_equal_or_None() (in module pacman.utilities.utility_calls), 209
is_error() (spinnman.processes.abstract_process.AbstractProcess method), 362
is_ethernet_available (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
is_ethernet_available (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
is_ethernet_available (spinnman.model.chip_info.ChipInfo attribute), 333
is_ethernet_available (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
is_ethernet_available (spinnman.model.ChipInfo attribute), 348
is_ethernet_available (spinnman.model.ChipSummaryInfo attribute), 350
is_free (spinnman.model.heap_element.HeapElement attribute), 341
is_free (spinnman.model.HeapElement attribute), 356
is_full (spinn_front_end_common.interface.buffer_management.storage_objects.BuffersSentDeque attribute), 514
is_full (spinn_front_end_common.interface.buffer_management.storage_objects.BuffersSentDeque attribute), 522
is_in_injection_mode (spinn_front_end_common.abstract_models.abstract_supports_database_injection.AbstractSupportsDatabaseInjection attribute), 498
is_in_injection_mode (spinn_front_end_common.abstract_models.AbstractSupportsDatabaseInjection attribute), 501
is_in_injection_mode (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex attribute), 597
is_in_injection_mode (spinn_front_end_common.utility_models.LivePacketGatherMachineVertex attribute), 608
is_in_injection_mode (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex attribute), 603
is_in_injection_mode (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex attribute), 614
is_link() (spinn_machine.Router method), 54
is_link() (spinn_machine.router.Router method), 37
is_link_at() (spinn_machine.Machine method), 49
is_link_at() (spinn_machine.machine.Machine method), 32
is_link_at() (spinn_machine.virtual_machine.VirtualMachine method), 40
is_link_at() (spinn_machine.VirtualMachine method), 56
is_monitor (spinn_machine.Processor attribute), 52
is_monitor (spinn_machine.processor.Processor attribute), 35
is_next_element (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 260
is_next_element (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 264
is_next_key (pacman.operations.routing_info_allocator_algorithms.malloc_ attribute), 193
is_next_key() (spinn_front_end_common.interface.buffer_management.buff method), 504
is_next_key() (spinn_front_end_common.interface.buffer_management.buff method), 507
is_next_key() (spinn_front_end_common.interface.buffer_management.buff method), 505
is_next_key() (spinn_front_end_common.interface.buffer_management.buff method), 508
is_next_key() (spinn_front_end_common.interface.buffer_management.stor method), 513
is_next_key() (spinn_front_end_common.interface.buffer_management.stor method), 521
is_next_timestamp (spinn_front_end_common.interface.buffer_managemen attribute), 513
is_next_timestamp (spinn_front_end_common.interface.buffer_managemen attribute), 521
is_next_timestamp() (spinn_front_end_common.interface.buffer_manageme method), 504
is_next_timestamp() (spinn_front_end_common.interface.buffer_manageme method), 507
is_next_timestamp() (spinn_front_end_common.interface.buffer_manageme method), 505
is_next_timestamp() (spinn_front_end_common.interface.buffer_manageme method), 508
is_payload (spinn_front_end_common.utility_models.multi_cast_command attribute), 598
is_payload (spinn_front_end_common.utility_models.MultiCastCommand attribute), 609
is_peer_to_peer_available (spinnman.messages.spinnaker_boot.system_variable_boot_values.Syst attribute), 318
is_peer_to_peer_available (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
is_processor_occupied() (pacman.model.placements.Placements method), 151
is_processor_occupied() (pacman.model.placements.placements.Placements method), 149

is_processor_with_id() (spinn_machine.Chip method), 41
is_processor_with_id() (spinn_machine.chip.Chip method), 23
is_ready_to_receive() (spinnman.connections.abstract_classes.Listenable method), 226
is_ready_to_receive() (spinnman.connections.abstract_classes.listenable.Listenable method), 220
is_ready_to_receive() (spinnman.connections.abstract_classes.scp_receiver.SCPReceiver method), 222
is_ready_to_receive() (spinnman.connections.abstract_classes.SCPReceiver method), 227
is_ready_to_receive() (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection method), 235
is_ready_to_receive() (spinnman.connections.udp_packet_connections.UDPConnection method), 239
is_ready_to_receive() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_wifi_connection.PushBotWIFIConnection method), 631
is_ready_to_receive() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_wifi_connection.PushBotWIFIConnection method), 638
is_recording() (spinn_front_end_common.abstract_models.abstract_recordable.AbstractRecordable method), 498
is_recording() (spinn_front_end_common.abstract_models.AbstractRecordable method), 501
is_recording() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex method), 603
is_recording() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex method), 614
is_recording() (spynnaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable method), 670
is_recording() (spynnaker.pyNN.models.common.AbstractNeuronRecordable method), 674
is_recording() (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder method), 672
is_recording() (spynnaker.pyNN.models.common.NeuronRecorder method), 675
is_recording() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 781
is_recording() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
is_recording() (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex method), 785
is_recording() (spynnaker.pyNN.models.neuron.PopulationMachineVertex method), 793
is_recording() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArrayMachineVertex method), 799
is_recording() (spynnaker.pyNN.models.spike_source.SpikeSourceArrayMachineVertex method), 806
is_recording_spikes() (spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractS method), 670
is_recording_spikes() (spynnaker.pyNN.models.common.AbstractSpikeRecordable method), 675
is_recording_spikes() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP method), 782
is_recording_spikes() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
is_recording_spikes() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourc method), 794
is_recording_spikes() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSou method), 797
is_recording_spikes() (spynnaker.pyNN.models.spike_source.SpikeSourceArray method), 801
is_recording_spikes() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 804
is_recording_spikes() (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector method), 810
is_recording_spikes() (spynnaker.pyNN.models.utility_models.SpikeInjector method), 813
is_reinjecting_multicast (spinn_front_end_common.utilities.utility_objs.re_injection_statu attribute), 574
is_reinjecting_multicast (spinn_front_end_common.utilities.utility_objs.re_ attribute), 574
is_reinjecting_nearest_neighbour (spinn_front_end_common.utilities.utility_objs.re_injection_statu attribute), 574
is_reinjecting_point_to_point (spinn_front_end_common.utilities.utility_objs.re_injection_statu attribute), 575
is_root_chip (spinnman.messages.spinnaker_boot.system_variable_boot_va attribute), 318
is_root_chip (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
is_route() (spinnman.model.p2p_table.P2PTable method), 348
is_route() (spinnman.model.P2PTable method), 357
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_depen method), 745
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_depen method), 748

method), 746
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair method), 746
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceVogels2011 method), 747
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence method), 747
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditive method), 748
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet method), 749
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceMultiplicative method), 750
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamics method), 750
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsStatic method), 749
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsSTDP method), 751
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamics method), 752
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsStatic method), 754
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsSTDP method), 752
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsStatic method), 753
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsSTDP method), 753
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamics method), 754
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsStatic method), 755
is_same_as() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.SynapseDynamicsSTDP method), 755
is_same_as() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 757
is_same_as() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics method), 761
is_same_as() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
is_same_as() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 760
is_same_as() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 763
is_same_as() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
is_scamp_version_compabible() (spinnman.transceiver.Transceiver static method), 397
is_single() (in module pacman.utilities.utility_calls), 209
is_state_updated (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState attribute), 515
is_state_updated (spinn_front_end_common.interface.buffer_management.storage_objects.ChannelBufferState attribute), 523
isyn_exc (spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.IFCurrExpBase attribute), 259
isyn_exc (spynnaker.pyNN.models.neuron.builds.IFCurrExpBase attribute), 263
isyn_exc (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta attribute), 598
isyn_exc (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual.SynapseTypeDual attribute), 609
isyn_exc (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential attribute), 456
isyn_exc (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta attribute), 489
isyn_exc (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDual attribute), 523
isyn_exc (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential attribute), 259
isyn_exc2 (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual.SynapseTypeDual attribute), 263
isyn_exc2 (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDual attribute), 598
isyn_inh (spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.IFCurrExpBase attribute), 609
isyn_inh (spynnaker.pyNN.models.neuron.builds.IFCurrExpBase attribute), 456
isyn_inh (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta attribute), 489
isyn_inh (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual.SynapseTypeDual attribute), 523
isyn_inh (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential attribute), 259
isyn_inh (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta attribute), 263
isyn_inh (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDual attribute), 598
isyn_inh (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential attribute), 609
item (spinn_machine.exceptions.SpinnMachineAlreadyExistsException attribute), 97

items (pacman.executor.algorithm_decorators.algorithm_decorator.AllOf attribute), 87
items (pacman.executor.algorithm_decorators.algorithm_decorator.OneOf attribute), 87
items (pacman.executor.algorithm_decorators.AllOf attribute), 93
items (pacman.executor.algorithm_decorators.OneOf attribute), 93
iterchips() (spinnman.model.p2p_table.P2PTable method), 343
iterchips() (spinnman.model.P2PTable method), 358
iteritems() (spinnman.model.cpu_infos.CPUInfos method), 339
iteritems() (spinnman.model.CPUInfos method), 354
iterkeys() (spinnman.model.cpu_infos.CPUInfos method), 339
iterkeys() (spinnman.model.CPUInfos method), 354
itervalues() (spinnman.model.cpu_infos.CPUInfos method), 339
itervalues() (spinnman.model.CPUInfos method), 354
IzkCondExpBase (class in spynnaker.pyNN.models.neuron.builds), 721
IzkCondExpBase (class in spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base), 716
IzkCurrExpBase (class in spynnaker.pyNN.models.neuron.builds), 722
IzkCurrExpBase (class in spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base), 717

J

joins (pacman.operations.fixed_route_router.fixed_route_router.FixedRouteRouter attribute), 185
JULES_PER_MILLISECOND_FOR_BOXED_48_CHIP_FRAME_IDLE_COST (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_PER_MILLISECOND_FOR_FRAME_IDLE_COST (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_PER_MILLISECOND_PER_CHIP_ACTIVE_OVERHEAD (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_PER_MILLISECOND_PER_FPGA (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_PER_MILLISECOND_PER_FRAME_ACTIVE_COST (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_PER_MILLISECOND_PER_IDLE_CHIP (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_PER_MILLISECOND_PER_UNBOXED_48_CHIP_FRAME_IDLE_COST (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_PER_SPIKE (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564
JULES_TO_KILLIWATT_HOURS (spinn_front_end_common.utilities.report_functions.energy_report.EnergyReport attribute), 564

K

key (pacman.model.routing_info.base_key_and_mask.BaseKeyAndMask attribute), 165
key (pacman.model.routing_info.BaseKeyAndMask attribute), 168
key (pacman.operations.router_compressors.mundys_router_compressor.rou attribute), 191
key (pacman.operations.router_compressors.mundys_router_compressor.rou attribute), 191
key (spinn_front_end_common.utility_models.multi_cast_command.MultiC attribute), 598
key (spinn_front_end_common.utility_models.MultiCastCommand attribute), 609
key (spinnman.messages.eieio.data_messages.key_data_element.KeyDataE attribute), 261
key (spinnman.messages.eieio.data_messages.key_payload_data_element.K attribute), 261
key (spinnman.messages.eieio.data_messages.KeyDataElement attribute), 265
key (spinnman.messages.eieio.data_messages.KeyPayloadDataElement attribute), 265
key (spinnman.messages.multicast_message.MulticastMessage attribute), 324
KEY_16_BIT (spinnman.messages.eieio.eieio_type.EIEIOType attribute), 267
KEY_16_BIT (spinnman.messages.eieio.EIEIOType attribute), 267
KEY_32_BIT (spinnman.messages.eieio.eieio_type.EIEIOType attribute), 267
KEY_32_BIT (spinnman.messages.eieio.EIEIOType attribute), 267
key_bytes (spinnman.messages.eieio.eieio_type.EIEIOType attribute), 267
key_bytes (spinnman.messages.eieio.EIEIOType attribute), 267
key_combo (pacman.model.routing_info.base_key_and_mask.BaseKeyAnd attribute), 165
key_combo (pacman.model.routing_info.BaseKeyAndMask attribute), 168
key_list_function (pacman.model.constraints.key_allocator_constraints.fixe attribute), 99
key_list_function (pacman.model.constraints.key_allocator_constraints.Fixe attribute), 101
KEY_PAYLOAD_16_BIT (spinnman.messages.eieio.eieio_type.EIEIOType attribute), 267

KEY_PAYLOAD_16_BIT (spinnman.messages.eieio.EIEIOType attribute), 267
KEY_PAYLOAD_32_BIT (spinnman.messages.eieio.eieio_type.EIEIOType attribute), 267
KEY_PAYLOAD_32_BIT (spinnman.messages.eieio.EIEIOType attribute), 267
key_prefix (spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.LivePacketGatherParameters attribute), 573
key_prefix (spinn_front_end_common.utilities.utility_objs.LivePacketGatherParameters attribute), 576
KeyDataElement (class in spinnman.messages.eieio.data_messages), 265
KeyDataElement (class in spinnman.messages.eieio.data_messages.key_data_element), 261
KeyFieldGenerator (class in pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_info_allocator.malloc_key_field_generator), 193
KeyPayloadDataElement (class in spinnman.messages.eieio.data_messages), 265
KeyPayloadDataElement (class in spinnman.messages.eieio.data_messages.key_payload_data_element), 261
keys (pacman.utilities.utility_objs.resource_tracker.ResourceTracker attribute), 203
keys (pacman.utilities.utility_objs.ResourceTracker attribute), 207
keys_and_masks (pacman.model.constraints.key_allocator_constraints.fixed_key_and_mask_constraint.FixedKeyAndMaskConstraint attribute), 99
keys_and_masks (pacman.model.constraints.key_allocator_constraints.FixedKeyAndMaskConstraint attribute), 101
keys_and_masks (pacman.model.routing_info.partition_routing_info.PartitionRoutingInfo attribute), 166
keys_and_masks (pacman.model.routing_info.PartitionRoutingInfo attribute), 169

L

label (pacman.model.graphs.abstract_edge.AbstractEdge attribute), 137
label (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 139
label (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition attribute), 140
label (pacman.model.graphs.abstract_vertex.AbstractVertex attribute), 141
label (pacman.model.graphs.AbstractEdge attribute), 142
label (pacman.model.graphs.AbstractGraph attribute), 145
label (pacman.model.graphs.AbstractOutgoingEdgePartition attribute), 146
label (pacman.model.graphs.AbstractVertex attribute), 147
label (pacman.model.graphs.application.application_edge.ApplicationEdge attribute), 109
label (pacman.model.graphs.application.application_vertex.ApplicationVertex attribute), 113
label (pacman.model.graphs.application.ApplicationEdge attribute), 114
label (pacman.model.graphs.application.ApplicationVertex attribute), 117
label (pacman.model.graphs.impl.Graph attribute), 128
label (pacman.model.graphs.impl.graph.Graph attribute), 124
label (pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition attribute), 126
label (pacman.model.graphs.impl.OutgoingEdgePartition attribute), 129
label (pacman.model.graphs.machine.machine_edge.MachineEdge attribute), 130
label (pacman.model.graphs.machine.machine_vertex.MachineVertex attribute), 133
label (pacman.model.graphs.machine.MachineEdge attribute), 134
label (pacman.model.graphs.machine.MachineVertex attribute), 136
label (spynnaker.pyNN.models.pynn_population_common.PyNNPopulation attribute), 815
largest_free_sdram_block (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
largest_free_sdram_block (spinnman.model.ChipSummaryInfo attribute), 350
largest_free_sram_block (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
largest_free_sram_block (spinnman.model.ChipSummaryInfo attribute), 350
LASER_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters attribute), 639
LASER_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters attribute), 640
laser_active_time() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
laser_active_time() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
LASER_FREQUENCY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters attribute), 639

LASER_FREQUENCY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 640
laser_frequency() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
laser_frequency() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
LASER_TOTAL_PERIOD (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 639
LASER_TOTAL_PERIOD (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 640
laser_total_period() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
laser_total_period() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
last_biff_id (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
last_biff_id (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
last_buffer_operation (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState attribute), 515
last_buffer_operation (spinn_front_end_common.interface.buffer_management.storage_objects.ChannelBufferState attribute), 523
last_id (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 815
last_received_packet_from_core() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 510
last_received_packet_from_core() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 518
last_sent_packet_to_core() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 511
last_sent_packet_to_core() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 519
last_sequence_no_for_core() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 511
last_sequence_no_for_core() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 519
led_0 (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
led_0 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
led_1 (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
led_1 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
LED_BACK_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 639
LED_BACK_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 640
led_back_active_time() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
led_back_active_time() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
LED_FREQUENCY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 639
LED_FREQUENCY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 640
led_frequency() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 817
led_frequency() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
LED_FRONT_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 639
LED_FRONT_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 640
led_front_active_time() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
led_front_active_time() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
led_half_period_10_ms (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
led_half_period_10_ms (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
LED_TOTAL_PERIOD (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 639
LED_TOTAL_PERIOD (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLaser attribute), 640
led_total_period() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
led_total_period() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol

static method), 823
LEDAction (class in spinnman.messages.scp.enums), 279
LEDAction (class in spinnman.messages.scp.enums.led_action), 275
LEFT_RETINA (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
LEFT_RETINA (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
LEFT_RETINA_DISABLE (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
LEFT_RETINA_DISABLE (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
LEFT_RETINA_ENABLE (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
LEFT_RETINA_ENABLE (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
LEFT_RETINA_KEY_SET (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
LEFT_RETINA_KEY_SET (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
LEFT_SHIFT (data_specification.enums.logic_operation.LogicOperation attribute), 417
LEFT_SHIFT (data_specification.enums.LogicOperation attribute), 420
LESS_THAN (data_specification.enums.Condition attribute), 419
LESS_THAN (data_specification.enums.condition.Condition attribute), 416
LESS_THAN_OR_EQUAL (data_specification.enums.Condition attribute), 419
LESS_THAN_OR_EQUAL (data_specification.enums.condition.Condition attribute), 416
level_of_deepest_parent() (spinn_utilities.log.ConfiguredFormatter static method), 10
line_plot() (in module spynnaker.plot_utils), 838
Link (class in spinn_machine), 45
Link (class in spinn_machine.link), 28
LINK_0 (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LINK_0 (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
LINK_1 (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LINK_1 (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
LINK_2 (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LINK_2 (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
LINK_3 (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LINK_3 (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
LINK_4 (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LINK_4 (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
LINK_5 (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LINK_5 (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
LINK_ADD_TABLE (spinn_machine.Machine attribute), 47
LINK_ADD_TABLE (spinn_machine.machine.Machine attribute), 29
link_ids (spinn_machine.fixed_route_entry.FixedRouteEntry attribute), 38
link_ids (spinn_machine.FixedRouteEntry attribute), 45
link_ids (spinn_machine.multicast_routing_entry.MulticastRoutingEntry attribute), 34
link_ids (spinn_machine.MulticastRoutingEntry attribute), 51
link_peek_timeout_microseconds (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 318
link_peek_timeout_microseconds (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
link_register (spinnman.model.cpu_info.CPUInfo attribute), 336
link_register (spinnman.model.CPUInfo attribute), 351
links (spinn_machine.Router attribute), 54
links (spinn_machine.router.Router attribute), 37
links_available (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 318
links_available (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
links_available (spinnman.model.chip_info.ChipInfo attribute), 333
links_available (spinnman.model.ChipInfo attribute), 348
listen_port (spinn_front_end_common.utilities.notification_protocol.socket_addresses.SocketAddressWithChecksum attribute), 563
listen_port (spinn_front_end_common.utilities.notification_protocol.SocketAddressWithChecksum attribute), 564
listen_port (spinn_utilities.socket_address.SocketAddress attribute), 13
Listenable (class in spinn-

man.connections.abstract_classes), 226
Listenable (class in spinnman.connections.abstract_classes.listenable), 220
LiveEventConnection (class in spinn_front_end_common.utilities.connections), 553
LiveEventConnection (class in spinn_front_end_common.utilities.connections.live_event_connection), 551
LivePacketGather (class in spinn_front_end_common.utility_models), 606
LivePacketGather (class in spinn_front_end_common.utility_models.live_packet_gather), 594
LivePacketGatherMachineVertex (class in spinn_front_end_common.utility_models), 607
LivePacketGatherMachineVertex (class in spinn_front_end_common.utility_models.live_packet_gather_machine_vertex), 596
LivePacketGatherParameters (class in spinn_front_end_common.utilities.utility_objs), 576
LivePacketGatherParameters (class in spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters), 573
lo (pacman.utilities.utility_objs.Field attribute), 203
lo (pacman.utilities.utility_objs.field.Field attribute), 199
load_config() (in module spinn_utilities.conf_loader), 8
load_fixed_route() (spinnman.processes.load_fixed_route_routing_entry_process.LoadFixedRouteRoutingEntryProcess method), 364
load_fixed_route() (spinnman.processes.LoadFixedRouteRoutingEntryProcess method), 372
load_fixed_route() (spinnman.transceiver.Transceiver method), 397
load_initial_buffers() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 525
load_initial_buffers() (spinn_front_end_common.interface.buffer_management.buffer_manager method), 529
load_iptags() (spinn_front_end_common.interface.interface_functions.load_iptags static method), 536
load_iptags() (spinn_front_end_common.interface.interface_functions.load_iptags static method), 540
load_module() (in module spinn_utilities.package_loader), 11
load_modules() (in module spinn_utilities.package_loader), 11
load_multicast_routes() (spinnman.transceiver.Transceiver method), 397
load_reverse_iptags() (spinn_front_end_common.interface.interface_functions.load_reverse_iptags static method), 536
load_reverse_iptags() (spinn_front_end_common.interface.interface_functions.load_reverse_iptags static method), 540
load_routes() (spinnman.processes.load_routes_process.LoadMultiCastRoutesProcess method), 365
load_routes() (spinnman.processes.LoadMultiCastRoutesProcess method), 369
LoadExecutableImages (class in spinn_front_end_common.interface.interface_functions), 538
LoadExecutableImages (class in spinn_front_end_common.interface.interface_functions.load_executable_images), 533
LoadFixedRouteRoutingEntryProcess (class in spinnman.processes), 372
LoadFixedRouteRoutingEntryProcess (class in spinnman.processes.load_fixed_route_routing_entry_process), 364
LoadFixedRoutes (class in spinn_front_end_common.interface.interface_functions.load_fixed_routes), 538
LoadMultiCastRoutesProcess (class in spinnman.processes), 369
LoadMultiCastRoutesProcess (class in spinnman.processes.load_routes_process), 365
LOCAL (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LOCAL (spinnman.model.enums.diagnostic_filter_source.DiagnosticFilterSource attribute), 327
LOCAL (spinnman.model.enums.DiagnosticFilterDestination attribute), 329
LOCAL (spinnman.model.enums.DiagnosticFilterSource attribute), 331
local_host (spynnaker.pyNN.external_devices_models.push_bot.push_bot_protocol.PushBotRoutingEntryProcess attribute), 640
local_host (spynnaker.pyNN.external_devices_models.push_bot.push_bot_protocol.PushBotRoutingEntryProcess attribute), 641
local_ip_address (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection attribute), 235
local_ip_address (spinnman.transceiver.buffer_manager.BufferManager attribute), 239
local_ip_address (spinnman.connections.udp_packet_connections.UDPConnection attribute), 239
local_ip_address (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.TagsLoader attribute), 631
local_ip_address (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.TagsLoader attribute), 638
LOCAL_MONITOR (spinnman.model.enums.diagnostic_filter_destination.DiagnosticFilterDestination attribute), 326
LOCAL_MONITOR (spinnman.model.enums.DiagnosticFilterDestination

attribute), 329
local_port (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection attribute), 235
local_port (spinnman.connections.udp_packet_connections.UDPConnection attribute), 239
local_port (spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_protocol.PushBotUDPConnection attribute), 631
local_port (spynnaker.pyNN.external_devices_models.push_bot.push_bot_wifi_connection.PushBotWIFIConnection attribute), 638
local_port (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_viewer.PushBotRetinaViewer attribute), 640
local_port (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotRetinaViewer attribute), 641
local_size (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 815
locate_constraints_of_type() (in module pacman.utilities.utility_calls), 209
locate_first_constraint_of_type() (in module pacman.utilities.utility_calls), 209
locate_memory_region_for_placement() (in module spinn_front_end_common.utilities.helpful_functions), 580
locate_spinnaker_connection_for_board_address() (spinnman.transceiver.Transceiver method), 398
LocateExecutableStartType (class in spinn_front_end_common.interface.interface_functions), 538
LocateExecutableStartType (class in spinn_front_end_common.interface.interface_functions.locate_executable_start_type), 533
location (pacman.model.constraints.placer_constraints.chip_and_core_constraint.ChipAndCoreConstraint attribute), 104
location (pacman.model.constraints.placer_constraints.ChipAndCoreConstraint attribute), 106
lock (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
lock (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
log_peer_to_peer_sequence_length (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
log_peer_to_peer_sequence_length (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
logging_parser() (in module spinn_utilities.conf_loader), 8
LOGIC_OP (data_specification.enums.Commands attribute), 418
LOGIC_OP (data_specification.enums.commands.Commands attribute), 415
logical_and() (data_specification.data_specification_generator.DataSpecificationGenerator method), 434
logical_and() (data_specification.DataSpecificationGenerator method), 472
logical_left_shift() (data_specification.data_specification_generator.DataSpecificationGenerator method), 435
logical_left_shift() (data_specification.DataSpecificationGenerator method), 473
logical_not() (data_specification.data_specification_generator.DataSpecificationGenerator method), 436
logical_not() (data_specification.DataSpecificationGenerator method), 473
logical_or() (data_specification.data_specification_generator.DataSpecificationGenerator method), 436
logical_or() (data_specification.DataSpecificationGenerator method), 474
logical_right_shift() (data_specification.data_specification_generator.DataSpecificationGenerator method), 437
logical_right_shift() (data_specification.DataSpecificationGenerator method), 474
logical_xor() (data_specification.data_specification_generator.DataSpecificationGenerator method), 437
logical_xor() (data_specification.DataSpecificationGenerator method), 475
LogicOperation (class in data_specification.enums), 420
LogicOperation (class in data_specification.enums.logic_operation), 417
LOOP (data_specification.enums.Commands attribute), 418
LOOP (data_specification.enums.commands.Commands attribute), 415
low() (in module spinn_utilities.utility_calls), 831
low() (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats method), 827
low() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
LOWER_HALF_WORD (spinnman.messages.eieio.eieio_prefix.EIEIOPrefix attribute), 266
LOWER_HALF_WORD (spinnman.messages.eieio.EIEIOPrefix attribute), 267
ltpc_period (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
ltpc_period (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322

M

Machine (class in spinn_machine), 46
Machine (class in spinn_machine.machine), 29
machine (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
machine (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
machine (spinn_front_end_common.utilities.FailedState attribute), 583

machine (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 583
machine (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
machine_graph (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
machine_time_step (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
machine_time_step (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
machine_time_step (spinn_front_end_common.utilities.FailedState attribute), 583
machine_time_step (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 583
machine_time_step (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
machine_time_step() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager static method), 837
MachineDataSpecableVertex (class in spinn_front_end_common.abstract_models.impl), 494
MachineDataSpecableVertex (class in spinn_front_end_common.abstract_models.impl.machine_data_specable_vertex), 493
MachineDimensions (class in spinnman.model), 357
MachineDimensions (class in spinnman.model.machine_dimensions), 342
MachineEdge (class in pacman.model.graphs.machine), 133
MachineEdge (class in pacman.model.graphs.machine.machine_edge), 130
MachineExecuteDataSpecification (class in spinn_front_end_common.interface.interface_functions), 538
MachineExecuteDataSpecification (class in spinn_front_end_common.interface.interface_functions.machine_execute_data_specification), 533
MachineFPGAVertex (class in pacman.model.graphs.machine), 134
MachineFPGAVertex (class in pacman.model.graphs.machine.machine_fpga_vertex), 131
MachineGenerator (class in spinn_front_end_common.interface.interface_functions), 539
MachineGenerator (class in spinn_front_end_common.interface.interface_functions.machine_generator), 534
MachineGraph (class in pacman.model.graphs.machine), 135
MachineGraph (class in pacman.model.graphs.machine.machine_graph), 136
MachineOutgoingEdgePartition (class in pacman.model.graphs.machine), 135
MachineOutgoingEdgePartition (class in pacman.model.graphs.machine.machine_outgoing_edge_partition), 132
MachineSpiNNakerLinkVertex (class in pacman.model.graphs.machine), 135
MachineSpiNNakerLinkVertex (class in pacman.model.graphs.machine.machine_spinnaker_link_vertex), 132
MachineVertex (class in pacman.model.graphs.machine), 136
MachineVertex (class in pacman.model.graphs.machine.machine_vertex), 133
MAGIC_NUMBER (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 425
MAGIC_NUMBER (data_specification.DataSpecificationGenerator attribute), 463
MailboxCommand (class in spinnman.model.enums), 331
MailboxCommand (class in spinnman.model.enums.mailbox_command), 327
MALLOC (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
MALLOC (spinnman.model.enums.RunTimeError attribute), 330
malloc_sdram() (spinnman.processes.malloc_sdram_process.MallocSDRAMProcess method), 365
malloc_sdram() (spinnman.processes.MallocSDRAMProcess method), 370
malloc_sdram() (spinnman.transceiver.Transceiver method), 398
MallocBasedChipIdAllocator (class in pacman.operations.chip_id_allocator_algorithms), 185
MallocBasedChipIdAllocator (class in pacman.operations.chip_id_allocator_algorithms.malloc_based_chip_id_allocator), 184
MallocBasedRouteMerger (class in pacman.operations.router_compressors.malloc_based_route_merger), 192
MallocBasedRoutingInfoAllocator (class in pacman.operations.routing_info_allocator_algorithms), 194
MallocBasedRoutingInfoAllocator (class in pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_info_allocator), 193
MallocSDRAMProcess (class in spinnman.processes), 369
MallocSDRAMProcess (class in spinnman.processes.malloc_sdram_process), 365

MANAGEMENT_BIT (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
MANAGEMENT_BIT (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
MANAGEMENT_MASK (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
MANAGEMENT_MASK (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
mark_no_changes() (spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun method), 495
mark_no_changes() (spinn_front_end_common.abstract_models.AbstractChangableAfterRun method), 499
mark_no_changes() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 782
mark_no_changes() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 789
mark_no_changes() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
mark_no_changes() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 760
mark_no_changes() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 763
mark_no_changes() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
mark_no_changes() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 815
mark_no_changes() (spynnaker.pyNN.models.pynn_projection_common.PyNNProjectionCommon method), 816
mark_no_changes() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray method), 794
mark_no_changes() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 797
mark_no_changes() (spynnaker.pyNN.models.spike_source.SpikeSourceArray method), 801
mark_no_changes() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 804
mark_regions_reloaded() (spinn_front_end_common.abstract_models.abstract_rewrites_data_specification.AbstractRewritesDataSpecification method), 498
mark_regions_reloaded() (spinn_front_end_common.abstract_models.AbstractRewritesDat method), 501
mark_regions_reloaded() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP method), 782
mark_regions_reloaded() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 790
mark_regions_reloaded() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSou method), 798
mark_regions_reloaded() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 804
mask (pacman.model.constraints.key_allocator_constraints.fixed_mask_con attribute), 100
mask (pacman.model.constraints.key_allocator_constraints.FixedMaskCons attribute), 102
mask (pacman.model.routing_info.base_key_and_mask.BaseKeyAndMask attribute), 165
mask (pacman.model.routing_info.BaseKeyAndMask attribute), 168
mask (pacman.operations.router_compressors.mundys_router_compressor.r attribute), 191
mask (pacman.operations.router_compressors.mundys_router_compressor.r attribute), 191
MASK (pacman.operations.routing_info_allocator_algorithms.destination_ attribute), 193
mask (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_s attribute), 603
mask (spinn_front_end_common.utility_models.ReverseIPTagMulticastSou attribute), 614
mask (spinn_machine.multicast_routing_entry.MulticastRoutingEntry attribute), 34
mask (spinn_machine.MulticastRoutingEntry attribute), 51
MASTER_POP_ENTRY_DTYPE (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ attribute), 730
MASTER_POP_ENTRY_DTYPE (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterI attribute), 732
master_slave_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 819
master_slave_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
master_slave_set_master_clock_active() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic method), 819
master_slave_set_master_clock_active() (spyn-

naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
master_slave_set_master_clock_not_started() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
master_slave_set_master_clock_not_started() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
master_slave_set_slave() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
master_slave_set_slave() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
master_slave_use_internal_counter() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
master_slave_use_internal_counter() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
MasterPopTableAs2dArray (class in spynnaker.pyNN.models.neuron.master_pop_table_generators), 731
MasterPopTableAs2dArray (class in spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_2d_array), 729
MasterPopTableAsBinarySearch (class in spynnaker.pyNN.models.neuron.master_pop_table_generators), 732
MasterPopTableAsBinarySearch (class in spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search), 730
match_emergency_routing_status_to_incoming_packet (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
match_emergency_routing_status_to_incoming_packet (spinnman.model.DiagnosticFilter attribute), 355
MAX_BANDWIDTH_PER_ETHERNET_CONNECTED_CHIP (spinn_machine.Machine attribute), 47
MAX_BANDWIDTH_PER_ETHERNET_CONNECTED_CHIP (spinn_machine.machine.Machine attribute), 29
MAX_BUFFER_SIZE (spinn_front_end_common.utility_models.chip_power_monitor.ChipPowerMonitorMachineVertex attribute), 586
max_buffer_size_possible (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region.BufferedSendingRegion attribute), 513
max_buffer_size_possible (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedSendingRegion attribute), 521
MAX_BW (pacman.operations.router_algorithms.basic_dijkstra_routing.BasicDijkstraRouting attribute), 190
MAX_BW (pacman.operations.router_algorithms.BasicDijkstraRouting attribute), 190
max_chip_x (spinn_machine.Machine attribute), 49
max_chip_x (spinn_machine.machine.Machine attribute), 32
max_chip_y (spinn_machine.Machine attribute), 50
max_chip_y (spinn_machine.machine.Machine attribute), 32
MAX_CORES_PER_CHIP (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex attribute), 586
max_delay (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerLinkProtocol attribute), 833
max_delay (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface attribute), 837
max_delay (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState attribute), 839
MAX_KEYS_SUPPORTED (pacman.operations.routing_info_allocator_algorithms.destination_based_ri_allocator.DestinationBasedRiAllocator attribute), 193
MAX_LENGTH_IN_CHARS (spinn_machine.utilities.progress_bar.ProgressBar attribute), 22
MAX_LENGTH_IN_CHARS (spinn_utilities.progress_bar.ProgressBar attribute), 12
max_n_elements (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage attribute), 260
max_n_elements (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 268
MAX_NUMBER_OF_TIMER_TIC_OVERRUN (spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl attribute), 544
MAX_NUMBER_OF_TIMER_TIC_OVERRUN (spinn_front_end_common.interface.provenance.ProvidesProvenanceDataFromMachine attribute), 545
max_packets_in_timestamp (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region.BufferedSendingRegion attribute), 513
max_packets_in_timestamp (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedSendingRegion attribute), 521
max_size (spynnaker.pyNN.models.neuron.synaptic_matrix.SynapticMatrixWithWeightOnly attribute), 834
max_supported_length (pacman.operations.routing_info_allocator_algorithms.zoned_malloc_based_ri_allocator.ZonedMallocBasedRiAllocator attribute), 191
max_value (spinnman.messages.eieio.eieio_type.EIEIOType attribute), 267
max_value (spinnman.messages.eieio.EIEIOType attribute), 267
max_value (spynnaker.pyNN.external_devices_models.push_bot.abstract_protocol.AbstractPushBotCommandValue attribute), 649

max_value (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotOutputDevice attribute), 649
max_write_pointer (data_specification.memory_region.MemoryRegion attribute), 455
max_write_pointer (data_specification.MemoryRegion attribute), 488
maximum_user_cores_on_chip (spinn_machine.Machine attribute), 50
maximum_user_cores_on_chip (spinn_machine.machine.Machine attribute), 32
maximum_user_cores_on_chip (spinn_machine.virtual_machine.VirtualMachine attribute), 40
maximum_user_cores_on_chip (spinn_machine.VirtualMachine attribute), 56
MaxVertexAtomsConstraint (class in pacman.model.constraints.partitioner_constraints), 103
MaxVertexAtomsConstraint (class in pacman.model.constraints.partitioner_constraints.maxVertexAtomsConstraint), 102
mean (spynnaker.pyNN.utilities.running_stats.RunningStats attribute), 829
mean() (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats method), 827
mean() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
measured_interval (spinn_utilities.timer.Timer attribute), 13
mem_regions (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
mem_regions (data_specification.DataSpecificationExecutorFunctions attribute), 462
mem_slot (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 438
mem_slot (data_specification.DataSpecificationGenerator attribute), 476
mem_writer (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
mem_writer (data_specification.DataSpecificationExecutorFunctions attribute), 462
memory_pointer (data_specification.memory_region.MemoryRegion attribute), 455
memory_pointer (data_specification.MemoryRegion attribute), 488
memory_space (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
memory_space (data_specification.DataSpecificationExecutorFunctions attribute), 462
MemoryIO (class in spinnman.utilities.io), 377
MemoryIO (class in spinnman.utilities.io.memory_io), 374
MemoryMapOnChipReportDevice (class in spinn_front_end_common.utilities.report_functions), 565
MemoryMapOnChipReport (class in spinn_front_end_common.utilities.report_functions.memory_map 564
MemoryMapOnHostChipReport (class in spinn_front_end_common.utilities.report_functions), 565
MemoryMapOnHostChipReport (class in spinn_front_end_common.utilities.report_functions.memory_map 565
MemoryMapOnHostReport (class in spinn_front_end_common.utilities.report_functions), 565
MemoryMapOnHostReport (class in spinn_front_end_common.utilities.report_functions.memory_map 565
MemoryRegion (class in data_specification), 488
MemoryRegion (class in data_specification.memory_region), 455
MemoryRegionCollection (class in data_specification), 488
MemoryRegionCollection (class in data_specification.memory_region_collection), 456
MemReadException, 834
merge() (spinn_machine.multicast_routing_entry.MulticastRoutingEntry method), 34
merge() (spinn_machine.MulticastRoutingEntry method), 51
merge_entry() (pacman.model.routing_table_by_partition.multicast_routing method), 172
merge_entry() (pacman.model.routing_table_by_partition.MulticastRouting method), 173
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fp attribute), 654
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevic attribute), 662
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_ret attribute), 657
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
MERSENNE_TWISTER (data_specification.enums.random_number_generator.RandomNu attribute), 417
MERSENNE_TWISTER (data_specification.enums.RandomNumberGenerator attribute), 420
message (spinn_front_end_common.utilities.utility_objs.provenance_data_i

attribute), 574
message (spinn_front_end_common.utilities.utility_objs.Provenance.Item attribute), 577
message_type (spinn_front_end_common.utilities.utility_objs.Provenance.Item attribute), 573
message_type (spinn_front_end_common.utilities.utility_objs.LivePacketGather attribute), 576
messages (spinn_front_end_common.interface.buffer_management.storage_objects.buffers_sent_deque.BuffersSentDeque attribute), 514
messages (spinn_front_end_common.interface.buffer_management.storage_objects.BuffersSentDeque attribute), 522
messages (spinnman.messages.spinnaker_boot.spinnaker_boot_messages.SpinnakerBootMessages attribute), 316
messages (spinnman.messages.spinnaker_boot.SpinnakerBootMessages attribute), 321
min_delay (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon attribute), 833
min_delay (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface attribute), 837
min_delay (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState attribute), 829
min_packet_length() (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage static method), 260
min_packet_length() (spinnman.messages.eieio.data_messages.EIEIODataMessage static method), 264
min_value (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice attribute), 649
min_value (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotOutputDevice attribute), 649
missing_info (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState attribute), 516
missing_info (spinn_front_end_common.interface.buffer_management.storage_objects.ChannelBufferState attribute), 523
mode (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
mode (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
MODE_128 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARetinaDevice attribute), 654
MODE_128 (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
MODE_16 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARetinaDevice attribute), 654
MODE_16 (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
MODE_32 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARetinaDevice attribute), 654
MODE_32 (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
MODE_64 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARetinaDevice attribute), 654
MODE_64 (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
mem (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 343
mon (spinnman.model.RouterDiagnostics attribute), 358
MONITOR (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
MONITOR (spinnman.model.enums.P2PTableRoute attribute), 331
monitor_mailbox_command (spinnman.model.cpu_info.CPUInfo attribute), 336
monitor_mailbox_command (spinnman.model.CPUInfo attribute), 351
monitor_mailbox_data_address (spinnman.model.cpu_info.CPUInfo attribute), 336
monitor_mailbox_data_address (spinnman.model.CPUInfo attribute), 352
monitor_mailbox_flags (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
monitor_mailbox_flags (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
MostDirectConnectionSelector (class in spinnman.processes), 370
MostDirectConnectionSelector (class in spinnman.processes.output_device.AbstractPushBotOutputDevice), 365
MOTOR_0_LEAKY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 639
MOTOR_0_LEAKY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 641
motor_0_leaky_velocity() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEth static method), 818
motor_0_leaky_velocity() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
MOTOR_0_PERMANENT (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 639
MOTOR_0_PERMANENT (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parame attribute), 641
motor_0_permanent_velocity() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEth static method), 818
motor_0_permanent_velocity() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
MOTOR_1_LEAKY (spyn-

naker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_motor.PushBotMotor attribute), 639
MOTOR_1_LEAKY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotMotor attribute), 641
motor_1_leaky_velocity() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
motor_1_leaky_velocity() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
MOTOR_1_PERMANENT (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotMotor attribute), 639
MOTOR_1_PERMANENT (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotMotor attribute), 641
motor_1_permanent_velocity() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
motor_1_permanent_velocity() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
move_forward() (pacman.operations.rigged_algorithms.hilbert_state.HilbertState method), 189
MultapseConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 693
MultapseConnector (class in spynnaker.pyNN.models.neural_projections.connectors.multapse_connector), 684
MULTICAST (pacman.model.graphs.common.edge_traffic_type.EdgeTrafficType attribute), 119
MULTICAST (pacman.model.graphs.common.EdgeTrafficType attribute), 121
MULTICAST (spinn_front_end_common.utilities.utility_objs.dpri_flags.DPRIFlags attribute), 572
MULTICAST (spinnman.messages.scp.enums.signal.SignalType attribute), 277
MULTICAST (spinnman.model.enums.diagnostic_filter_packet_type.DiagnosticFilterPacketType attribute), 326
MULTICAST (spinnman.model.enums.DiagnosticFilterPacketType attribute), 330
multicast_default_from (spinn_machine.Link attribute), 46
multicast_default_from (spinn_machine.link.Link attribute), 29
multicast_default_to (spinn_machine.Link attribute), 46
multicast_default_to (spinn_machine.link.Link attribute), 29
multicast_routing_entries (pacman.model.routing_tables.multicast_routing_table.MultiSpikeRecorder attribute), 174
multicast_routing_entries (pacman.model.routing_tables.multicast_routing_table.MultiSpikeRecorder attribute), 176
MultiCastCommand (class in spinn_front_end_common.utility_models), 608
MultiCastCommand (class in spinn_front_end_common.utility_models.multi_cast_command), 597
MulticastMessage (class in spinnman.messages.multicast_message), 324
MulticastReceiver (class in spinnman.connections.abstract_classes), 226
MulticastReceiver (class in spinnman.connections.abstract_classes.multicast_receiver), 220
MulticastRoutingEntry (class in spinn_machine), 50
MulticastRoutingEntry (class in spinn_machine.multicast_routing_entry), 33
MulticastRoutingTable (class in pacman.model.routing_tables), 176
MulticastRoutingTable (class in pacman.model.routing_tables.multicast_routing_table), 174
MulticastRoutingTableByPartition (class in pacman.model.routing_table_by_partition), 172
MulticastRoutingTableByPartition (class in pacman.model.routing_table_by_partition.multicast_routing_table_b 170
MulticastRoutingTableByPartitionEntry (class in pacman.model.routing_table_by_partition), 172
MulticastRoutingTableByPartitionEntry (class in pacman.model.routing_table_by_partition.multicast_routing_table_b 171
MulticastRoutingTables (class in pacman.model.routing_tables), 177
MulticastRoutingTables (class in pacman.model.routing_tables.multicast_routing_tables), 175
MulticastSender (class in spinnman.connections.abstract_classes), 226
MulticastSender (class in spinnman.connections.abstract_classes.multicast_sender), 221
MULTIPLY (data_specification.enums.arithemetic_operation.ArithmeticOp attribute), 414
MULTIPLY (data_specification.enums.ArithmeticOperation attribute), 417
MultiSpikeRecorder (class in spynnaker.pyNN.models.common), 675
MultiSpikeRecorderTable (class in spynnaker.pyNN.models.common.multi_spike_recorder), 672

MundyOnChipRouterCompression (class in spinn_front_end_common.mapping_algorithms.on_chip_router_compression.mundy_on_chip_router_compression), 550
MundyRouterCompressor (class in pacman.operations.router_compressors.mundys_router_compressor.routing_table_condenser), 191
MundyRouterCompressor.KeyMask (class in pacman.operations.router_compressors.mundys_router_compressor.routing_table_condenser), 191
MundyRouterCompressor.RoutingEntry (class in pacman.operations.router_compressors.mundys_router_compressor.routing_table_condenser), 191
munich_key() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 822
munich_key_i() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 822
munich_key_i_d() (in module spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 822
MunichIoEthernetProtocol (class in spynnaker.pyNN.protocols), 822
MunichIoEthernetProtocol (class in spynnaker.pyNN.protocols.munich_io_ethernet_protocol), 817
MunichIoSpiNNakerLinkProtocol (class in spynnaker.pyNN.protocols), 823
MunichIoSpiNNakerLinkProtocol (class in spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 818
MunichIoSpiNNakerLinkProtocol.MODES (class in spynnaker.pyNN.protocols), 823
MunichIoSpiNNakerLinkProtocol.MODES (class in spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 818
MunichMotorDevice (class in spynnaker.pyNN.external_devices_models), 662
MunichMotorDevice (class in spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device), 655
MunichRetinaDevice (class in spynnaker.pyNN.external_devices_models), 664
MunichRetinaDevice (class in spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device), 657
MV (data_specification.enums.Commands attribute), 418
MV (data_specification.enums.commands.Commands attribute), 415
MY_ORO_BOTICS (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES attribute), 823
MY_ORO_BOTICS (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES attribute), 818

N

n_active_peer_to_peer_addresses (spinnman.messages.spinnaker_boot.system_variable_boot_values.Syst attribute), 318
n_active_peer_to_peer_addresses (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 322
n_ADDITIONAL_PROVENANCE_DATA_ITEMS (spynnaker.pyNN.models.neuron.population_machine_vertex.Pop attribute), 784
n_ADDITIONAL_PROVENANCE_DATA_ITEMS (spynnaker.pyNN.models.neuron.PopulationMachineVertex attribute), 792
n_ADDITIONAL_PROVENANCE_ITEMS (spinn_front_end_common.utility_models.live_packet_gather_ma attribute), 596
n_ADDITIONAL_PROVENANCE_ITEMS (spinn_front_end_common.utility_models.LivePacketGatherMach attribute), 608
n_atoms (pacman.model.graphs.application.application_fpga_vertex.Applic attribute), 111
n_atoms (pacman.model.graphs.application.application_spinnaker_link_ver attribute), 112
n_atoms (pacman.model.graphs.application.application_vertex.ApplicationV attribute), 114
n_atoms (pacman.model.graphs.application.ApplicationFPGAVertex attribute), 115
n_atoms (pacman.model.graphs.application.ApplicationSpiNNakerLinkVert attribute), 118
n_atoms (pacman.model.graphs.application.ApplicationVertex attribute), 117
n_atoms (spinn_front_end_common.utility_models.chip_power_monitor_ap attribute), 585
n_atoms (spinn_front_end_common.utility_models.command_sender.Comm attribute), 589
n_atoms (spinn_front_end_common.utility_models.CommandSender attribute), 605
n_atoms (spinn_front_end_common.utility_models.extra_monitor_support_ attribute), 591
n_atoms (spinn_front_end_common.utility_models.live_packet_gather.Live attribute), 595
n_atoms (spinn_front_end_common.utility_models.LivePacketGather attribute), 607
n_atoms (spinn_front_end_common.utility_models.reverse_ip_tag_multi_ca attribute), 601
n_atoms (spinn_front_end_common.utility_models.ReverseIpTagMultiCast attribute), 612
n_atoms (spynnaker.pyNN.external_devices_models.munich_spinnaker_lin attribute), 656
n_atoms (spynnaker.pyNN.external_devices_models.MunichMotorDevice attribute), 664

n_atoms (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 782
n_atoms (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 790
n_atoms (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson attribute), 798
n_atoms (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 804
n_atoms (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex attribute), 808
n_atoms (spynnaker.pyNN.models.utility_models.DelayExtensionVertex attribute), 812
n_available_multicast_entries (spinn_machine.Router attribute), 54
n_available_multicast_entries (spinn_machine.router.Router attribute), 37
N_BUFFER_OVERFLOWS (spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex.DelayExtensionMachineVertex attribute), 806
N_BUFFER_OVERFLOWS (spynnaker.pyNN.models.utility_models.DelayExtensionMachineVertex attribute), 810
N_BYTES_PER_NEURON (spynnaker.pyNN.models.common.abstract_uint32_recorder.AbstractUInt32Recorder attribute), 670
N_BYTES_PER_NEURON (spynnaker.pyNN.models.common.AbstractUInt32Recorder attribute), 674
n_channels (spinnman.connections.scp_request_pipeline.SCPRequestPipeLine attribute), 244
n_channels (spinnman.connections.SCPRequestPipeLine attribute), 246
n_chips (spinn_machine.Machine attribute), 50
n_chips (spinn_machine.machine.Machine attribute), 32
n_chips (spinn_machine.virtual_machine.VirtualMachine attribute), 40
n_chips (spinn_machine.VirtualMachine attribute), 56
n_cores (pacman.model.resources.core_resource.CoreResource attribute), 152
n_cores (pacman.model.resources.CoreResource attribute), 163
n_cores (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
n_cores (spinnman.model.ChipSummaryInfo attribute), 350
N_CPU_CYCLES_PER_NEURON (spynnaker.pyNN.models.common.abstract_uint32_recorder.AbstractUInt32Recorder attribute), 671
N_CPU_CYCLES_PER_NEURON (spynnaker.pyNN.models.common.AbstractUInt32Recorder attribute), 674
n_delay_stages (spynnaker.pyNN.models.neural_projections.projection_vertex.ProjectionApplicationEdge attribute), 698
n_delay_stages (spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge attribute), 700
n_delay_stages (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex attribute), 808
n_delay_stages (spynnaker.pyNN.models.utility_models.DelayExtensionVertex attribute), 812
N_DELAYS (spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex.DelayExtensionMachineVertex attribute), 806
N_DELAYS (spynnaker.pyNN.models.utility_models.DelayExtensionMachineVertex attribute), 810
n_dropped_fixed_route_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_dropped_fixed_route_packets (spinnman.model.RouterDiagnostics attribute), 358
n_dropped_multicast_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_dropped_multicast_packets (spinnman.model.RouterDiagnostics attribute), 358
n_dropped_nearest_neighbour_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_dropped_nearest_neighbour_packets (spinnman.model.RouterDiagnostics attribute), 358
n_dropped_packet_overflows (spinn_front_end_common.utilities.utility_objs.re_injection_status.ReInjectionStatus attribute), 575
n_dropped_packets (spinn_front_end_common.utilities.utility_objs.re_injection_status.ReInjectionStatus attribute), 575
n_dropped_peer_to_peer_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_dropped_peer_to_peer_packets (spinnman.model.RouterDiagnostics attribute), 358
n_edges (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition attribute), 141
n_edges (pacman.model.graphs.AbstractOutgoingEdgePartition attribute), 146
n_edges (pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition attribute), 126
n_edges (pacman.model.graphs.impl.OutgoingEdgePartition attribute), 129
n_elements (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage attribute), 261
n_elements (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 265
n_external_fixed_route_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344

n_external_fixed_route_packets (spinnman.model.RouterDiagnostics attribute), 359
n_external_multicast_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_external_multicast_packets (spinnman.model.RouterDiagnostics attribute), 359
n_external_nearest_neighbour_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_external_nearest_neighbour_packets (spinnman.model.RouterDiagnostics attribute), 359
n_external_peer_to_peer_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_external_peer_to_peer_packets (spinnman.model.RouterDiagnostics attribute), 359
n_free_multicast_routing_entries (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
n_free_multicast_routing_entries (spinnman.model.ChipSummaryInfo attribute), 350
n_items (spynnaker.pyNN.utilities.running_stats.RunningStats attribute), 829
n_keys (pacman.model.routing_info.base_key_and_mask.BaseKeyAndMask attribute), 165
n_keys (pacman.model.routing_info.BaseKeyAndMask attribute), 168
n_keys_for_partition() (pacman.model.routing_info.abstract_machine_partition_n_keys_map.AbstractMachinePartitionNKeysMap method), 164
n_keys_for_partition() (pacman.model.routing_info.AbstractMachinePartitionNKeysMap method), 167
n_keys_for_partition() (pacman.model.routing_info.dict_based_machine_partition_n_keys_map.DictBasedMachinePartitionNKeysMap method), 165
n_keys_for_partition() (pacman.model.routing_info.DictBasedMachinePartitionNKeysMap method), 168
n_link_dumps (spinn_front_end_common.utilities.utility_objs.re_injection_status.ReInjectionStatus attribute), 575
n_local_fixed_route_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_local_fixed_route_packets (spinnman.model.RouterDiagnostics attribute), 359
n_local_multicast_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 344
n_local_multicast_packets (spinnman.model.RouterDiagnostics attribute), 359
n_local_nearest_neighbour_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
n_local_nearest_neighbour_packets (spinnman.model.RouterDiagnostics attribute), 359
n_local_peer_to_peer_packets (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
n_local_peer_to_peer_packets (spinnman.model.RouterDiagnostics attribute), 359
n_missed_dropped_packets (spinn_front_end_common.utilities.utility_objs.re_injection_statu attribute), 575
N_MONITORS_ACTIVE_DURING_COMMS (spinn_front_end_common.utilities.report_functions.energy_repo attribute), 564
n_neurons (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol attribute), 822
n_neurons (spynnaker.pyNN.protocols.RetinaKey attribute), 827
n_outgoing_edge_partitions (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 139
n_outgoing_edge_partitions (pacman.model.graphs.AbstractGraph attribute), 145
n_outgoing_edge_partitions (pacman.model.graphs.abstract_machine_graph.AbstractMachineGraph attribute), 128
n_outgoing_edge_partitions (pacman.model.graphs.impl.graph.Graph attribute), 125
N_PACKETS_ADDED (spynaker.pyNN.models.utility_models.delay_extension_machine_ver attribute), 806
N_PACKETS_ADDED (spynaker.pyNN.models.utility_models.DelayExtensionMachineVerte attribute), 810
N_PACKETS_PROCESSED (spynaker.pyNN.models.utility_models.delay_extension_machine_ver attribute), 806
N_PACKETS_PROCESSED (spynaker.pyNN.models.utility_models.DelayExtensionMachineVerte attribute), 810
N_PACKETS_RECEIVED (spynaker.pyNN.models.utility_models.delay_extension_machine_ver attribute), 807

N_PACKETS_RECEIVED (spynnaker.pyNN.models.utility_models.DelayExtensionMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES attribute), 810
N_PACKETS_SENT (spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex.DelayExtensionMachineVertex.EXTRA_PROVENANC attribute), 807
N_PACKETS_SENT (spynnaker.pyNN.models.utility_models.DelayExtensionMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES attribute), 810
n_payload_bytes (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload attribute), 822
n_payload_bytes (spynnaker.pyNN.protocols.RetinaPayload attribute), 827
n_placements (pacman.model.placements.Placements attribute), 152
n_placements (pacman.model.placements.placements.Placements attribute), 149
n_processor_dumps (spinn_front_end_common.utilities.utility_objs.reinjection_status.ReInjectionStatus attribute), 575
n_processors (spinn_machine.Chip attribute), 42
n_processors (spinn_machine.chip.Chip attribute), 23
N_RECORDING_REGIONS (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 779
N_RECORDING_REGIONS (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 786
n_regions_to_allocate() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMul static method), 603
n_regions_to_allocate() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex static method), 614
n_reinjected_packets (spinn_front_end_common.utilities.utility_objs.reinjection_status.ReInjectionStatus attribute), 575
n_requests (spinnman.messages.eieio.command_messages.hostdata_test.HostDataRead attribute), 250
n_requests (spinnman.messages.eieio.command_messages.HostDataRead attribute), 255
n_requests (spinnman.messages.eieio.command_messages.spinnaker_requests.SpinnakerRequestReadData attribute), 253
n_requests (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData attribute), 257
n_resent (spinnman.connections.scp_request_pipeline.SCPRequestPipeLine attribute), 244
n_resent (spinnman.connections.SCPRequestPipeLine attribute), 246
n_retry_code_resent (spinnman.connections.scp_request_pipeline.SCPRequestPipeLine attribute), 244
n_retry_code_resent (spinnman.connections.SCPRequestPipeLine attribute), 246
n_samples_per_recording (spinn_front_end_common.utility_models.chip_power_monitor_r attribute), 587
n_scamp_working_cores (spinnman.messages.spinnaker_boot.system_variable_boot_values.Syst attribute), 318
n_scamp_working_cores (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
n_shared_message_buffers (spinnman.messages.spinnaker_boot.system_variable_boot_values.Syst attribute), 318
n_shared_message_buffers (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
n_timeouts (spinnman.connections.scp_request_pipeline.SCPRequestPipeL attribute), 244
n_timeouts (spinnman.connections.SCPRequestPipeLine attribute), 246
n_timestamps (spinn_front_end_common.interface.buffer_management.stor attribute), 513
n_timestamps (spinn_front_end_common.interface.buffer_management.stor attribute), 521
n_user_processors (spinn_machine.Chip attribute), 42
n_user_processors (spinn_machine.chip.Chip attribute), 23
n_vertices (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 140
n_vertices (pacman.model.graphs.AbstractGraph attribute), 145
n_vertices (pacman.model.graphs.simple_machine_vertex.ReverseIPTagMul attribute), 128
n_vertices (pacman.model.graphs.SimpleMachineVertex attribute), 125
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 745
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 748
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 746
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 746
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 747
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 747
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 748
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 749
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 750
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep attribute), 750
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dep

attribute), 749
n_weight_terms (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.SpinnakerPairwiseSTDPTimeDependence attribute), 751
n_working_cores (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
n_working_cores (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
name (pacman.executor.algorithm_decorators.abstract_input.AbstractInput attribute), 87
name (pacman.executor.algorithm_decorators.AbstractInput attribute), 91
name (pacman.executor.algorithm_decorators.all_of_input.AllOfInput attribute), 89
name (pacman.executor.algorithm_decorators.AllOfInput attribute), 91
name (pacman.executor.algorithm_decorators.one_of_input.OneOfInput attribute), 90
name (pacman.executor.algorithm_decorators.OneOfInput attribute), 92
name (pacman.executor.algorithm_decorators.single_input.SingleInput attribute), 90
name (pacman.executor.algorithm_decorators.SingleInput attribute), 92
name (pacman.utilities.utility_objs.Field attribute), 203
name (pacman.utilities.utility_objs.field.Field attribute), 199
name (pacman.utilities.utility_objs.flexi_field.FlexiField attribute), 199
name (pacman.utilities.utility_objs.FlexiField attribute), 203
name (spinnman.model.version_info.VersionInfo attribute), 346
name (spinnman.model.VersionInfo attribute), 360
names (spinn_front_end_common.utilities.utility_objs.provenance_data_item.ProvenanceDataItem attribute), 574
names (spinn_front_end_common.utilities.utility_objs.ProvenanceDataItem attribute), 577
NATIVE_128_X_128 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameter_update_protocol.PushBotLuminanceSetup attribute), 639
NATIVE_128_X_128 (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameter_update_protocol.PushBotRetinaResolution attribute), 641
NATIVE_128_X_128 (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey attribute), 822
NATIVE_128_X_128 (spynnaker.pyNN.protocols.RetinaKey attribute), 827
nearest_ethernet_x (spinn_machine.Chip attribute), 42
nearest_ethernet_x (spinn_machine.chip.Chip attribute), 23
nearest_ethernet_x (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
nearest_ethernet_x (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
nearest_ethernet_x (spinnman.model.chip_info.ChipInfo attribute), 333
nearest_ethernet_x (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
nearest_ethernet_x (spinnman.model.ChipInfo attribute), 349
nearest_ethernet_x (spinnman.model.ChipSummaryInfo attribute), 350
nearest_ethernet_y (spinn_machine.Chip attribute), 42
nearest_ethernet_y (spinn_machine.chip.Chip attribute), 24
nearest_ethernet_y (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
nearest_ethernet_y (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
nearest_ethernet_y (spinnman.model.chip_info.ChipInfo attribute), 333
nearest_ethernet_y (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
nearest_ethernet_y (spinnman.model.ChipInfo attribute), 349
nearest_ethernet_y (spinnman.model.ChipSummaryInfo attribute), 350
NEAREST_NEIGHBOUR (spinn_front_end_common.utilities.utility_objs.dpri_flags.DPRIFlags attribute), 572
NEAREST_NEIGHBOUR (spinnman.messages.scp.enums.signal.SignalType attribute), 277
NEAREST_NEIGHBOUR (spinnman.messages.spinnaker_boot.diagnostic_filter.PacketTypeDiagnosticFilter attribute), 326
NEAREST_NEIGHBOUR (spinnman.messages.spinnaker_boot.diagnostic_filter.PacketType attribute), 330
nearest_neighbour_delay_us (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
nearest_neighbour_delay_us (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
nearest_neighbour_forward (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318

nearest_neighbour_forward (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
nearest_neighbour_last_id (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
nearest_neighbour_last_id (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
nearest_neighbour_memory_pointer (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 318
nearest_neighbour_memory_pointer (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
nearest_neighbour_retry (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
nearest_neighbour_retry (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
needs_buffering() (in module spynnaker.pyNN.models.common), 676
needs_buffering() (in module spynnaker.pyNN.models.common.recording_utils), 672
needs_database (spinn_front_end_common.interface.interface_functions.load_database_interface.LoadDatabaseInterface attribute), 531
needs_database (spinn_front_end_common.interface.interface_functions.notification_interface.NotificationInterface attribute), 538
needs_to_write_region() (data_specification.memory_region_collection.MemoryRegionCollection method), 456
needs_to_write_region() (data_specification.MemoryRegionCollection method), 489
NeedsNMachineTimeSteps (class in spinn_front_end_common.abstract_models.impl), 495
NeedsNMachineTimeSteps (class in spinn_front_end_common.abstract_models.impl.needs_n_machine_time_steps), 494
NestedFunctionException, 455
netinit_bc_wait_time (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
netinit_bc_wait_time (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
netinit_phase (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
netinit_phase (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
NetworkSpecification (class in pacman.operations.algorithm_reports.network_specification), 180
NEURON_PARAMS (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGION attribute), 828
NeuronModelIzh (class in spynnaker.pyNN.models.neuron.neuron_models), 739
NeuronModelIzh (class in spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh), 735
NeuronModelLeakyIntegrate (class in spynnaker.pyNN.models.neuron.neuron_models), 741
NeuronModelLeakyIntegrate (class in spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_ 736
NeuronModelLeakyIntegrateAndFire (class in spynnaker.pyNN.models.neuron.neuron_models), 742
NeuronModelLeakyIntegrateAndFire (class in spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_ 737
NeuronParameter (class in spynnaker.pyNN.models.neural_properties), 701
NeuronParameter (class in spynnaker.pyNN.models.neural_properties.neural_parameter), 701
NeuronRecorderDatabaseInterface (class in spynnaker.pyNN.models.common), 675
NeuronRecorderDatabaseInterface (class in spynnaker.pyNN.models.common.neuron_recorder), 679
NEW (spinnman.messages.scp.enums.iptag_command.IPTagCommand attribute), 274
NEW (spinnman.messages.scp.enums.IPTagCommand attribute), 279
next() (pacman.operations.routing_info_allocator_algorithms.malloc_based_ method), 193
next_address (spinnman.model.heap_element.HeapElement attribute), 341
next_address (spinnman.model.HeapElement attribute), 356
next_element (spinnman.messages.eieio.data_messages.eieio_data_message attribute), 261
next_element (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 265
next_key (pacman.operations.routing_info_allocator_algorithms.malloc_bas attribute), 193
next_key (spinn_front_end_common.interface.buffer_management.storage_ attribute), 513
next_key (spinn_front_end_common.interface.buffer_management.storage_ attribute), 521
next_timestamp (spinn_front_end_common.interface.buffer_management.st attribute), 514
next_timestamp (spinn_front_end_common.interface.buffer_management.st

attribute), 521
NO_APPLICATION (spinn_front_end_common.utilities.utility_objs.execute_flow_type.ExecutableType attribute), 572
NO_APPLICATION (spinn_front_end_common.utilities.utility_objs.ExecutableType attribute), 576
no_blocks_freed (spinnman.processes.de_alloc_sdram_process.DeAllocSDRAMProcess attribute), 362
no_blocks_freed (spinnman.processes.DeAllocSDRAMProcess attribute), 368
no_machine_time_steps (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
no_machine_time_steps (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
no_machine_time_steps (spinn_front_end_common.utilities.FailedState attribute), 583
no_machine_time_steps (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 583
no_machine_time_steps (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
no_operation() (data_specification.data_specification_generator.DataSpecificationGenerator method), 438
no_operation() (data_specification.DataSpecificationGenerator method), 476
NO_PAYLOAD (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.ExternalPayload attribute), 822
NO_PAYLOAD (spynnaker.pyNN.protocols.RetinaPayload attribute), 827
NoConfigFoundException, 5
NON_DEFAULT_ROUTED (spinnman.model.enums.diagnostic_filter_default_routing_status.DiagnosticFilterDefaultRoutingStatus attribute), 325
NON_DEFAULT_ROUTED (spinnman.model.enums.DiagnosticFilterDefaultRoutingStatus attribute), 329
NON_LOCAL (spinnman.model.enums.diagnostic_filter_source.DiagnosticFilterSource attribute), 327
NON_LOCAL (spinnman.model.enums.DiagnosticFilterSource attribute), 331
NONE (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
NONE (spinnman.model.enums.P2PTableRoute attribute), 331
NONE (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
NONE (spinnman.model.enums.RunTimeError attribute), 330
none_labelled_edge_count (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
none_labelled_vertex_count (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
none_labelled_vertex_count (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
none_labelled_vertex_count (spinn_front_end_common.utilities.FailedState attribute), 583
none_labelled_vertex_count (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 583
none_labelled_vertex_count (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
none_pynn_default_parameters (spynnaker.pyNN.external_devices_models.external_device_lif_control attribute), 653
none_pynn_default_parameters (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl attribute), 661
none_pynn_default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_control attribute), 622
none_pynn_default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_control attribute), 623
none_pynn_default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_control attribute), 624
none_pynn_default_parameters (spynnaker.pyNN.external_devices_models.push_bot.push_bot_control attribute), 625
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP attribute), 782
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 790
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.if_cond_exp_base.IFCondExp attribute), 708
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc.IFCondExpS attribute), 709
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_delta.IFCurrDelta attribute), 711
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base.IFCurr attribute), 712
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.IFCurrExpBa attribute), 713
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive.IFCu attribute), 714

none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.IFCondExpBase attribute), 719
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.IFCondExpStoc attribute), 722
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrDelta attribute), 723
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrDualExpBase attribute), 720
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrExpBase attribute), 721
none_pynn_default_parameters (spynnaker.pyNN.models.neuron.builds.IFCurrExpCa2Adaptive attribute), 723
none_pynn_default_parameters (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray attribute), 794
none_pynn_default_parameters (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson attribute), 798
none_pynn_default_parameters (spynnaker.pyNN.models.spike_source.SpikeSourceArray attribute), 801
none_pynn_default_parameters (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 804
NOP (data_specification.enums.Commands attribute), 418
NOP (data_specification.enums.commands.Commands attribute), 415
NORMAL (spinnman.model.enums.diagnostic_filter_emergency_routing_status.DiagnosticFilterEmergencyRoutingStatus attribute), 326
NORMAL (spinnman.model.enums.DiagnosticFilterEmergencyRoutingStatus attribute), 330
NORTH (pacman.utilities.constants.EDGES attribute), 207
NORTH (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
NORTH (spinnman.model.enums.P2PTableRoute attribute), 331
NORTH_EAST (pacman.utilities.constants.EDGES attribute), 207
NORTH_EAST (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
NORTH_EAST (spinnman.model.enums.P2PTableRoute attribute), 331
NOT (data_specification.enums.logic_operation.LogicOperation attribute), 417
NOT (data_specification.enums.LogicOperation attribute), 420
NOT_EQUAL (data_specification.enums.Condition attribute), 419
NOT_EQUAL (data_specification.enums.condition.Condition attribute), 416
NotificationProtocol (class in spinn_front_end_common.interface.interface_functions), 539
NotificationProtocol (class in spinn_front_end_common.interface.interface_functions.notificati 534
NotificationProtocol (class in spinn_front_end_common.utilities.notification_protocol), 563
NotificationProtocol (class in spinn_front_end_common.utilities.notification_protocol.notificati 562
NotificationProtocolPauseStop (class in spinnman.messages.eieio.command_messages), 255
NotificationProtocolPauseStop (class in spinnman.messages.eieio.command_messages.notification_protocol_pa 255
NotificationProtocolStartResume (class in spinnman.messages.eieio.command_messages), 255
NotificationProtocolStartResume (class in spinnman.messages.eieio.command_messages.notification_protocol_st 251
notify_host_name (spinn_front_end_common.utilities.notification_protocol attribute), 563
notify_host_name (spinn_front_end_common.utilities.notification_protocol attribute), 564
notify_host_name (spinn_utilities.socket_address.SocketAddress attribute), 13
notify_port_no (spinn_front_end_common.utilities.notification_protocol.soc attribute), 563
notify_port_no (spinn_front_end_common.utilities.notification_protocol.So attribute), 564
notify_port_no (spinn_utilities.socket_address.SocketAddress attribute), 13
NULL (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
NULL (spinnman.model.enums.RunTimeError attribute), 330
number_of_boards_located (spinnman.transceiver.Transceiver attribute), 398
number_of_defaultable_entries (pacman.model.routing_tables.multicast_routing_table.MulticastRout attribute), 174
number_of_defaultable_entries (pacman.model.routing_tables.MulticastRoutingTable attribute), 176
number_of_entries (pac-

man.model.routing_tables.multicast_routing_table.MulticastRoutingTable attribute), 175
number_of_entries (pacman.model.routing_tables.MulticastRoutingTable attribute), 176
number_of_packets_sent_per_time_step (spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.LivePacketGatherParameters attribute), 573
number_of_packets_sent_per_time_step (spinn_front_end_common.utilities.utility_objs.LivePacketGatherParameters attribute), 576
NUMPY_CONNECTORS_DTYPE (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics attribute), 756
NUMPY_CONNECTORS_DTYPE (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics attribute), 760
NUMPY_SYNAPSES_DTYPE (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector attribute), 677
NUMPY_SYNAPSES_DTYPE (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector attribute), 687

O

OFF (spinnman.messages.scp.enums.led_action.LEDAction attribute), 275
OFF (spinnman.messages.scp.enums.LEDAction attribute), 279
offset (spinnman.messages.eieio.command_messages.eieio_command_message.EIEIOCommandMessage attribute), 249
offset (spinnman.messages.eieio.command_messages.EIEIOCommandMessage attribute), 254
offset (spinnman.messages.sdp.sdp_message.SDPMessage attribute), 312
offset (spinnman.messages.sdp.SDPMessage attribute), 315
offset (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
offset (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
ON (spinnman.messages.scp.enums.led_action.LEDAction attribute), 275
ON (spinnman.messages.scp.enums.LEDAction attribute), 279
OneOf (class in pacman.executor.algorithm_decorators), 93
OneOf (class in pacman.executor.algorithm_decorators.algorithm_decorator), 87
OneOfInput (class in pacman.executor.algorithm_decorators), 91
OneOfInput (class in pacman.executor.algorithm_decorators.one_of_input), 89
OneToOneConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 694
OneToOneConnector (class in spynnaker.pyNN.models.neural_projections.connectors.one_to_one_c
OneToOnePlacer (class in pacman.operations.placer_algorithms.one_to_one_placer), 185
ongoing_function_definition (data_specification.data_specification_generator.DataSpecificatio attribute), 438
ongoing_function_definition (data_specification.DataSpecificationGenerator attribute), 476
ongoing_loop (data_specification.data_specification_generator.DataSpecific attribute), 438
ongoing_loop (data_specification.DataSpecificationGenerator attribute), 476
opcode (data_specification.data_specification_executor_functions.DataSpec attribute), 424
opcode (data_specification.DataSpecificationExecutorFunctions attribute), 462
opcode (spinnman.messages.spinnaker_boot.spinnaker_boot_message.Spin attribute), 316
opcode (spinnman.messages.spinnaker_boot.SpinnakerBootMessage attribute), 320
operand_1 (spinnman.messages.spinnaker_boot.spinnaker_boot_message.S attribute), 316
operand_1 (spinnman.messages.spinnaker_boot.SpinnakerBootMessage attribute), 320
operand_2 (spinnman.messages.spinnaker_boot.spinnaker_boot_message.S attribute), 316
operand_2 (spinnman.messages.spinnaker_boot.SpinnakerBootMessage attribute), 321
operand_3 (spinnman.messages.spinnaker_boot.spinnaker_boot_message.S attribute), 316
operand_3 (spinnman.messages.spinnaker_boot.SpinnakerBootMessage attribute), 321
operation (spinnman.exceptions.SpinnmanTimeoutException attribute), 382
operation (spinnman.exceptions.SpinnmanUnexpectedResponseCodeExcep attribute), 383
operation (spinnman.exceptions.SpinnmanUnsupportedOperationException attribute), 383
optional_inputs (pacman.executor.algorithm_classes.abstract_algorithm.Abs attribute), 80
optional_inputs (pacman.executor.algorithm_classes.AbstractAlgorithm attribute), 84
optionxform() (spinn_utilities.configs.camel_case_config_parser.CamelCas method), 4
optionxform() (spinn_utilities.configs.case_sensitive_parser.CaseSensitiveP method), 4

OR (data_specification.enums.logic_operation.LogicOperation attribute), 417
OR (data_specification.enums.LogicOperation attribute), 420
OrderedSet (class in spinn_machine.utilities.ordered_set), 21
OrderedSet (class in spinn_utilities.ordered_set), 10
out_going_links (pacman.model.routing_table_by_partition.multicast_routing_table_by_partition_entry.MulticastRoutingTableByPartit attribute), 172
out_going_links (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartitionEntry attribute), 173
out_going_processors (pacman.model.routing_table_by_partition.multicast_routing_table_by_partition_entry.MulticastRoutingTableByPartitionEntry attribute), 172
out_going_processors (pacman.model.routing_table_by_partition.MulticastRoutingTableByPartitionEntry attribute), 173
outdated_config() (in module spinn_utilities.conf_loader), 8
outgoing_edge_partitions (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 140
outgoing_edge_partitions (pacman.model.graphs.AbstractGraph attribute), 145
outgoing_edge_partitions (pacman.model.graphs.impl.Graph attribute), 128
outgoing_edge_partitions (pacman.model.graphs.impl.graph.Graph attribute), 125
OutgoingEdgePartition (class in pacman.model.graphs.impl), 128
OutgoingEdgePartition (class in pacman.model.graphs.impl.outgoing_edge_partition), 125
Output (class in pacman.executor.algorithm_decorators), 92
Output (class in pacman.executor.algorithm_decorators.output), 90
OUTPUT_BUFFERING_SDP_PORT (spinn_front_end_common.utilities.constants.SDP_PORTS attribute), 578
output_type (pacman.executor.algorithm_decorators.Output attribute), 92
output_type (pacman.executor.algorithm_decorators.output.Output attribute), 90
outputs (pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgorithm attribute), 80
outputs (pacman.executor.algorithm_classes.AbstractAlgorithm attribute), 84
over() (spinn_machine.utilities.progress_bar.ProgressBar method), 22
over() (spinn_utilities.progress_bar.ProgressBar method), 12
OVER_RUN_THRESHOLD_BEFORE_ERROR (spinn_front_end_common.mapping_algorithms.on_chip_router_ attribute), 550
overrides (class in pacman.model.decorators), 108
overrides (class in pacman.model.decorators.overrides), 108
overrides (class in spinn_utilities.overrides), 11

P

p (pacman.model.constraints.placer_constraints.chip_and_core_constraint.C attribute), 104
p (pacman.model.constraints.placer_constraints.ChipAndCoreConstraint attribute), 106
p (pacman.model.placements.Placement attribute), 150
p (pacman.model.placements.placement.Placement attribute), 148
p (pacman.operations.multi_cast_router_check_functionality.valid_routes_c attribute), 185
p (spinnman.messages.eieio.command_messages.spinnaker_request_buffers attribute), 252
p (spinnman.messages.eieio.command_messages.spinnaker_request_read_d attribute), 253
p (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffers attribute), 256
p (spinnman.messages.eieio.command_messages.SpinnakerRequestReadDa attribute), 257
p (spinnman.model.cpu_info.CPUInfo attribute), 337
p (spinnman.model.CPUInfo attribute), 352
p (spinnman.model.io_buffer.IOBuffer attribute), 342
p (spinnman.model.IOBuffer attribute), 356
p (spinnman.model.version_info.VersionInfo attribute), 346
p (spinnman.model.VersionInfo attribute), 361
p2p_b_repeats (spinnman.messages.spinnaker_boot.system_variable_boot_ attribute), 319
p2p_b_repeats (spinnman.messages.spinnaker_boot.SystemVariableDefiniti attribute), 323
p2p_root_x (spinnman.messages.spinnaker_boot.system_variable_boot_val attribute), 319
p2p_root_x (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
p2p_root_y (spinnman.messages.spinnaker_boot.system_variable_boot_val attribute), 319
p2p_root_y (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
P2PTable (class in spinnman.model), 357
P2PTable (class in spinnman.model.p2p_table), 342
P2PTableRoute (class in spinnman.model.enums), 331
P2PTableRoute (class in spinnman.model.enums.p2p_table_route), 327
PABT (spinnman.model.enums.run_time_error.RunTimeError attribute), 328

PABT (spinnman.model.enums.RunTimeError attribute), 330
packet (spinnman.exceptions.SpinnmanEIEIOPacketParsingException attribute), 381
packet_type (spinnman.exceptions.SpinnmanInvalidPacketException attribute), 381
packet_types (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
packet_types (spinnman.model.DiagnosticFilter attribute), 355
pacman (module), 77, 213
pacman.exceptions (module), 211
pacman.executor (module), 97
pacman.executor.algorithm_classes (module), 84
pacman.executor.algorithm_classes.abstract_algorithm (module), 80
pacman.executor.algorithm_classes.abstract_python_algorithm (module), 81
pacman.executor.algorithm_classes.external_algorithm (module), 82
pacman.executor.algorithm_classes.python_class_algorithm (module), 82
pacman.executor.algorithm_classes.python_function_algorithm (module), 83
pacman.executor.algorithm_decorators (module), 91
pacman.executor.algorithm_decorators.abstract_input (module), 87
pacman.executor.algorithm_decorators.algorithm_decorator (module), 87
pacman.executor.algorithm_decorators.all_of_input (module), 89
pacman.executor.algorithm_decorators.one_of_input (module), 89
pacman.executor.algorithm_decorators.output (module), 90
pacman.executor.algorithm_decorators.single_input (module), 90
pacman.executor.algorithm_metadata_xml_reader (module), 94
pacman.executor.injection_decorator (module), 94
pacman.executor.pacman_algorithm_executor (module), 95
pacman.model (module), 180
pacman.model.abstract_classes (module), 98
pacman.model.abstract_classes.abstract_has_global_max_atoms (module), 98
pacman.model.constraints (module), 107
pacman.model.constraints.abstract_constraint (module), 107
pacman.model.constraints.key_allocator_constraints (module), 100
pacman.model.constraints.key_allocator_constraints.abstract_key_allocator_constraint (module), 99
pacman.model.constraints.key_allocator_constraints.contiguous_key_range_constraint (module), 99
pacman.model.constraints.key_allocator_constraints.fixed_key_and_mask_constraint (module), 99
pacman.model.constraints.key_allocator_constraints.fixed_key_field_constraint (module), 100
pacman.model.constraints.key_allocator_constraints.fixed_mask_constraint (module), 100
pacman.model.constraints.key_allocator_constraints.flexi_key_field_constraint (module), 100
pacman.model.constraints.partitioner_constraints (module), 103
pacman.model.constraints.partitioner_constraints.abstract_partitioner_constraint (module), 102
pacman.model.constraints.partitioner_constraints.max_vertex_atoms_constraint (module), 102
pacman.model.constraints.partitioner_constraints.same_atoms_as_vertex_constraint (module), 102
pacman.model.constraints.placer_constraints (module), 105
pacman.model.constraints.placer_constraints.abstract_placer_constraint (module), 103
pacman.model.constraints.placer_constraints.board_constraint (module), 104
pacman.model.constraints.placer_constraints.chip_and_core_constraint (module), 104
pacman.model.constraints.placer_constraints.radial_placement_from_chip_constraint (module), 105
pacman.model.constraints.placer_constraints.same_chip_as_constraint (module), 105
pacman.model.constraints.router_constraints (module), 107
pacman.model.constraints.router_constraints.abstract_router_constraint (module), 107
pacman.model.decorators (module), 108
pacman.model.decorators.overrides (module), 108
pacman.model.graphs (module), 142
pacman.model.graphs.abstract_edge (module), 137
pacman.model.graphs.abstract_fpga_vertex (module), 137
pacman.model.graphs.abstract_graph (module), 137
pacman.model.graphs.abstract_outgoing_edge_partition (module), 140
pacman.model.graphs.abstract_spinnaker_link_vertex (module), 141
pacman.model.graphs.abstract_vertex (module), 141
pacman.model.graphs.abstract_virtual_vertex (module), 142
pacman.model.graphs.application (module), 114
pacman.model.graphs.application.application_edge (module), 109
pacman.model.graphs.application.application_fpga_vertex (module), 110
pacman.model.graphs.application.application_graph (module), 111

pacman.model.graphs.application.application_outgoing_edge_partition (module), 111
pacman.model.graphs.application.application_spinnaker_link_vertex (module), 112
pacman.model.graphs.application.application_vertex (module), 113
pacman.model.graphs.common (module), 121
pacman.model.graphs.common.constrained_object (module), 118
pacman.model.graphs.common.edge_traffic_type (module), 119
pacman.model.graphs.common.graph_mapper (module), 119
pacman.model.graphs.common.slice (module), 120
pacman.model.graphs.impl (module), 126
pacman.model.graphs.impl.graph (module), 123
pacman.model.graphs.impl.outgoing_edge_partition (module), 125
pacman.model.graphs.machine (module), 133
pacman.model.graphs.machine.machine_edge (module), 130
pacman.model.graphs.machine.machine_fpga_vertex (module), 131
pacman.model.graphs.machine.machine_graph (module), 131
pacman.model.graphs.machine.machine_outgoing_edge_partition (module), 132
pacman.model.graphs.machine.machine_spinnaker_link_vertex (module), 132
pacman.model.graphs.machine.machine_vertex (module), 133
pacman.model.graphs.machine.simple_machine_vertex (module), 133
pacman.model.placements (module), 150
pacman.model.placements.placement (module), 147
pacman.model.placements.placements (module), 148
pacman.model.resources (module), 159
pacman.model.resources.abstract_resource (module), 152
pacman.model.resources.core_resource (module), 152
pacman.model.resources.cpu_cycles_per_tick_resource (module), 153
pacman.model.resources.dtcm_resource (module), 153
pacman.model.resources.element_free_space (module), 153
pacman.model.resources.iptag_resource (module), 153
pacman.model.resources.pre_allocated_resource_container (module), 154
pacman.model.resources.resource_container (module), 155
pacman.model.resources.reverse_iptag_resource (module), 156
pacman.model.resources.sdram_resource (module), 156
pacman.model.resources.specific_board_iptag_resource (module), 157
pacman.model.resources.specific_board_reverse_iptag_resource (module), 158
pacman.model.resources.specific_chip_sdram_resource (module), 159
pacman.model.resources.specific_core_resource (module), 159
pacman.model.routing_info (module), 167
pacman.model.routing_info.abstract_machine_partition_n_keys_map (module), 164
pacman.model.routing_info.base_key_and_mask (module), 164
pacman.model.routing_info.dict_based_machine_partition_n_keys_map (module), 165
pacman.model.routing_info.partition_routing_info (module), 166
pacman.model.routing_info.routing_info (module), 166
pacman.model.routing_table_by_partition (module), 172
pacman.model.routing_table_by_partition.multicast_routing_table_by_parti (module), 170
pacman.model.routing_table_by_partition.multicast_routing_table_by_parti (module), 171
pacman.model.routing_tables (module), 176
pacman.model.routing_tables.multicast_routing_table (module), 174
pacman.model.routing_tables.multicast_routing_tables (module), 175
pacman.model.tags (module), 179
pacman.model.tags.tags (module), 178
pacman.operations (module), 195
pacman.operations.algorithm_reports (module), 184
pacman.operations.algorithm_reports.network_specification (module), 180
pacman.operations.algorithm_reports.reports (module), 180
pacman.operations.chip_id_allocator_algorithms (module), 185
pacman.operations.chip_id_allocator_algorithms.malloc_based_chip_id_all (module), 184
pacman.operations.fixed_route_router (module), 185
pacman.operations.fixed_route_router.fixed_route_router (module), 185
pacman.operations.multi_cast_router_check_functionality (module), 186
pacman.operations.multi_cast_router_check_functionality.valid_routes_che (module), 185
pacman.operations.partition_algorithms (module), 187
pacman.operations.partition_algorithms.basic_partitioner (module), 186
pacman.operations.partition_algorithms.partition_and_place_partitioner (module), 186
pacman.operations.placer_algorithms (module), 188
pacman.operations.placer_algorithms.basic_placer (module), 187
pacman.operations.placer_algorithms.connective_based_placer

(module), 187
pacman.operations.placer_algorithms.one_to_one_placer (module), 187
pacman.operations.placer_algorithms.radial_placer (module), 187
pacman.operations.rig_algorithms (module), 189
pacman.operations.rig_algorithms.rig_mc_route (module), 188
pacman.operations.rig_algorithms.rig_place (module), 188
pacman.operations.rig_algorithms.rig_place_and_route (module), 188
pacman.operations.rig_algorithms.rig_route (module), 188
pacman.operations.rigged_algorithms (module), 190
pacman.operations.rigged_algorithms.hilbert_placer (module), 189
pacman.operations.rigged_algorithms.hilbert_state (module), 189
pacman.operations.rigged_algorithms.isomorph_check (module), 189
pacman.operations.rigged_algorithms.random_placer (module), 190
pacman.operations.router_algorithms (module), 190
pacman.operations.router_algorithms.basic_dijkstra_routing (module), 190
pacman.operations.router_compressors (module), 192
pacman.operations.router_compressors.basic_route_merger (module), 191
pacman.operations.router_compressors.malloc_based_route_merger (module), 192
pacman.operations.router_compressors.mundys_router_compressor (module), 191
pacman.operations.router_compressors.mundys_router_compressor_utilities (module), 191
pacman.operations.routing_info_allocator_algorithms (module), 194
pacman.operations.routing_info_allocator_algorithms.basic_routing_info_allocator (module), 193
pacman.operations.routing_info_allocator_algorithms.destination_based_key_allocator (module), 193
pacman.operations.routing_info_allocator_algorithms.field_based_routing_info_allocator (module), 192
pacman.operations.routing_info_allocator_algorithms.field_based_routing_info_allocator_objs (module), 192
pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator (module), 193
pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_info_allocator (module), 192
pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.key_field_generator (module), 193
pacman.operations.routing_info_allocator_algorithms.malloc_based_routing_allocator.malloc_based_routing_info_allocator (module), 193
pacman.operations.routing_table_generators (module), 194
pacman.operations.routing_table_generators.basic_routing_table_generator (module), 194
pacman.operations.tag_allocator_algorithms (module), 194
pacman.operations.tag_allocator_algorithms.basic_tag_allocator (module), 194
pacman.utilities (module), 210
pacman.utilities.algorithm_utilities (module), 197
pacman.utilities.algorithm_utilities.element_allocator_algorithm (module), 195
pacman.utilities.algorithm_utilities.field_based_system_utilities (module), 195
pacman.utilities.algorithm_utilities.machine_algorithm_utilities (module), 196
pacman.utilities.algorithm_utilities.partition_algorithm_utilities (module), 196
pacman.utilities.algorithm_utilities.placer_algorithm_utilities (module), 196
pacman.utilities.algorithm_utilities.routing_info_allocator_utilities (module), 196
pacman.utilities.constants (module), 207
pacman.utilities.file_format_converters (module), 198
pacman.utilities.file_format_converters.convert_to_file_core_allocations (module), 197
pacman.utilities.file_format_converters.convert_to_file_machine (module), 197
pacman.utilities.file_format_converters.convert_to_file_machine_graph (module), 197
pacman.utilities.file_format_converters.convert_to_file_machine_graph_pur (module), 198
pacman.utilities.file_format_converters.convert_to_file_placement (module), 198
pacman.utilities.file_format_converters.convert_to_memory_multi_cast_rou (module), 198
pacman.utilities.file_format_converters.convert_to_memory_placements (module), 198
pacman.utilities.file_format_converters.create_file_constraints (module), 198
pacman.utilities.file_format_schemas (module), 198
pacman.utilities.rig_converters (module), 208
pacman.utilities.utility_calls (module), 208
pacman.utilities.utility_objs (module), 203
pacman.utilities.utility_objs.field (module), 199
pacman.utilities.utility_objs.flexi_field (module), 199
pacman.utilities.utility_objs.resource_tracker (module), 199
pacman.utilities.utility_objs.vertex_sorter (module), 210
PACMANAlgorithmExecutor (class in pacman.executor), 97
PACMANAlgorithmExecutor (class in pacman.operations.pacman_algorithm_executor), 95
PacmanAlgorithmFailedToCompleteException, 211

PacmanAlgorithmFailedToGenerateOutputsException, 211
PacmanAlreadyExistsException, 211
PacmanAlreadyPlacedError, 212
PacmanConfigurationException, 212
PacmanElementAllocationException, 212
PacmanException, 212
PacmanExternalAlgorithmFailedToCompleteException, 212
PacmanInvalidParameterException, 212
PacmanNoMergeException, 212
PacmanNotExistException, 212
PacmanNotFoundError, 212
PacmanNotPlacedError, 212
PacmanPartitionException, 213
PacmanPlaceException, 213
PacmanProcessorAlreadyOccupiedError, 213
PacmanProcessorNotOccupiedError, 213
PacmanProvenanceExtractor (class in spinn_front_end_common.interface.provenance), 545
PacmanProvenanceExtractor (class in spinn_front_end_common.interface.provenance.provenance_extractor), 544
PacmanPruneException, 213
PacmanRouteInfoAllocationException, 213
PacmanRoutingException, 213
PacmanTypeError, 213
PacmanValueError, 213
padding_1 (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
padding_1 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
padding_2 (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
padding_2 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
padding_3 (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
padding_3 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
padding_4 (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
padding_4 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
PaddingRequest (class in spinnman.messages.eieio.command_messages), 255
PaddingRequest (class in spinnman.messages.eieio.command_messages.padding_request), 251
param_types (pacman.executor.algorithm_decorators.abstract_input.AbstractInput attribute), 87
param_types (pacman.executor.algorithm_decorators.AbstractInput attribute), 91
param_types (pacman.executor.algorithm_decorators.all_of_input.AllOfInp attribute), 89
param_types (pacman.executor.algorithm_decorators.AllOfInput attribute), 91
param_types (pacman.executor.algorithm_decorators.one_of_input.OneOfI attribute), 90
param_types (pacman.executor.algorithm_decorators.OneOfInput attribute), 92
param_types (pacman.executor.algorithm_decorators.single_input.SingleInp attribute), 90
param_types (pacman.executor.algorithm_decorators.SingleInput attribute), 92
parameter (spinn_machine.exceptions.SpinnMachineInvalidParameterExcep attribute), 27
parameter (spinnman.exceptions.SpinnmanInvalidParameterException attribute), 382
parameter (spinnman.exceptions.SpinnmanInvalidParameterTypeException attribute), 382
PARAMS_REGION (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_m attribute), 655
PARAMS_REGION (spynnaker.pyNN.external_devices_models.MunichMotorDevice attribute), 663
PARAMS_SIZE (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_m attribute), 655
PARAMS_SIZE (spynnaker.pyNN.external_devices_models.MunichMotorDevice attribute), 663
partition (pacman.model.routing_info.partition_routing_info.PartitionRouti attribute), 166
partition (pacman.model.routing_info.PartitionRoutingInfo attribute), 169
partition (spinn_front_end_common.utilities.utility_objs.live_packet_gat attribute), 573
partition (spinn_front_end_common.utilities.utility_objs.LivePacketGath attribute), 576
PartitionAndPlacePartitioner (class in pacman.operations.partition_algorithms), 187
PartitionAndPlacePartitionerDefinition (in pacman.operations.partition_algorithms.partition_and_place_partitio 186
partitioner_report() (in module pacman.operations.algorithm_reports.reports), 181
PartitionRoutingInfo (class in pacman.model.routing_info), 169
PartitionRoutingInfo (class in pacman.model.routing_info.partition_routing_info), 166
PAUSE (spinnman.messages.scp.enums.Signal attribute), 280

PAUSE (spinnman.messages.scp.enums.signal.Signal attribute), 277
pause_stop_commands (spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands.SendMeMulticastCommandsVertex attribute), 498
pause_stop_commands (spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands.SendMeMulticastCommandsVertex attribute), 501
pause_stop_commands (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARetinaDevice attribute), 654
pause_stop_commands (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
pause_stop_commands (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
pause_stop_commands (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 665
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice attribute), 649
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotRetinaDevice attribute), 649
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_laser_device.PushBotEthernetLaserDevice attribute), 627
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice attribute), 627
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice attribute), 628
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_speaker_device.PushBotEthernetSpeakerDevice attribute), 630
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_model.BaghdsticFilterDiagnosticFilter attribute), 634
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_LEDDevice attribute), 634
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_PushBotEthernetMotorDevice attribute), 635
pause_stop_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_PushBotEthernetSpeakerDevice attribute), 636
PAUSED (spinnman.model.enums.cpu_state.CPUState attribute), 325
PAUSED (spinnman.model.enums.CPUState attribute), 329
payload (spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand attribute), 598
payload (spinn_front_end_common.utility_models.MultiCastCommand attribute), 609
payload (spinnman.messages.multicast_messages.key_payload_data_element.KeyPayloadDataElement attribute), 261
payload (spinnman.messages.multicast_messages.KeyPayloadDataElement attribute), 265
payload (spinnman.messages.multicast_message.MulticastMessage attribute), 324
payload_as_time_stamps (spinn_front_end_common.utilities.utility_objs.live_packet_gatherDevice attribute), 573
payload_as_time_stamps (spinn_front_end_common.utilities.utility_objs.LivePacketGatherDevice attribute), 576
payload_base (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader attribute), 259
payload_base (spinnman.messages.eieio.data_messages.EIEIODataHeader attribute), 263
payload_bytes (spinnman.messages.eieio.eieio_type.EIEIOType attribute), 267
payload_bytes (spinnman.messages.eieio.EIEIOType attribute), 267
payload_is_timestamp (spinnman.messages.eieio.data_messages.key_payload_data_element.KeyPayloadDataElement attribute), 261
payload_is_timestamp (spinnman.messages.eieio.data_messages.KeyPayloadDataElement attribute), 265
payload_prefix (spinn_front_end_common.utilities.utility_objs.LivePacketGatherDevice attribute), 573
payload_prefix (spinn_front_end_common.utilities.utility_objs.LivePacketGatherDevice attribute), 576
payload_right_shift (spinn_front_end_common.utilities.utility_objs.live_packet_gatherDevice attribute), 573
payload_right_shift (spinn_front_end_common.utilities.utility_objs.LivePacketGatherDevice attribute), 577
payload_statuses (spinnman.model.BaghdsticFilterDiagnosticFilter attribute), 340
payload_statuses (spinnman.model.DiagnosticFilter attribute), 355
peer_to_peer_hop_table_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
peer_to_peer_hop_table_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
physical_cpu_id (spinnman.model.cpu_info.CPUInfo attribute), 337
physical_cpu_id (spinnman.model.CPUInfo attribute), 352
physical_to_virtual_core_map (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319

physical_to_virtual_core_map (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
physical_to_virtual_core_map (spinnman.model.chip_info.ChipInfo attribute), 333
physical_to_virtual_core_map (spinnman.model.ChipInfo attribute), 349
pixels (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey attribute), 822
pixels (spynnaker.pyNN.protocols.RetinaKey attribute), 827
PKT (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
PKT (spinnman.model.enums.RunTimeError attribute), 330
Placement (class in pacman.model.placements), 150
Placement (class in pacman.model.placements.placement), 147
placement_report_with_application_graph_by_core() (in module pacman.operations.algorithm_reports.reports), 181
placement_report_with_application_graph_by_vertex() (in module pacman.operations.algorithm_reports.reports), 181
placement_report_without_application_graph_by_core() (in module pacman.operations.algorithm_reports.reports), 181
placement_report_without_application_graph_by_vertex() (in module pacman.operations.algorithm_reports.reports), 182
Placements (class in pacman.model.placements), 150
Placements (class in pacman.model.placements.placements), 148
placements (pacman.model.placements.Placements attribute), 152
placements (pacman.model.placements.placements.Placements attribute), 149
placements (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
placements (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
placements (spinn_front_end_common.utilities.FailedState attribute), 583
placements (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 583
placements (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
PlacementsProvenanceGatherer (class in spinn_front_end_common.interface.interface_functions), 539
PlacementsProvenanceGatherer (class in spinn_front_end_common.interface.interface_functions.placement 534
PlacementTuple (class in pacman.operations.multi_cast_router_check_functionality.valid_route 185
placer_reports_with_application_graph() (in module pacman.operations.algorithm_reports.reports), 182
placer_reports_without_application_graph() (in module pacman.operations.algorithm_reports.reports), 182
plot_spikes() (in module spynnaker.plot_utils), 838
POINT_TO_POINT (spinn_front_end_common.utilities.utility_objs.dpri_fl attribute), 572
POINT_TO_POINT (spinnman.messages.scp.enums.signal.SignalType attribute), 277
POINT_TO_POINT (spinnman.model.enums.diagnostic_filter_packet_type.DiagnosticFilter attribute), 326
POINT_TO_POINT (spinnman.model.enums.DiagnosticFilterPacketType attribute), 330
POISSON_PARAMS_REGION (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine attribute), 799
POISSON_PARAMS_REGION (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVe attribute), 805
poll_individual_sensor_continuously() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic method), 819
poll_individual_sensor_continuously() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
poll_individual_sensor_continuously_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 819
poll_individual_sensor_continuously_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
poll_sensors_once() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic method), 819
poll_sensors_once() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
poll_sensors_once_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 819
poll_sensors_once_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824

pool_size (spinnman.messages.scp.impl.iptag_get_info_response.IPTagGetInfoResponse attribute), 288
pop() (spinn_machine.utilities.ordered_set.OrderedSet method), 21
pop() (spinn_utilities.ordered_set.OrderedSet method), 10
POPULATION_BASED_REGIONS (class in spynnaker.pyNN.utilities.constants), 828
POPULATION_TABLE (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 828
PopulationMachineVertex (class in spynnaker.pyNN.models.neuron), 792
PopulationMachineVertex (class in spynnaker.pyNN.models.neuron.population_machine_vertex), 784
PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES (class in spynnaker.pyNN.models.neuron), 792
PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES (class in spynnaker.pyNN.models.neuron.population_machine_vertex), 784
port (pacman.model.resources.iptag_resource.IPtagResource attribute), 154
port (pacman.model.resources.IPtagResource attribute), 161
port (pacman.model.resources.reverse_iptag_resource.ReverseIPtagResource attribute), 156
port (pacman.model.resources.ReverseIPtagResource attribute), 162
port (pacman.model.resources.specific_board_iptag_resource.SpecificBoardIPtagResource attribute), 157
port (pacman.model.resources.specific_board_reverse_iptag_resource.SpecificBoardReverseIPtagResource attribute), 158
port (spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.LivePacketGatherParameters attribute), 573
port (spinn_front_end_common.utilities.utility_objs.LivePacketGatherParameters attribute), 577
port (spinn_machine.tags.abstract_tag.AbstractTag attribute), 18
port (spinn_machine.tags.AbstractTag attribute), 20
port (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector attribute), 810
port (spynnaker.pyNN.models.utility_models.SpikeInjector attribute), 813
port_num (spinnman.connections.socket_address_with_chip.SocketAddressWithChip attribute), 245
port_num (spinnman.connections.SocketAddressWithChip attribute), 247
port_num (spinnman.model.bmp_connection_data.BMPConnectionData attribute), 332
port_num (spinnman.model.BMPConnectionData attribute), 348
post_vertex (pacman.model.graphs.abstract_edge.AbstractEdge attribute), 137
post_vertex (pacman.model.graphs.AbstractEdge attribute), 142
post_vertex (pacman.model.graphs.application.application_edge.ApplicationEdge attribute), 110
post_vertex (pacman.model.graphs.application.ApplicationEdge attribute), 114
post_vertex (pacman.model.graphs.machine.machine_edge.MachineEdge attribute), 130
post_vertex (pacman.model.graphs.machine.MachineEdge attribute), 134
POWER_DOWN (spinnman.messages.scp.enums.Signal attribute), 280
POWER_DOWN (spinnman.messages.scp.enums.signal.Signal attribute), 277
POWER_OFF (spinnman.messages.scp.enums.power_command.PowerCommand attribute), 275
POWER_OFF (spinnman.messages.scp.enums.PowerCommand attribute), 279
power_off() (spinnman.transceiver.Transceiver method), 398
power_off_machine() (spinnman.transceiver.Transceiver method), 399
POWER_ON (spinnman.messages.scp.enums.power_command.PowerCommand attribute), 275
POWER_ON (spinnman.messages.scp.enums.PowerCommand attribute), 279
power_on() (spinnman.transceiver.Transceiver method), 399
power_on_machine() (spinnman.transceiver.Transceiver method), 399
PowerCommand (class in spinnman.messages.scp.enums), 279
power_command_parameters (spinnman.messages.scp.enums.power_command.PowerCommandParameters attribute), 275
POWERED_DOWN (spinnman.model.enums.cpu_state.CPUState attribute), 325
POWERED_DOWN (spinnman.model.enums.CPUState attribute), 329
ppf() (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats method), 827
ppf() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
PRE_SYNAPTIC_EVENT_COUNT (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES attribute), 784
PRE_SYNAPTIC_EVENT_COUNT (spynnaker.pyNN.models.neuron.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES attribute), 792
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_pair_rule.AbstractPairRule attribute), 828

attribute), 745
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence attribute), 748
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependencePfisterSpikeTriplet attribute), 746
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceRecurrent attribute), 746
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair attribute), 747
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair attribute), 747
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceVogels2011 attribute), 748
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependencePfisterSpikeTriplet attribute), 749
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceRecurrent attribute), 750
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair attribute), 750
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair attribute), 749
pre_trace_n_bytes (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceVogels2011 attribute), 751
pre_vertex (pacman.model.graphs.abstract_edge.AbstractEdge attribute), 137
pre_vertex (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition attribute), 141
pre_vertex (pacman.model.graphs.AbstractEdge attribute), 142
pre_vertex (pacman.model.graphs.AbstractOutgoingEdgePartition attribute), 146
pre_vertex (pacman.model.graphs.application.application_edge.ApplicationEdge attribute), 110
pre_vertex (pacman.model.graphs.application.ApplicationEdge attribute), 115
pre_vertex (pacman.model.graphs.impl.outgoing_edge_partition.OutgoingEdgePartition attribute), 126
pre_vertex (pacman.model.graphs.impl.OutgoingEdgePartition attribute), 129
pre_vertex (pacman.model.graphs.machine.machine_edge.MachineEdge attribute), 130
pre_vertex (pacman.model.graphs.machine.MachineEdge attribute), 134
PreAllocatedResourceContainer (class in pacman.model.resources), 163
PreAllocatedResourceContainer (class in pacman.model.resources.pre_allocated_resource_container), 154
PreAllocateResourceForChipPowerMonitor (class in spinn_front_end_common.interface.interface_functions.pre_alloc 535
PreAllocateResourceForExtraMonitorSupport (class in spinn_front_end_common.interface.interface_functions.prealloca 535
PreAllocateResourceForLivePacketGatherers (class in spinn_front_end_common.interface.interface_functions), 539
PreAllocateResourceForLivePacketGatherers (class in spinn_front_end_common.interface.interface_functions.pre_alloc 535
prefix (spinnman.messages.eieio.data_messages.EIEIODataHeader attribute), 263
prefix_type (spinn_front_end_common.utilities.utility_objs.live_packet_gat attribute), 573
prefix_type (spinnman.messages.eieio.data_messages.eieio_data_header.EII attribute), 259
prefix_type (spinnman.messages.eieio.data_messages.EIEIODataHeader attribute), 263
prepend_extra_pre_run_algorithms() (spinn_front_end_common.interface.abstract_spinnaker_base.Abs method), 548
PRINT_STRUCT (data_specification.enums.commands.Commands attribute), 418
PRINT_STRUCT (data_specification.enums.commands.Commands attribute), 415
print_struct() (data_specification.data_specification_generator.DataSpecifica method), 438
print_struct() (data_specification.DataSpecificationGenerator method), 476
print_text() (data_specification.data_specification_generator.DataSpecificati method), 439
print_text() (data_specification.DataSpecificationGenerator method), 477
PRINT_TXT (data_specification.enums.Commands attribute), 418
PRINT_TXT (data_specification.enums.commands.Commands attribute), 415
PRINT_VAL (data_specification.enums.Commands attribute), 418
PRINT_VAL (data_specification.enums.commands.Commands attribute), 415
print_value() (data_specification.data_specification_generator.DataSpecifica method), 439

print_value() (data_specification.DataSpecificationGenerator method), 477
problem (spinn_machine.exceptions.SpinnMachineInvalidParameterException attribute), 27
problem (spinnman.exceptions.SpinnmanInvalidPacketException attribute), 381
problem (spinnman.exceptions.SpinnmanInvalidParameterException attribute), 382
problem (spinnman.exceptions.SpinnmanInvalidParameterTypeException attribute), 382
problem (spinnman.exceptions.SpinnmanIOException attribute), 381
Processor (class in spinn_machine), 51
Processor (class in spinn_machine.processor), 34
processor_id (spinn_machine.Processor attribute), 52
processor_id (spinn_machine.processor.Processor attribute), 35
processor_ids (spinn_machine.core_subset.CoreSubset attribute), 25
processor_ids (spinn_machine.CoreSubset attribute), 43
processor_ids (spinn_machine.fixed_route_entry.FixedRouteEntry attribute), 28
processor_ids (spinn_machine.FixedRouteEntry attribute), 45
processor_ids (spinn_machine.multicast_routing_entry.MulticastRoutingEntry attribute), 34
processor_ids (spinn_machine.MulticastRoutingEntry attribute), 51
processor_state_register (spinnman.model.cpu_info.CPUInfo attribute), 337
processor_state_register (spinnman.model.CPUInfo attribute), 352
processors (spinn_machine.Chip attribute), 42
processors (spinn_machine.chip.Chip attribute), 24
PROFILE_TAG_LABELS (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex attribute), 784
PROFILE_TAG_LABELS (spynnaker.pyNN.models.neuron.PopulationMachineVertex attribute), 792
ProfileData (class in spinn_front_end_common.interface.profiling.profile_data), 541
ProfileDataGatherer (class in spinn_front_end_common.interface.interface_functions.profile_data_gatherer), 535
PROFILING (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 828
ProgressBar (class in spinn_machine.utilities.progress_bar), 22
ProgressBar (class in spinn_utilities.progress_bar), 12
ProjectionApplicationEdge (class in spynnaker.pyNN.models.neural_projections), 700
ProjectionApplicationEdge (class in spynnaker.pyNN.models.neural_projections.projection_application_edge), 697
ProjectionMachineEdge (class in spynnaker.pyNN.models.neural_projections), 700
ProjectionMachineEdge (class in spynnaker.pyNN.models.neural_projections.projection_machine_edge), 698
protocol (spynnaker.pyNN.external_devices_models.push_bot.push_bot_eth attribute), 626
protocol (spynnaker.pyNN.external_devices_models.push_bot.push_bot_eth attribute), 633
protocol_instance (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 819
protocol_instance (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
protocol_property (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bo attribute), 649
protocol_property (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBot attribute), 649
PROVENANCE_DATA (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGION attribute), 828
PROVENANCE_REGION (spinn_front_end_common.utility_models.command_sender_mac attribute), 589
PROVENANCE_REGION (spinn_front_end_common.utility_models.CommandSenderMach attribute), 605
PROVENANCE_REGION (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine attribute), 799
PROVENANCE_REGION (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVe attribute), 805
ProvenanceDataItem (class in spinn_front_end_common.utilities.utility_objs), 577
ProvenanceDataItem (class in spinn_front_end_common.utilities.utility_objs.provenance_data_i 574
ProvenanceJSONWriter (class in spinn_front_end_common.interface.interface_functions), 539
ProvenanceJSONWriter (class in spinn_front_end_common.interface.interface_functions.provenan 535
ProvenanceXMLWriter (class in spinn_front_end_common.interface.interface_functions),

540
ProvenanceXMLWriter (class in spinn_front_end_common.interface.interface_functions.provenance_xml_writer), 535
provide_injectables() (in module pacman.executor.injection_decorator), 95
ProvidesKeyToAtomMappingImpl (class in spinn_front_end_common.abstract_models.impl), 495
ProvidesKeyToAtomMappingImpl (class in spinn_front_end_common.abstract_models.impl.provides_key_to_atom_mapping_impl), 494
ProvidesProvenanceDataFromMachineImpl (class in spinn_front_end_common.interface.provenance), 545
ProvidesProvenanceDataFromMachineImpl (class in spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl), 544
ProvidesProvenanceDataFromMachineImpl.PROVENANCE_DATA_ENTRIES (class in spinn_front_end_common.interface.provenance), 545
ProvidesProvenanceDataFromMachineImpl.PROVENANCE_DATA_ENTRIES (class in spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl), 544
pull_off_cached_lists() (in module spynnaker.pyNN.models.common), 676
pull_off_cached_lists() (in module spynnaker.pyNN.models.common.recording_utils), 672
PUSH_BOT (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES attribute), 818
PUSH_BOT (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES attribute), 823
push_bot_laser_config_active_time() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
push_bot_laser_config_active_time() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
push_bot_laser_config_active_time_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
push_bot_laser_config_active_time_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
push_bot_laser_config_total_period() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
push_bot_laser_config_total_period() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
push_bot_laser_config_total_period_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
push_bot_laser_config_total_period_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
push_bot_laser_set_frequency() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
push_bot_laser_set_frequency() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 824
push_bot_laser_set_frequency_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
push_bot_laser_set_frequency_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 824
push_bot_led_back_active_time() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 819
push_bot_led_back_active_time() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_led_back_active_time_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 819
push_bot_led_back_active_time_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_led_front_active_time() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_led_front_active_time() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_led_front_active_time_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_led_front_active_time_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_led_set_frequency() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_led_set_frequency() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_led_set_frequency_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_led_set_frequency_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_led_total_period() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820

push_bot_led_total_period() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_led_total_period_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_led_total_period_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_motor_0_leaking_towards_zero() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_motor_0_leaking_towards_zero() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_motor_0_leaking_towards_zero_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_motor_0_leaking_towards_zero_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_motor_0_permanent() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_motor_0_permanent() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_motor_0_permanent_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_motor_0_permanent_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_motor_1_leaking_towards_zero() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_motor_1_leaking_towards_zero() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_motor_1_leaking_towards_zero_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_motor_1_leaking_towards_zero_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_motor_1_permanent() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_motor_1_permanent() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_motor_1_permanent_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_motor_1_permanent_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_speaker_config_active_time() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_speaker_config_active_time() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_speaker_config_active_time_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_speaker_config_active_time_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_speaker_config_total_period() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_speaker_config_total_period() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_speaker_config_total_period_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_speaker_config_total_period_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_speaker_set_melody() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_speaker_set_melody() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_speaker_set_melody_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_speaker_set_melody_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
push_bot_speaker_set_tone() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
push_bot_speaker_set_tone() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
push_bot_speaker_set_tone_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
push_bot_speaker_set_tone_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
PushBotEthernetDevice (class in spynnaker.pyNN.models.push_bot.push_bot_etherne 632

PushBotEthernetDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 625
PushBotEthernetLaserDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 633
PushBotEthernetLaserDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 626
PushBotEthernetLEDDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 634
PushBotEthernetLEDDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 627
PushBotEthernetMotorDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 635
PushBotEthernetMotorDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 628
PushBotEthernetRetinaDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 635
PushBotEthernetRetinaDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 629
PushBotEthernetSpeakerDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 636
PushBotEthernetSpeakerDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 629
PushBotLaser (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 640
PushBotLaser (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 639
PushBotLED (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 640
PushBotLED (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 639
PushBotLifEthernet (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_protocol), 624
PushBotLifEthernet (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_protocol), 622
PushBotLifSpinnakerLink (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_protocol), 624
PushBotLifSpinnakerLink (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_protocol), 623
PushBotMotor (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 640
PushBotMotor (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 639
PushBotRetinaConnection (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 636
PushBotRetinaConnection (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 630
PushBotRetinaResolution (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 641
PushBotRetinaResolution (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 639
PushBotRetinaViewer (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 641
PushBotRetinaViewer (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 640
PushBotSpeaker (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 641
PushBotSpeaker (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 640
PushBotSpiNNakerLinkLaserDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker), 646
PushBotSpiNNakerLinkLaserDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker), 642
PushBotSpiNNakerLinkLEDDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker), 646
PushBotSpiNNakerLinkLEDDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker), 643
PushBotSpiNNakerLinkMotorDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker), 647
PushBotSpiNNakerLinkMotorDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker), 644
PushBotSpiNNakerLinkRetinaDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker), 647

PushBotSpiNNakerLinkRetinaDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol), 644
PushBotSpiNNakerLinkSpeakerDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol), 648
PushBotSpiNNakerLinkSpeakerDevice (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol), 645
PushBotTranslator (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 637
PushBotTranslator (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_translator), 631
PushBotWIFIConnection (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet), 637
PushBotWIFIConnection (class in spynnaker.pyNN.external_devices_models.push_bot.push_bot_wifi_connection), 631
pwm_pin_output_timer_a_channel_0_ratio() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_a_channel_0_ratio() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
pwm_pin_output_timer_a_channel_0_ratio_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
pwm_pin_output_timer_a_channel_0_ratio_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
pwm_pin_output_timer_a_channel_1_ratio() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_a_channel_1_ratio() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
pwm_pin_output_timer_a_channel_1_ratio_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
pwm_pin_output_timer_a_channel_1_ratio_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
pwm_pin_output_timer_a_duration() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_a_duration() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
pwm_pin_output_timer_a_duration_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
pwm_pin_output_timer_a_duration_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
pwm_pin_output_timer_b_channel_0_ratio() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_b_channel_0_ratio() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
pwm_pin_output_timer_b_channel_0_ratio_key() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_b_channel_0_ratio_key() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
pwm_pin_output_timer_b_channel_1_ratio() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_b_channel_1_ratio() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
pwm_pin_output_timer_b_channel_1_ratio_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
pwm_pin_output_timer_b_channel_1_ratio_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
pwm_pin_output_timer_b_duration() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_b_duration() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 825
pwm_pin_output_timer_b_duration_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
pwm_pin_output_timer_b_duration_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 825
pwm_pin_output_timer_c_channel_0_ratio() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 820
pwm_pin_output_timer_c_channel_0_ratio() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
pwm_pin_output_timer_c_channel_0_ratio_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 820
pwm_pin_output_timer_c_channel_0_ratio_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
pwm_pin_output_timer_c_channel_1_ratio() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821

pwm_pin_output_timer_c_channel_1_ratio() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
pwm_pin_output_timer_c_channel_1_ratio_key() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
pwm_pin_output_timer_c_channel_1_ratio_key() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
pwm_pin_output_timer_c_duration() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
pwm_pin_output_timer_c_duration() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
pwm_pin_output_timer_c_duration_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
pwm_pin_output_timer_c_duration_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
PyNNPopulationCommon (class in spynnaker.pyNN.models.pynn_population_common), 814
PyNNProjectionCommon (class in spynnaker.pyNN.models.pynn_projection_common), 816
PyNNSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics), 762
PyNNSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_synapse_dynamics), 758
PythonClassAlgorithm (class in pacman.executor.algorithm_classes), 85
PythonClassAlgorithm (class in pacman.executor.algorithm_classes.python_class_algorithm), 82
PythonFunctionAlgorithm (class in pacman.executor.algorithm_classes), 86
PythonFunctionAlgorithm (class in pacman.executor.algorithm_classes.python_function_algorithm), 83

Q

query_state_of_io_lines() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
query_state_of_io_lines() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
query_state_of_io_lines_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
query_state_of_io_lines_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826

R

RadialPlacementFromChipConstraint (class in pacman.model.constraints.placer_constraints), 106
RadialPlacementFromChipConstraint (class in pacman.model.constraints.placer_constraints.radial_placement_from_chip), 108
RadialPlacer (class in pacman.operations.placer_algorithms), 188
RadialPlacer (class in pacman.operations.placer_algorithms.radial_placer), 187
RallocException, 978
random_distribution (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 440
random_distribution (data_specification.DataSpecificationGenerator attribute), 477
random_seed (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 319
random_seed (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
RandomNumberGenerator (class in data_specification.enums), 420
RandomNumberGenerator (class in data_specification.enums.random_number_generator), 417
RandomPlacer (class in pacman.operations.rigged_algorithms.random_placer), 190
rate (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson attribute), 798
rate (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 804
RC_ARG (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_ARG (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_BUF (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_BUF (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_CMD (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_CMD (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_CPU (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_CPU (spinnman.messages.scp.enums.SCPResult attribute), 279

RC_DEAD (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_DEAD (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_LEN (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_LEN (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_OK (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_OK (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_P2P_BUSY (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_P2P_BUSY (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_P2P_NOREPLY (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_P2P_NOREPLY (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_P2P_REJECT (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_P2P_REJECT (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_P2P_TIMEOUT (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_P2P_TIMEOUT (spinnman.messages.scp.enums.SCPResult attribute), 279
RC_PKT_TX (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_PKT_TX (spinnman.messages.scp.enums.SCPResult attribute), 280
RC_PORT (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_PORT (spinnman.messages.scp.enums.SCPResult attribute), 280
RC_ROUTE (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_ROUTE (spinnman.messages.scp.enums.SCPResult attribute), 280
RC_SUM (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_SUM (spinnman.messages.scp.enums.SCPResult attribute), 280
RC_TIMEOUT (spinnman.messages.scp.enums.scp_result.SCPResult attribute), 276
RC_TIMEOUT (spinnman.messages.scp.enums.SCPResult attribute), 280
READ (data_specification.enums.Commands attribute), 418
READ (data_specification.enums.commands.Commands attribute), 415
read() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_stor method), 60
read() (spinn_storage_handlers.abstract_classes.abstract_data_reader.Abstra method), 63
read() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStora method), 65
read() (spinn_storage_handlers.abstract_classes.AbstractDataReader method), 68
read() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedB method), 70
read() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileData method), 70
read() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTe method), 71
read() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
read() (spinn_storage_handlers.BufferedFileDataStorage method), 73
read() (spinn_storage_handlers.BufferedTempfileDataStorage method), 73
read() (spinn_storage_handlers.file_data_reader.FileDataReader method), 71
read() (spinn_storage_handlers.FileDataReader method), 74
read() (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigP method), 4
read() (spinnman.utilities.io.abstract_io.AbstractIO method), 373
read() (spinnman.utilities.io.AbstractIO method), 376
read() (spinnman.utilities.io.file_io.FileIO method), 374
read() (spinnman.utilities.io.FileIO method), 377
read() (spinnman.utilities.io.memory_io.MemoryIO method), 375
read() (spinnman.utilities.io.MemoryIO method), 378
read_a_config() (in module spinn_utilities.conf_loader), 8
read_udc_data() (spinnman.transceiver.Transceiver method), 399
read_all() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_ method), 60
read_all() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataSt method), 65
read_all() (spinn_storage_handlers.buffered_bytearray_data_storage.Buffere method), 70
read_all() (spinn_storage_handlers.buffered_file_data_storage.BufferedFile method), 70

read_all() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage method), 71
read_all() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
read_all() (spinn_storage_handlers.BufferedFileDataStorage method), 73
read_all() (spinn_storage_handlers.BufferedTempfileDataStorage method), 73
read_bmp_version() (spinnman.transceiver.Transceiver method), 399
read_byte() (spinn_storage_handlers.abstract_classes.abstract_byte_reader.AbstractByteReader method), 61
read_byte() (spinn_storage_handlers.abstract_classes.AbstractByteReader method), 66
read_bytes() (spinn_storage_handlers.abstract_classes.abstract_byte_reader.AbstractByteReader method), 61
read_bytes() (spinn_storage_handlers.abstract_classes.AbstractByteReader method), 66
read_bytestring() (spinnman.messages.scp.abstract_messages.AbstractSCPResponse method), 273
read_bytestring() (spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse method), 270
read_config() (in module spinn_front_end_common.utilities.helpful_functions), 580
read_config_boolean() (in module spinn_front_end_common.utilities.helpful_functions), 580
read_config_int() (in module spinn_front_end_common.utilities.helpful_functions), 581
read_data() (in module spinn_front_end_common.utilities.helpful_functions), 581
read_data_bytestring() (spinn_front_end_common.utilities.utility_objects.ChipSelectionAndStatus.ChipSelectionAndStatus method), 568
read_data_bytestring() (spinnman.messages.scp.abstract_messages.AbstractSCPResponse method), 273
read_data_bytestring() (spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse method), 271
read_data_bytestring() (spinnman.messages.scp.impl.check_ok_response.CheckOKResponse method), 282
read_data_bytestring() (spinnman.messages.scp.impl.CheckOKResponse method), 297
read_data_bytestring() (spinnman.messages.scp.impl.count_state_response.CountStateResponse method), 283
read_data_bytestring() (spinnman.messages.scp.impl.get_chip_info_response.GetChipInfoResponse method), 286
read_data_bytestring() (spinnman.messages.scp.impl.get_version_response.GetVersionResponse method), 287
read_data_bytestring() (spinnman.messages.scp.impl.iptag_get_info_response.IPTagGetInfoResponse method), 288
read_eieio_command_message() (in module spinnman.messages.eieio), 267
read_eieio_command_header() (in module spinnman.messages.eieio.create_eieio_command), 266
read_eieio_data_message() (in module spinnman.messages.eieio), 268
read_eieio_data_message() (in module spinnman.messages.eieio.create_eieio_data), 266
read_files (spinn_utilities.configs.camel_case_config_parser.CamelCaseConfigParser attribute), 4
read_fixed_route() (spinnman.processes.read_fixed_route_routing_entry_process.ReadFixedRoute method), 366
read_fixed_route() (spinnman.processes.ReadFixedRouteRoutingEntryProcess method), 371
read_fixed_route() (spinnman.transceiver.Transceiver method), 399
read_fpga_register() (spinnman.transceiver.Transceiver method), 399
read_from_int() (spinnman.model.diagnostic_filter.DiagnosticFilter static method), 340
read_from_int() (spinnman.model.DiagnosticFilter static method), 355
read_in_data_from_file() (in module spynnaker.pyNN.utilities.utility_calls), 831
read_int() (spinn_storage_handlers.abstract_classes.abstract_byte_reader.AbstractByteReader method), 61
read_int() (spinn_storage_handlers.abstract_classes.AbstractByteReader method), 67
read_iobuf() (spinnman.processes.read_iobuf_process.ReadIOBufProcess method), 366
read_iobuf() (spinnman.processes.ReadIOBufProcess method), 370
read_link_memory() (spinnman.processes.read_memory_process.ReadMemoryProcess method), 366
read_link_memory() (spinnman.processes.ReadMemoryProcess method), 370
read_long() (spinn_storage_handlers.abstract_classes.abstract_byte_reader.AbstractByteReader method), 62
read_long() (spinn_storage_handlers.abstract_classes.AbstractByteReader method), 67

read_memory() (spinnman.processes.read_memory_process.ReadMemoryProcess method), 366
read_memory() (spinnman.processes.ReadMemoryProcess method), 370
read_memory() (spinnman.transceiver.Transceiver method), 400
read_neighbour_memory() (spinnman.transceiver.Transceiver method), 400
READ_PARAM (data_specification.enums.Commands attribute), 418
READ_PARAM (data_specification.enums.commands.Commands attribute), 415
read_parameters_from_machine() (spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set.AbstractReadParametersBeforeSet method), 667
read_parameters_from_machine() (spynnaker.pyNN.models.abstract_models.AbstractReadParametersBeforeSet method), 668
read_parameters_from_machine() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 782
read_parameters_from_machine() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 790
read_parameters_from_machine() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
read_parameters_from_machine() (spynnaker.pyNN.models.neuron.SynapticManager method), 792
read_parameters_from_machine() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 798
read_parameters_from_machine() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 804
read_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics method), 756
read_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics method), 762
read_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics method), 760
read_plastic_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 764
read_short() (spinn_storage_handlers.abstract_classes.abstract_data_reader.AbstractDataReader method), 62
read_short() (spinn_storage_handlers.abstract_classes.AbstractDataReader method), 67
read_spikes_from_file() (in module spynnaker.pyNN.utilities.utility_calls), 831
read_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_syn method), 756
read_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSyn method), 762
read_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics method), 758
read_static_synaptic_data() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 763
read_synapses() (spynnaker.pyNN.models.neuron.synapse_io.abstract_syna method), 765
read_synapses() (spynnaker.pyNN.models.neuron.synapse_io.AbstractSyna method), 766
read_synapses() (spynnaker.pyNN.models.neuron.synapse_io.synapse_io_r method), 765
read_synapses() (spynnaker.pyNN.models.neuron.synapse_io.SynapseIORo method), 766
read_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstr method), 743
read_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.Abst method), 744
read_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.syna method), 744
read_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.syna method), 741
read_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.Syna method), 745
read_synaptic_data() (spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.Syna method), 744
READ_TYPES (in module spinnman.constants), 381
read_value() (data_specification.data_specification_generator.DataSpecifica method), 440
read_value() (data_specification.DataSpecificationGenerator method), 477
ReadADC (class in spinnman.messages.scp.impl.read_adc), 289
read() (spinn_storage_handlers.abstract_classes.abstract_data_reader.Abs method), 64
read_byte() (spinn_storage_handlers.abstract_classes.AbstractDataReader method), 68
read_byte() (spinn_storage_handlers.file_data_reader.FileDataReader method), 72

readall() (spinn_storage_handlers.FileDataReader method), 74
ReadFixedRouteRoutingEntryProcess (class in spinnman.processes), 371
ReadFixedRouteRoutingEntryProcess (class in spinnman.processes.read_fixed_route_routing_entry_process), 365
ReadFPGARegister (class in spinnman.messages.scp.impl), 301
ReadFPGARegister (class in spinnman.messages.scp.impl.read_fpga_register), 289
readinto() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage method), 60
readinto() (spinn_storage_handlers.abstract_classes.abstract_data_reader.AbstractDataReader method), 64
readinto() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage method), 65
readinto() (spinn_storage_handlers.abstract_classes.AbstractDataReader method), 69
readinto() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedBytearrayDataStorage method), 70
readinto() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage method), 70
readinto() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage method), 71
readinto() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
readinto() (spinn_storage_handlers.BufferedFileDataStorage method), 73
readinto() (spinn_storage_handlers.BufferedTempfileDataStorage method), 73
readinto() (spinn_storage_handlers.file_data_reader.FileDataReader method), 72
readinto() (spinn_storage_handlers.FileDataReader method), 74
ReadIOBufProcess (class in spinnman.processes), 370
ReadIOBufProcess (class in spinnman.processes.read_iobuf_process), 366
ReadLink (class in spinnman.messages.scp.impl), 301
ReadLink (class in spinnman.messages.scp.impl.read_link), 290
ReadMemory (class in spinnman.messages.scp.impl), 302
ReadMemory (class in spinnman.messages.scp.impl.read_memory), 290
ReadMemoryProcess (class in spinnman.processes), 370
ReadMemoryProcess (class in spinnman.processes.read_memory_process), 366
ReadRouterDiagnosticsProcess (class in spinnman.processes), 370
ReadRouterDiagnosticsProcess (class in spinnman.processes.read_router_diagnostics_process), 366
ReadStatusProcess (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp 570
READY (spinnman.model.enums.cpu_state.CPUState attribute), 325
READY (spinnman.model.enums.CPUState attribute), 329
real_class (pacman.executor.algorithm_decorators.algorithm_decorator.AllO attribute), 87
real_class (pacman.executor.algorithm_decorators.algorithm_decorator.One attribute), 87
real_class (pacman.executor.algorithm_decorators.AllOf attribute), 93
real_class (pacman.executor.algorithm_decorators.OneOf attribute), 98
receive() (spinnman.connections.udp_packet_connections.udp_connection.U 235
receive() (spinnman.connections.udp_packet_connections.UDPConnection method), 239
receive() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_et method), 682
receive() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_et method), 688
receive_boot_message() (spinnman.connections.abstract_classes.spinnaker_boot_receiver.Spinna method), 224
receive_boot_message() (spinnman.connections.abstract_classes.SpinnakerBootReceiver method), 229
receive_boot_message() (spinnman.connections.udp_packet_connections.boot_connection.BootC method), 231
receive_boot_message() (spinnman.connections.udp_packet_connections.BootConnection method), 238
receive_buffer_command_message() (spinn_front_end_common.interface.buffer_management.buffer_ method), 525
receive_buffer_command_message() (spinn_front_end_common.interface.buffer_management.BufferM method), 529
receive_eieio_message() (spinnman.connections.abstract_classes.eieio_receiver.EIEIOReceiver method), 219
receive_eieio_message() (spinnman.connections.abstract_classes.EIEIOReceiver method), 225
receive_eieio_message() (spinnman.connections.udp_packet_connections.eieio_connection.EIEI method), 232
receive_eieio_message() (spinnman.connections.udp_packet_connections.EIEIOConnection method), 240
receive_ip_address() (spinn-

man.connections.udp_packet_connections.ip_address_connection.IPAddressesConnection method), 232
receive_ip_address() (spinnman.connections.udp_packet_connections.IPAddressesConnection method), 241
receive_multicast_message() (spinnman.connections.abstract_classes.multicast_receiver.MulticastReceiver method), 221
receive_multicast_message() (spinnman.connections.abstract_classes.MulticastReceiver method), 226
receive_response() (spinn_front_end_common.utilities.scp.clear_iobuf_process.ClearIOBUFProcess method), 565
receive_response() (spinn_front_end_common.utilities.scp.clear_gsyn_inhibitory_recorder.ClearGsynInhibitoryProcess method), 566
receive_response() (spinn_front_end_common.utilities.scp.update_runtime_process.UpdateRuntimeProcess method), 566
receive_response() (spinn_front_end_common.utilities.scp.UpdateRuntimeProcess method), 567
receive_scp_response() (spinnman.connections.abstract_classes.scp_receiver.SCPReceiver method), 222
receive_scp_response() (spinnman.connections.abstract_classes.SCPReceiver method), 227
receive_scp_response() (spinnman.connections.udp_packet_connections.bmp_connection.BMPConnection method), 231
receive_scp_response() (spinnman.connections.udp_packet_connections.BMPConnection method), 237
receive_scp_response() (spinnman.connections.udp_packet_connections.scamp_connection.SCAMPConnection method), 233
receive_scp_response() (spinnman.connections.udp_packet_connections.SCAMPConnection method), 242
receive_sdp_message() (spinnman.connections.abstract_classes.sdp_receiver.SDPReceiver method), 223
receive_sdp_message() (spinnman.connections.abstract_classes.SDPReceiver method), 228
receive_sdp_message() (spinnman.connections.udp_packet_connections.sdp_connection.SDPConnection method), 234
receive_sdp_message() (spinnman.connections.udp_packet_connections.SDPConnection method), 242
receive_with_address() (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection method), 235
receive_with_address() (spinnman.connections.udp_packet_connections.UDPConnection method), 239
record (spynnaker.pyNN.models.common.eieio_spike_recorder.EIEIOSpike
attribute), 671
record (spynnaker.pyNN.models.common.EIEIOSpikeRecorder attribute), 675
record (spynnaker.pyNN.models.common.multi_spike_recorder.MultiSpike
attribute), 672
record (spynnaker.pyNN.models.common.MultiSpikeRecorder attribute), 676
record (spynnaker.pyNN.models.common.spike_recorder.SpikeRecorder attribute), 673
record (spynnaker.pyNN.models.common.SpikeRecorder attribute), 676
record_gsyn_inhibitory (spynnaker.pyNN.models.common.gsyn_inhibitory_recorder.GsynInhib
attribute), 671
RECORDING (spinn_front_end_common.utility_models.chip_power_mon
attribute), 586
RECORDING (spynnaker.pyNN.utilities.constants.POPULATION_BASED
attribute), 828
RECORDING_REGION (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP
attribute), 779
RECORDING_REGION (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 787
RECORDING_SIZE_PER_ENTRY (spinn_front_end_common.utility_models.chip_power_monitor_r
attribute), 586
recording_variables (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 672
recording_variables (spynnaker.pyNN.models.common.NeuronRecorder attribute), 675
RecordingCommon (class in spynnaker.pyNN.models.recording_common), 816
REFORMAT (data_specification.enums.Commands attribute), 418
REFORMAT (data_specification.enums.commands.Commands attribute), 415
regenerate_data_specification() (spinn_front_end_common.abstract_models.abstract_rewrites_dat
method), 498
regenerate_data_specification() (spinn_front_end_common.abstract_models.AbstractRewritesDat
method), 501
regenerate_data_specification() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP
method), 785
regenerate_data_specification() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 790

regenerate_data_specification() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 785
regenerate_data_specification() (spynnaker.pyNN.models.neuron.SynapticManager method), 792
regenerate_data_specification() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 798
regenerate_data_specification() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 804
region_data (data_specification.memory_region.MemoryRegion attribute), 456
region_data (data_specification.MemoryRegion attribute), 488
region_id (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState attribute), 516
region_id (spinn_front_end_common.interface.buffer_management.storage_objects.ChannelBufferState attribute), 523
region_id (spinnman.messages.eieio.command_messages.host_send_sequenced_data.HostSendSequencedData attribute), 251
region_id (spinnman.messages.eieio.command_messages.HostSendSequencedData attribute), 255
region_id (spinnman.messages.eieio.command_messages.spinnaker_request_buffers.SpinnakerRequestBuffers attribute), 252
region_id (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffers attribute), 256
region_id() (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead method), 250
region_id() (spinnman.messages.eieio.command_messages.HostDataRead method), 255
region_id() (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData method), 253
region_id() (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData method), 257
region_sizes (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 440
region_sizes (data_specification.DataSpecificationGenerator attribute), 478
regions (data_specification.memory_region_collection.MemoryRegionCollection attribute), 456
regions (data_specification.MemoryRegionCollection attribute), 489
register_binary_search_path() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon static method), 833
register_udp_listener() (spinnman.transceiver.Transceiver method), 401
registers (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
registers (data_specification.DataSpecificationExecutorFunctions attribute), 462
registers (spinnman.model.cpu_info.CPUInfo attribute), 337
registers (spinnman.model.CPUInfo attribute), 352
registers (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
registers (spinnman.model.RouterDiagnostics attribute), 359
reinject_fixed_route (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex attribute), 592
reinject_multicast (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex attribute), 592
reinject_nearest_neighbour (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex attribute), 592
reinject_point_to_point (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex attribute), 592
reinjection_functionality_status (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex attribute), 568
reinjection_status (spinn_front_end_common.utilities.utility_objs.re_injection_status.ReInjectionStatus attribute), 574
relative_order (pacman.utilities.ConstraintOrder attribute), 241
relative_order (pacman.utilities.vertex_sorter.ConstraintOrder attribute), 249
reload_buffer_files (spinn_front_end_common.interface.buffer_management.BufferManager method), 525
reload_buffer_files (spinn_front_end_common.interface.buffer_management.interfaces.abstract_buffer_manager.AbstractBufferManager method), 529
remote_ip_address (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection attribute), 235
remote_ip_address (spinnman.connections.udp_packet_connections.UDPConnection attribute), 240
remote_port (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection attribute), 235
remote_port (spinnman.connections.udp_packet_connections.UDPConnection attribute), 240
remove_payload_logic_to_current_output() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
remove_payload_logic_to_current_output() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol

method), 826
remove_payload_logic_to_current_output_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
remove_payload_logic_to_current_output_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
repeat (spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand attribute), 598
repeat (spinn_front_end_common.utility_models.MultiCastCommand attribute), 609
REPLY_EXPECTED (spinnman.messages.sdp.sdp_flag.SDPFlag attribute), 309
REPLY_EXPECTED (spinnman.messages.sdp.SDPFlag attribute), 312
REPLY_NOT_EXPECTED (spinnman.messages.sdp.sdp_flag.SDPFlag attribute), 309
REPLY_NOT_EXPECTED (spinnman.messages.sdp.SDPFlag attribute), 312
report (spinn_front_end_common.utilities.utility_objs.ProvenanceDataItem attribute), 574
report (spinn_front_end_common.utilities.utility_objs.ProvenanceDataItem attribute), 577
report_writer (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 440
report_writer (data_specification.DataSpecificationGenerator attribute), 478
required_inputs (pacman.executor.algorithm_classes.abstract_algorithm.Algorithm attribute), 80
required_inputs (pacman.executor.algorithm_classes.AbstractAlgorithm attribute), 84
required_optional_properties (pacman.utilities.ConstraintOrder attribute), 211
required_optional_properties (pacman.utilities.vertex_sorter.ConstraintOrder attribute), 210
requires_injection() (in module pacman.executor.injection_decorator), 95
requires_mapping (spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun attribute), 495
requires_mapping (spinn_front_end_common.abstract_models.AbstractChangableAfterRun attribute), 499
requires_mapping (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 782
requires_mapping (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 790
requires_mapping (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 815
requires_mapping (spynnaker.pyNN.models.pynn_projection_common.PyNNProjectionCommon attribute), 819
requires_mapping (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray attribute), 794
requires_mapping (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson attribute), 798
requires_mapping (spynnaker.pyNN.models.spike_source.SpikeSourceArray attribute), 801
requires_mapping (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 804
requires_mapping() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics method), 758
requires_mapping() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics method), 760
requires_mapping() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 763
requires_mapping() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 764
requires_memory_regions_to_be_reloaded() (spinn_front_end_common.abstract_models.abstract_rewrites_data_specs.AbstractRewritesDataSpecification method), 498
requires_memory_regions_to_be_reloaded() (spinn_front_end_common.abstract_models.AbstractRewritesDataSpecification method), 501
requires_memory_regions_to_be_reloaded() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 782
requires_memory_regions_to_be_reloaded() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 790
requires_memory_regions_to_be_reloaded() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 798
requires_memory_regions_to_be_reloaded() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 804
RESERVE (data_specification.enums.Commands attribute), 418
RESERVE (data_specification.enums.commands.Commands attribute), 415
reserve_a_system_processor() (spinn_machine.Chip method), 42
reserve_a_system_processor() (spinn_machine.chip.Chip method), 34
reserve_memory_region()

(data_specification.data_specification_generator.DataSpecificationGenerator method), 440
reserve_memory_region() (data_specification.DataSpecificationGenerator method), 478
reserve_memory_regions() (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device method), 656
reserve_memory_regions() (spynnaker.pyNN.external_devices_models.MunichMotorDevice method), 664
reserve_memory_regions() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 798
reserve_memory_regions() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 804
reserve_profile_region() (in module spinn_front_end_common.interface.profiling.profile_utils), 542
reserve_provenance_data_region() (spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFr method), 544
reserve_provenance_data_region() (spinn_front_end_common.interface.provenance.ProvidesProvenanceDataFromMachineImpl method), 545
reserve_system_processors() (spinn_machine.Machine method), 50
reserve_system_processors() (spinn_machine.machine.Machine method), 32
reserve_system_processors() (spinn_machine.virtual_machine.VirtualMachine method), 40
reserve_system_processors() (spinn_machine.VirtualMachine method), 56
RESET (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
RESET (spinnman.model.enums.RunTimeError attribute), 330
reset() (in module spinn_utilities.logger_utils), 10
reset() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
reset() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 525
reset() (spinn_front_end_common.interface.buffer_management.BufferManager method), 530
reset() (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface method), 837
reset() (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState static method), 830
reset_algorithms() (in module pacman.executor.algorithm_decorators), 94
reset_algorithms() (in module pacman.executor.algorithm_decorators.algorithm_decorator), 88
reset_count() (spinnman.messages.eieio.data_messages.eieio_data_header.E method), 259
reset_count() (spinnman.messages.eieio.data_messages.EIEIODataHeader method), 263
reset_counters() (spinn_front_end_common.utilities.utility_objs.extra_moni method), 571
reset_number_of_neurons_per_core() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCon method), 833
reset_reinjection_counters() (spinn_front_end_common.utility_models.extra_monitor_support method), 592
reset_retina() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_proto method), 821
reset_retina() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtoco method), 826
reset_retina_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 821
reset_retina_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
RESET_TO_DEFAULT (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munic attribute), 818
RESET_TO_DEFAULT (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES attribute), 823
ResetCountersMessage (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp 568
ResetCountersProcess (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp 571
ResourceContainer (class in pacman.model.resources), 161
ResourceContainer (class in pacman.model.resources.resource_container), 155
resources_required (pacman.model.graphs.machine.machine_fpga_vertex.MachineFPGA attribute), 131
resources_required (pacman.model.graphs.machine.machine_spinnaker_link_vertex.Mach attribute), 132
resources_required (pacman.model.graphs.machine.machine_vertex.MachineVertex attribute), 133
resources_required (pacman.model.graphs.machine.MachineFPGAVertex attribute), 135
resources_required (pac-

man.model.graphs.machine.MachineSpiNNakerLinkVertex attribute), 135
resources_required (pacman.model.graphs.machine.MachineVertex attribute), 136
resources_required (pacman.model.graphs.machine.simple_machine_vertex.SimpleMachineVertex attribute), 133
resources_required (pacman.model.graphs.machine.SimpleMachineVertex attribute), 136
resources_required (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex attribute), 587
resources_required (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 590
resources_required (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex attribute), 606
resources_required (spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex.ExtraMonitorSupportMachineVertex attribute), 592
resources_required (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex attribute), 597
resources_required (spinn_front_end_common.utility_models.live_packet_gather_machine_vertex.LivePacketGatherMachineVertex attribute), 608
resources_required (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex attribute), 603
resources_required (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex attribute), 614
resources_required (spynnaker.pyNN.models.neuron.population_machine_vertexes.PopulationMachineVertex attribute), 785
resources_required (spynnaker.pyNN.models.neuron.PopulationMachineVertex attribute), 793
resources_required (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex attribute), 800
resources_required (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVertex attribute), 806
resources_required (spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex.DelayExtensionMachineVertex attribute), 807
resources_required (spynnaker.pyNN.models.utility_models.DelayExtensionMachineVertex attribute), 810
ResourceTracker (class in pacman.utilities.utility_objs), 204
ResourceTracker (class in pacman.utilities.utility_objs.resource_tracker), 199
response (spinnman.exceptions.SpinnmanUnexpectedResponseCodeException attribute), 383
result (spinnman.messages.scp.scp_response_header.SCPResponseHeader attribute), 308
result (spinnman.messages.scp.SCPResponseHeader attribute), 309
resume() (spinn_front_end_common.interface.buffer_management.buffer_n method), 525
resume() (spinn_front_end_common.interface.buffer_management.BufferM method), 530
resume() (spinn_front_end_common.interface.buffer_management.storage_ method), 511
resume() (spinn_front_end_common.interface.buffer_management.storage_ method), 519
RetinaKey (class in spynnaker.pyNN.protocols), 827
RetinaKey (class in spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 831
RetinaPayload (class in spynnaker.pyNN.protocols), 827
RetinaPayload (class in spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol), 832
reverse_ip_tags (pacman.model.tags.Tags attribute), 180
reverse_ip_tags (pacman.model.tags.AbstractTags attribute), 179
reverse_iptags (pacman.model.resources.resource_container.ResourceContainer attribute), 156
reverse_iptags (pacman.model.resources.reverse_iptag_resource.ReverseIPTagResource attribute), 161
ReverseIPTag (class in spinn_machine.tags), 19
ReverseIPTag (class in spinn_machine.tags.reverse_iptag), 19
ReverseIpTagMultiCastSourceMachineVertex (class in spinn_front_end_common.utility_models), 609
ReverseIpTagMultiCastSource (class in spinn_front_end_common.utility_models.reverse_ip_tag_multi_c 599
ReverseIPTagMulticastSpikeSourcePoissonMachineVertex (class in spinn_front_end_common.utility_models), 612
ReverseIPTagMulticastSourceMachineVertex (class in spinn_front_end_common.utility_models.reverse_ip_tag_multica 601
ReverseIPtagResource (class in pacman.model.resources), 165
ReverseIPtagResource (class in pacman.model.resources.reverse_iptag_resource), 156
ReverseIPtagResource (class in pacman.model.resources.specific_board_reverse_iptag_resource), 158
ReverseIPTagSet (class in spinnman.messages.scp.impl), 302
ReverseIPTagSet (class in spinnman.messages.scp.impl.reverse_iptag_set), 291
rewind() (spinn_front_end_common.interface.buffer_management.buffer_m method), 504

rewind() (spinn_front_end_common.interface.buffer_management.buffer_sigma method), 507
rewind() (spinn_front_end_common.interface.buffer_management.buffer_model method), 505
rewind() (spinn_front_end_common.interface.buffer_management.buffer_models.sendsbuffersFromHostPreBufferedVertex method), 508
rewind() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region.BufferedSendingRegion method), 514
rewind() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedSendingRegion method), 521
RIGHT_RETINA (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
RIGHT_RETINA (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
RIGHT_RETINA_DISABLE (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
RIGHT_RETINA_DISABLE (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
RIGHT_RETINA_ENABLE (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
RIGHT_RETINA_ENABLE (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 664
RIGHT_RETINA_KEY_SET (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
RIGHT_RETINA_KEY_SET (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 665
RIGHT_SHIFT (data_specification.enums.logic_operation.LogicOperation attribute), 417
RIGHT_SHIFT (data_specification.enums.LogicOperation attribute), 420
right_shift (spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.LivePacketGatherParameters attribute), 573
right_shift (spinn_front_end_common.utilities.utility_objs.LivePacketGatherParameters attribute), 577
RigMCRoute (class in pacman.operations.rig_algorithms.rig_mc_route), 188
RigPlace (class in pacman.operations.rig_algorithms.rig_place), 188
RigPlaceAndRoute (class in pacman.operations.rig_algorithms.rig_place_and_route), 188
RigRoute (class in pacman.operations.rig_algorithms.rig_route), 188
ring_buffer_sigma (spynnaker.pyNN.models.AbstractSendsBuffersFromHost attribute), 782
ring_buffer_sigma (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP attribute), 790
ring_buffer_sigma (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager attribute), 783
ring_buffer_sigma (spynnaker.pyNN.models.neuron.SynapticManager attribute), 792
rng (data_specification.data_specification_generator.DataSpecificationGene attribute), 440
rng (data_specification.DataSpecificationGenerator attribute), 478
RoundRobinConnectionSelector (class in spinnman.processes.round_robin_connection_selector), 367
route (pacman.operations.router_compressors.mundys_router_compressor.r attribute), 191
route_translation (pacman.utilities.fr_format_converters.convert_to_memo attribute), 198
Router (class in spinn_machine), 52
Router (class in spinn_machine.router), 35
router (spinn_machine.Chip attribute), 42
router (spinn_machine.chip.Chip attribute), 24
ROUTER_DEFAULT_AVAILABLE_ENTRIES (spinn_machine.Router attribute), 53
ROUTER_DEFAULT_AVAILABLE_ENTRIES (spinn_machine.router.Router attribute), 36
ROUTER_DEFAULT_CLOCK_SPEED (spinn_machine.Router attribute), 53
ROUTER_DEFAULT_CLOCK_SPEED (spinn_machine.router.Router attribute), 36
router_emergency_timeout (spinn_front_end_common.utilities.utility_objs.re_injection_statu attribute), 575
router_path_chips (pacman.operations.fixed_route_router.fixed_route_router.FixedRoute attribute), 185
ROUTER_REGISTER_REGISTERS (in module spinnman.constants), 381
router_report_from_compressed_router_tables() (in module pacman.operations.algorithm_reports.reports), 183
router_report_from_paths() (in module pacman.operations.algorithm_reports.reports), 183

router_report_from_router_tables() (in module pacman.operations.algorithm_reports.reports), 183
router_table_copy_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
router_table_copy_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
router_table_copy_address() (spinnman.model.chip_info.ChipInfo method), 333
router_table_copy_address() (spinnman.model.ChipInfo method), 349
router_time_phase_timer (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
router_time_phase_timer (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
router_timeout (spinn_front_end_common.utilities.utility_objs.re_injection_status.ReInjectionStatus attribute), 575
RouterAlloc (class in spinnman.messages.scp.impl), 303
RouterAlloc (class in spinnman.messages.scp.impl.router_alloc), 291
RouterClear (class in spinnman.messages.scp.impl), 303
RouterClear (class in spinnman.messages.scp.impl.router_clear), 291
RouterDiagnostics (class in spinnman.model), 358
RouterDiagnostics (class in spinnman.model.router_diagnostics), 343
RouterInit (class in spinnman.messages.scp.impl), 303
RouterInit (class in spinnman.messages.scp.impl.router_init), 292
RouterProvenanceGatherer (class in spinn_front_end_common.interface.interface_functions), 540
RouterProvenanceGatherer (class in spinn_front_end_common.interface.interface_functions.router_provenance_gatherer), 536
ROUTING (pacman.utilities.utility_objs.flexi_field.SUPPORTED_TAGS attribute), 199
routing_entry_key (spinn_machine.multicast_routing_entry.MulticastRoutingEntry attribute), 34
routing_entry_key (spinn_machine.MulticastRoutingEntry attribute), 51
routing_info() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.PushBotSpinnakerLinkRetinaDevice method), 644
routing_info() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.PushBotSpinnakerLinkRetinaDevice method), 648
routing_info_report() (in module pacman.operations.algorithm_reports.reports), 183
routing_infos (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 548
routing_key_partition_atom_mapping() (spinn_front_end_common.abstract_models.abstract_provides_key method), 497
routing_key_partition_atom_mapping() (spinn_front_end_common.abstract_models.AbstractProvidesKey method), 500
routing_key_partition_atom_mapping() (spinn_front_end_common.abstract_models.impl.provides_key_to method), 494
routing_key_partition_atom_mapping() (spinn_front_end_common.abstract_models.impl.ProvidesKeyTo method), 495
routing_key_partition_atom_mapping() (spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl method), 653
routing_key_partition_atom_mapping() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl method), 661
routing_tables (pacman.model.routing_tables.multicast_routing_tables.MulticastRoutingTables attribute), 178
routing_tables (pacman.model.routing_tables.MulticastRoutingTables attribute), 177
RoutingInfo (class in pacman.model.routing_info), 169
RoutingInfo (class in pacman.model.routing_info.routing_info), 166
RoutingSetup (class in spinn_front_end_common.interface.interface_functions), 540
RoutingSetup (class in spinn_front_end_common.interface.interface_functions.routing_s 536
RoutingTableFromMachineReport (class in spinn_front_end_common.utilities.report_functions), 565
RoutingTableFromMachineReport (class in spinn_front_end_common.utilities.report_functions.routing_table 565
RoutingTableLoader (class in spinn_front_end_common.interface.interface_functions), 548
RoutingTableLoader (class in spinn_front_end_common.interface.interface_functions.routing_t 536
ROW_LENGTH_MASK (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ attribute), 730
ROW_LENGTH_MASK (spynnaker.pyNN.models.neuron.master_pop_table_generators.Master attribute), 732
run (spinn_front_end_common.utilities.simulator_interface.SimulatorInterfa attribute), 583
run (spinn_front_end_common.utilities.SimulatorInterface attribute), 584

run() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
run() (spinn_front_end_common.utilities.database.database_connection.DatabaseConnection method), 555
run() (spinn_front_end_common.utilities.database.DatabaseConnection method), 559
run() (spinn_front_end_common.utilities.failed_state.FailedState method), 579
run() (spinn_front_end_common.utilities.FailedState method), 583
run() (spinnman.connections.connection_listener.ConnectionListener method), 243
run() (spinnman.connections.ConnectionListener method), 245
run() (spinnman.processes.application_run_process.ApplicationRunProcess method), 362
run() (spinnman.processes.ApplicationRunProcess method), 368
run() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon method), 833
run() (spynnaker.pyNN.connections.ethernet_control_connection.EthernetControlConnection method), 619
run() (spynnaker.pyNN.connections.EthernetControlConnection method), 620
run() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_viewer.PushBotRetinaViewer method), 640
run() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotRetinaViewer method), 641
run_application() (spinn_front_end_common.interface.interface_functions.application_runner.ApplicationRunner method), 530
run_application() (spinn_front_end_common.interface.interface_functions.ApplicationRunner method), 537
RUN_FOREVER (spinn_front_end_common.interface.simulator_state.Simulator_State attribute), 550
run_time_error (spinnman.model.cpu_info.CPUInfo attribute), 337
run_time_error (spinnman.model.CPUInfo attribute), 352
RUN_TIME_EXCEPTION (spinnman.model.enums.cpu_state.CPUState attribute), 325
RUN_TIME_EXCEPTION (spinnman.model.enums.CPUState attribute), 329
run_until_complete() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
RUNNING (spinn_front_end_common.utilities.utility_objs.executable_type.ExecutableType attribute), 572
RUNNING (spinn_front_end_common.utilities.utility_objs.ExecutableType attribute), 576
RUNNING (spinnman.model.enums.cpu_state.CPUState attribute), 325
RUNNING (spinnman.model.enums.CPUState attribute), 329
RUNNING_COMMAND_SDP_PORT (spinn_front_end_common.utilities.constants.SDP_PORTS attribute), 578
RunningStats (class in spynnaker.pyNN.utilities.running_stats), 829
RUNTIME_SDP_PORT_SIZE (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP attribute), 779
RUNTIME_SDP_PORT_SIZE (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 787
RunTimeError (class in spinnman.model.enums), 329
RunTimeError (class in spinnman.model.enums.run_time_error), 328

S

S015 (data_specification.enums.data_type.DataType attribute), 416
S015 (data_specification.enums.DataType attribute), 419
S031 (data_specification.enums.data_type.DataType attribute), 416
S031 (data_specification.enums.DataType attribute), 419
S063 (data_specification.enums.data_type.DataType attribute), 416
S063 (data_specification.enums.DataType attribute), 419
S07 (data_specification.enums.data_type.DataType attribute), 416
S07 (data_specification.enums.DataType attribute), 419
S1615 (data_specification.enums.data_type.DataType attribute), 416
S1615 (data_specification.enums.DataType attribute), 419
S3231 (data_specification.enums.data_type.DataType attribute), 416
S3231 (data_specification.enums.DataType attribute), 419
S87 (data_specification.enums.data_type.DataType attribute), 416
S87 (data_specification.enums.DataType attribute), 419
safe (spynnaker.pyNN.models.neural_projections.connectors.abstract_conne attribute), 678
safe (spynnaker.pyNN.models.neural_projections.connectors.AbstractConne attribute), 687
SafeEval (class in spinn_utilities.safe_eval), 12
SameAtomsAsVertexConstraint (class in pacman.model.constraints.partitioner_constraints), 101
SameAtomsAsVertexConstraint (class in pacman.model.constraints.partitioner_constraints.same_atoms_as_ve 102
SameChipAsConstraint (class in pacman.model.constraints.placer_constraints), 106
SameChipAsConstraint (class in pacman.model.constraints.placer_constraints.same_chip_as_constrai 105

SAMPLE_RECORDING_REGION (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVertex attribute), 586
sampling_frequency (spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex.ChipPowerMonitorMachineVerte attribute), 587
sampling_interval (spynnaker.pyNN.models.recording_common.RecordingCommon attribute), 816
SARK_VERSRION_INCORRECT (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
SARK_VERSRION_INCORRECT (spinnman.model.enums.RunTimeError attribute), 330
SATURATION_COUNT (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRI attribute), 784
SATURATION_COUNT (spynnaker.pyNN.models.neuron.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES attribute), 792
save_write_pointer() (data_specification.data_specification_generator.DataSpecificationGenerator method), 440
save_write_pointer() (data_specification.DataSpecificationGenerator method), 478
scamp_connection_selector (spinnman.transceiver.Transceiver attribute), 401
SCAMPConnection (class in spinnman.connections.udp_packet_connections), 241
SCAMPConnection (class in spinnman.connections.udp_packet_connections.scamp_connection), 233
scan_packages() (in module pacman.executor.algorithm_decorators), 94
scan_packages() (in module pacman.executor.algorithm_decorators.algorithm_decorator), 88
scp_request_header (spinnman.messages.scp.abstract_messages.AbstractSCPRequest attribute), 273
scp_request_header (spinnman.messages.scp.abstract_messages.scp_request.AbstractSCPRequest attribute), 270
scp_response_header (spinnman.messages.scp.abstract_messages.AbstractSCPResponse attribute), 273
scp_response_header (spinnman.messages.scp.abstract_messages.scp_response.AbstractSCPResponse attribute), 271
SCPClearIOBUFRequest (class in spinn_front_end_common.utilities.scp), 567
SCPClearIOBUFRequest (class in spinn_front_end_common.utilities.scp.scp_clear_iobuf_request), 566
SCPCommand (class in spinnman.messages.scp.enums), 278
SCPCommand (class in spinnman.messages.scp.enums.scp_command), 275
SCPReceiver (class in spinnman.connections.abstract_classes), 227
SCPReceiver (class in spinnman.connections.abstract_classes.scp_receiver), 222
SCPRequestHeader (class in spinnman.messages.scp), 308
SCPRequestHeader (class in spinnman.messages.scp.scp_request_header), 307
SCPRequestPipeLine (class in spinnman.connections), 246
SCPRequestPipeLine (class in spinnman.connections.scp_request_pipeline), 243
SCPResponseHeader (class in spinnman.messages.scp), 309
SCPResponseHeader (class in spinnman.messages.scp.scp_response_header), 308
SCPResult (class in spinnman.messages.scp.enums), 279
SCPResult (class in spinnman.messages.scp.enums.scp_result), 276
SCPSender (class in spinnman.connections.abstract_classes), 227
SCPSender (class in spinnman.connections.abstract_classes.scp_sender), 222
SCPUpdateRuntimeRequest (class in spinn_front_end_common.utilities.scp), 567
SCPUpdateRuntimeRequest (class in spinn_front_end_common.utilities.scp.scp_update_runtime_reque 566
sdp_header (spinnman.messages.scp.abstract_messages.AbstractSCPReque attribute), 273
sdp_header (spinnman.messages.scp.abstract_messages.AbstractSCPRespo attribute), 273
sdp_header (spinnman.messages.scp.abstract_messages.scp_request.Abstrac attribute), 270
sdp_header (spinnman.messages.scp.abstract_messages.scp_response.Abstr attribute), 271
sdp_header (spinnman.messages.sdp.sdp_message.SDPMessage attribute), 312
sdp_header (spinnman.messages.sdp.SDPMessage attribute), 315
sdp_port (pacman.model.resources.reverse_iptag_resource.ReverseIPtagRes attribute), 156
sdp_port (pacman.model.resources.ReverseIPtagResource attribute), 162

sdp_port (pacman.model.resources.specific_board_reverse_iptag_resource.ReverseIPTag attribute), 158
sdp_port (spinn_machine.tags.reverse_iptag.ReverseIPTag attribute), 20
sdp_port (spinn_machine.tags.ReverseIPTag attribute), 21
SDP_PORTS (class in spinn_front_end_common.utilities.constants), 577
SDP_RUNNING_MESSAGE_CODES (in module spinn_front_end_common.utilities.constants), 578
SDPConnection (class in spinnman.connections.udp_packet_connections), 242
SDPConnection (class in spinnman.connections.udp_packet_connections.sdp_connection), 233
SDPFlag (class in spinnman.messages.sdp), 312
SDPFlag (class in spinnman.messages.sdp.sdp_flag), 309
SDPHeader (class in spinnman.messages.sdp), 312
SDPHeader (class in spinnman.messages.sdp.sdp_header), 310
SDPMessage (class in spinnman.messages.sdp), 314
SDPMessage (class in spinnman.messages.sdp.sdp_message), 312
SDPReceiver (class in spinnman.connections.abstract_classes), 228
SDPReceiver (class in spinnman.connections.abstract_classes.sdp_receiver), 223
SDPSender (class in spinnman.connections.abstract_classes), 228
SDPSender (class in spinnman.connections.abstract_classes.sdp_sender), 223
SDRAM (class in spinn_machine), 54
SDRAM (class in spinn_machine.sdram), 37
sdram (pacman.model.resources.resource_container.ResourceContainer attribute), 156
sdram (pacman.model.resources.ResourceContainer attribute), 162
sdram (spinn_machine.Chip attribute), 42
sdram (spinn_machine.chip.Chip attribute), 24
sdram_avilable_on_chip() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 203
sdram_avilable_on_chip() (pacman.utilities.utility_objs.ResourceTracker method), 207
sdram_base_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
sdram_base_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
sdram_base_address (spinnman.model.chip_info.ChipInfo attribute), 334
sdram_base_address (spinnman.model.ChipInfo attribute), 349
sdram_calculation() (spinn_front_end_common.utility_models.chip_power_ static method), 587
sdram_clock_frequency_mhz (spinnman.messages.spinnaker_boot.system_variable_boot_values.Syst attribute), 319
sdram_clock_frequency_mhz (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
sdram_heap_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.Syst attribute), 319
sdram_heap_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
sdram_heap_address (spinnman.model.chip_info.ChipInfo attribute), 334
sdram_heap_address (spinnman.model.ChipInfo attribute), 349
sdram_tags (pacman.model.resources.resource_container.ResourceContaine attribute), 156
sdram_tags (pacman.model.resources.ResourceContainer attribute), 162
sdram_usage (pacman.model.resources.specific_chip_sdram_resource.Spec attribute), 159
sdram_usage (pacman.model.resources.SpecificChipSDRAMResource attribute), 164
sdram_usage_report_per_chip() (in module pacman.operations.algorithm_reports.reports), 184
SDRAMAlloc (class in spinnman.messages.scp.impl), 304
SDRAMAlloc (class in spinnman.messages.scp.impl.sdram_alloc), 292
SDRAMDeAlloc (class in spinnman.messages.scp.impl), 304
SDRAMDeAlloc (class in spinnman.messages.scp.impl.sdram_de_alloc), 293
SDRAMResource (class in pacman.model.resources), 162
SDRAMResource (class in pacman.model.resources.sdram_resource), 156
SECOND_STAGE (spinnman.model.enumerations.static_filter_emergency_routing_status.Di attribute), 326
SECOND_STAGE (spinn-

man.model.enums.DiagnosticFilterEmergencyRoutingStatus attribute), 330
seed (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson attribute), 798
seed (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 804
seek() (spinnman.utilities.io.abstract_io.AbstractIO method), 373
seek() (spinnman.utilities.io.AbstractIO method), 376
seek() (spinnman.utilities.io.file_io.FileIO method), 374
seek() (spinnman.utilities.io.FileIO method), 377
seek() (spinnman.utilities.io.memory_io.MemoryIO method), 375
seek() (spinnman.utilities.io.MemoryIO method), 378
seek_read() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage method), 60
seek_read() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage method), 65
seek_read() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedBytearrayDataStorage method), 70
seek_read() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage method), 70
seek_read() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage method), 71
seek_read() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
seek_read() (spinn_storage_handlers.BufferedFileDataStorage method), 73
seek_read() (spinn_storage_handlers.BufferedTempfileDataStorage method), 74
seek_write() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage method), 60
seek_write() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage method), 65
seek_write() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedBytearrayDataStorage method), 70
seek_write() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage method), 70
seek_write() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage method), 71
seek_write() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
seek_write() (spinn_storage_handlers.BufferedFileDataStorage method), 73
seek_write() (spinn_storage_handlers.BufferedTempfileDataStorage method), 74
send() (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection method), 236
send() (spinnman.connections.udp_packet_connections.UDPConnection method), 240
send() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotWIFIConnection method), 632
send() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotWIFIConnection method), 638
send_boot_message() (spinnman.connections.abstract_classes.spinnaker_boot_sender.SpinnakerBootSender method), 224
send_boot_message() (spinnman.connections.abstract_classes.SpinnakerBootSender method), 229
send_boot_message() (spinnman.connections.udp_packet_connections.boot_connection.BootC method), 231
send_boot_message() (spinnman.connections.udp_packet_connections.BootConnection method), 238
send_buffer_times (spinn_front_end_common.utility_models.reverse_ip_tag attribute), 601
send_buffer_times (spinn_front_end_common.utility_models.reverse_ip_tag attribute), 603
send_buffer_times (spinn_front_end_common.utility_models.ReverseIpTag attribute), 612
send_buffer_times (spinn_front_end_common.utility_models.ReverseIPTag attribute), 614
send_buffer_times (spinn_front_end_common.interface.buffer_management.buffe attribute), 505
send_buffer_times (spinn_front_end_common.interface.buffer_management.buffe attribute), 508
send_buffers (spinn_front_end_common.utility_models.reverse_ip_tag_mul attribute), 603
send_buffers (spinn_front_end_common.utility_models.ReverseIPTagMulti attribute), 614
send_eieio_message() (spinnman.connections.abstract_classes.eieio_sender.EIEIOSender method), 220
send_eieio_message() (spinnman.connections.abstract_classes.EIEIOSender method), 225
send_eieio_message() (spinnman.connections.udp_packet_connections.eieio_connection.EIEIO method), 232
send_eieio_message() (spinnman.connections.udp_packet_connections.EIEIOConnection method), 240
send_eieio_message_to() (spinnman.connections.udp_packet_connections.eieio_connection.EIEIO method), 232
send_eieio_message_to() (spinnman.connections.udp_packet_connections.EIEIOConnection method), 240
send_event() (spinn_front_end_common.utilities.connections.live_event_co method), 552
send_event() (spinn_front_end_common.utilities.connections.LiveEventCon method), 554
send_event() (spinn_front_end_common.utilities.connections.live_event_c method), 552
send_event() (spinn_front_end_common.utilities.connections.LiveEventCo method), 554

send_multicast_message() (spinnman.connections.abstract_classes.multicast_sender.MulticastSender method), 221
send_multicast_message() (spinnman.connections.abstract_classes.MulticastSender method), 227
send_multicast_message() (spinnman.transceiver.Transceiver method), 401
send_port_trigger_message() (in module spinnman.utilities.utility_functions), 379
send_read_notification() (spinn_front_end_common.interface.interface_functions.NotificationProtocol method), 534
send_read_notification() (spinn_front_end_common.interface.interface_functions.NotificationProtocol method), 539
send_read_notification() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 562
send_read_notification() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 563
send_request() (spinnman.connections.scp_request_pipelines.SCPRequestPipeLine method), 244
send_request() (spinnman.connections.SCPRequestPipeLine method), 246
send_scp_message() (spinnman.transceiver.Transceiver method), 402
send_scp_request() (spinnman.connections.abstract_classes.scp_sender.SCPSender method), 222
send_scp_request() (spinnman.connections.abstract_classes.SCPSender method), 228
send_scp_request() (spinnman.connections.udp_packet_connections.bmp_connection.BMPConnection method), 231
send_scp_request() (spinnman.connections.udp_packet_connections.BMPConnection method), 237
send_scp_request() (spinnman.connections.udp_packet_connections.scamp_connection.SCAMPConnection method), 233
send_scp_request() (spinnman.connections.udp_packet_connections.SCAMPConnection method), 242
send_sdp_message() (spinnman.connections.abstract_classes.sdp_sender.SDPSender method), 223
send_sdp_message() (spinnman.connections.abstract_classes.SDPSender method), 229
send_sdp_message() (spinnman.connections.udp_packet_connections.sdp_connection.SDPConnection method), 234
send_sdp_message() (spinnman.connections.udp_packet_connections.SDPConnection method), 242
send_sdp_message() (spinnman.transceiver.Transceiver method), 402
send_signal() (spinnman.transceiver.Transceiver method), 402
send_spike() (spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpynnakerLiveSpikesConnection method), 619
send_spike() (spynnaker.pyNN.connections.SpynnakerLiveSpikesConnection method), 621
send_spikes() (spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpynnakerLiveSpikesConnection method), 620
send_spikes() (spynnaker.pyNN.connections.SpynnakerLiveSpikesConnection method), 621
send_start_resume_notification() (spinn_front_end_common.interface.interface_functions.NotificationProtocol method), 534
send_start_resume_notification() (spinn_front_end_common.interface.interface_functions.NotificationProtocol method), 539
send_start_resume_notification() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 562
send_start_resume_notification() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 563
send_stop_message() (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager method), 514
send_stop_message() (spinn_front_end_common.interface.buffer_management.BufferManager method), 522
send_stop_pause_notification() (spinn_front_end_common.interface.interface_functions.NotificationProtocol method), 534
send_stop_pause_notification() (spinn_front_end_common.interface.interface_functions.NotificationProtocol method), 539
send_stop_pause_notification() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 562
send_stop_pause_notification() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 563
send_to() (spinnman.connections.udp_packet_connections.udp_connection.UDPConnection method), 236
send_to() (spinnman.connections.udp_packet_connections.UDPConnection method), 240
sender_vertices (spinn_front_end_common.interface.buffer_management.buffer_manager.BufferManager attribute), 525
sender_vertices (spinn_front_end_common.interface.buffer_management.BufferManager attribute), 530
SendsBuffersFromHostPreBufferedImpl (class in spinn_front_end_common.interface.buffer_management.buffer_manager), 507
SendsBuffersFromHostPreBufferedImpl (class in spinn_front_end_common.interface.buffer_management.buffer_manager), 504

SendSignal (class in spinnman.messages.scp.impl), 304
SendSignal (class in spinnman.messages.scp.impl.send_signal), 293
SendSingleCommandProcess (class in spinnman.processes), 370
SendSingleCommandProcess (class in spinnman.processes.send_single_command_process), 367
sensor_transmission_key() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
sensor_transmission_key() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
sent_mode_command() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol static method), 821
sent_mode_command() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol static method), 826
sequence (spinnman.messages.scp.scp_request_header.SCPRequestHeader attribute), 308
sequence (spinnman.messages.scp.scp_response_header.SCPResponseHeader attribute), 308
sequence (spinnman.messages.scp.SCPRequestHeader attribute), 309
sequence (spinnman.messages.scp.SCPResponseHeader attribute), 309
sequence_no (spinnman.messages.eieio.command_messages.host_data_read_protocol.HostDataRead attribute), 250
sequence_no (spinnman.messages.eieio.command_messages.host_data_read_ack_protocol.HostDataReadAck attribute), 250
sequence_no (spinnman.messages.eieio.command_messages.host_send_sequenced_data_protocol.HostSendSequencedData attribute), 251
sequence_no (spinnman.messages.eieio.command_messages.HostDataRead attribute), 255
sequence_no (spinnman.messages.eieio.command_messages.HostDataReadAck attribute), 256
sequence_no (spinnman.messages.eieio.command_messages.HostSendSequencedData attribute), 255
sequence_no (spinnman.messages.eieio.command_messages.spinnaker_request_buffers.SpinnakerRequestBuffers attribute), 252
sequence_no (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData attribute), 253
sequence_no (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffers attribute), 256
sequence_no (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData attribute), 257
SERIAL (spinnman.messages.scp.enums.bmp_info.BMPInfo attribute), 274
SERIAL (spinnman.messages.scp.enums.BMPInfo attribute), 278
SET (spinnman.messages.scp.enums.iptag_command.IPTagCommand attribute), 274
SET (spinnman.messages.scp.enums.IPTagCommand attribute), 279
set() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 815
set_a_plus_a_minus() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence method), 751
set_a_plus_a_minus() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence method), 753
set_command_protocol() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetLEDProtocol method), 626
set_command_protocol() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 827
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetMotorDataProtocol method), 627
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetMotorPowerProtocol method), 628
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetSensorDelayProtocol method), 630
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetSensorPollingProtocol method), 633
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetSpeakerDataProtocol method), 634
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetSpeakerPlayPitchProtocol method), 635
set_command_protocol() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetSpeakerPlayToneProtocol method), 636
set_constraint() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 815
set_finished_state() (spinn_front_end_common.utilities.globals_variables), 579
set_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 734
set_global_parameters() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 739
set_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType method), 724

set_input_type_parameters() (spynnaker.pyNN.models.neuron.input_types.AbstractInputType method), 726
set_ip_tag() (spinnman.transceiver.Transceiver method), 403
set_led() (spinnman.transceiver.Transceiver method), 403
set_leds() (spinnman.transceiver.Transceiver method), 404
set_machine() (spinnman.processes.most_direct_connections.MostDirectConnectionSelector method), 365
set_machine() (spinnman.processes.MostDirectConnectionSelector method), 370
set_mapping_constraint() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 815
set_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc.IFCondExpStoc static method), 709
set_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_delta.IFCurrDelta static method), 711
set_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive.IFCurrExpCa2Adaptive static method), 714
set_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCondExpStoc static method), 722
set_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrDelta static method), 723
set_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrExpCa2Adaptive static method), 723
set_mode() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
set_mode() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
set_mode_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
set_mode_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
set_model_based_max_atoms_per_core() (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon method), 815
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_istat.EIFCondAlphaIsfaIstat static method), 705
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.EIFConductanceAlphaPopulation static method), 718
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.hh_cond_exp.HHCondExp static method), 706
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.HHCondExp static method), 719
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_cond_alpha.IFCondAlpha static method), 707
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_cond_exp_base.IFCondExpBase static method), 708
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_alpha.IFCurrAlpha static method), 710
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base.IFCurrDualExpBase static method), 712
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.IFCurrExpBase static method), 713
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.if_facets_hardware1.IFFacetsC static method), 715
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCondAlpha static method), 718
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCondExpBase static method), 719
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrAlpha static method), 720
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrDualExpBase static method), 720
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFCurrExpBase static method), 721
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IFFacetsConductancePopulatio static method), 731
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base.IzkCondE static method), 716
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base.IzkCurrExp static method), 717
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IzkCondExpBase static method), 722
set_model_max_atoms_per_core() (spynnaker.pyNN.models.neuron.builds.IzkCurrExpBase static method), 722
set_model_max_atoms_per_core() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourc static method), 794
set_model_max_atoms_per_core() (spyn-

naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson static method), 798
set_model_max_atoms_per_core() (spynnaker.pyNN.models.spike_source.SpikeSourceArray static method), 801
set_model_max_atoms_per_core() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson static method), 805
set_n_chips_required() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 548
set_n_keys_for_partition() (pacman.model.routing_info.dict_based_machine_partition_n_keys_map.DictBasedMachinePartitionNKeysMap method), 165
set_n_keys_for_partition() (pacman.model.routing_info.DictBasedMachinePartitionNKeysMap method), 168
set_n_machine_time_steps() (spinn_front_end_common.abstract_models.impl.needs_n_machine_time_steps.NeedsNMachineTimeSteps method), 494
set_n_machine_time_steps() (spinn_front_end_common.abstract_models.impl.NeedsNMachineTimeSteps method), 495
set_neural_parameters() (spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 734
set_neural_parameters() (spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 739
set_neural_parameters() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh_impl.NeuronModelIzh method), 736
set_neural_parameters() (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate method), 737
set_neural_parameters() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 740
set_neural_parameters() (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate method), 742
set_number_of_neurons_per_core() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon method), 834
set_number_of_neurons_per_core() (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface method), 837
set_number_of_neurons_per_core() (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState method), 830
set_output_pattern_for_payload() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
set_output_pattern_for_payload() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
set_output_pattern_for_payload_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
set_output_pattern_for_payload_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
set_packet_types() (spinn_front_end_common.utilities.utility_objs.extra_machine_resources.ExtraMachineResources method), 571
set_parameters() (spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input_ca2_adaptive.AbstractAdditionalInputCa2Adaptive method), 702
set_parameters() (spynnaker.pyNN.models.neuron.additional_inputs.AbstractAdditionalInputCa2Adaptive method), 704
set_parameters() (spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive_impl.AdditionalInputCa2AdaptiveImpl method), 703
set_parameters() (spynnaker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Adaptive method), 708
set_payload_pins_to_high_impedance() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
set_payload_pins_to_high_impedance() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
set_payload_pins_to_high_impedance_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
set_payload_pins_to_high_impedance_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
set_projection_information() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 678
set_projection_information() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
set_recording() (spynnaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable method), 670
set_recording() (spynnaker.pyNN.models.common.AbstractNeuronRecordable method), 674
set_recording() (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorderInterface method), 672
set_recording() (spynnaker.pyNN.models.common.NeuronRecorder method), 675
set_recording() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 782
set_recording() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 786
set_recording_spikes() (spynnaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable method), 670

set_recording_spikes() (spynnaker.pyNN.models.common.AbstractSpikeRecordable method), 675
set_recording_spikes() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 782
set_recording_spikes() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 790
set_recording_spikes() (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray method), 794
set_recording_spikes() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 798
set_recording_spikes() (spynnaker.pyNN.models.spike_source.SpikeSourceArray method), 801
set_recording_spikes() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 805
set_recording_spikes() (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector method), 810
set_recording_spikes() (spynnaker.pyNN.models.utility_models.SpikeInjector method), 813
set_register_value() (data_specification.data_specification_generator.DataSpecificationGenerator method), 441
set_register_value() (data_specification.DataSpecificationGenerator method), 479
set_reinjection_packets() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupportMachineVertex method), 592
set_reinjection_router_emergency_timeout() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupportMachineVertex method), 593
set_retina_key() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
set_retina_key() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
set_retina_key_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
set_retina_key_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
set_retina_transmission() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
set_retina_transmission() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
set_retina_transmission() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
set_retina_transmission() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
set_retina_transmission_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
set_retina_transmission_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
set_reverse_ip_tag() (spinnman.transceiver.Transceiver method), 404
set_reverse_ip_tag() (spinnman.transceiver.Transceiver method), 405
set_router_time_outs() (spinn_front_end_common.utility_models.extra_monitor_support.ExtraMonitorSupportMachineVertex method), 593
set_simulator() (in module spinn_front_end_common.utilities.globals_variables), 579
set_slice_values() (in module spynnaker.pyNN.utilities.utility_calls), 832
set_space() (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector method), 678
set_space() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector method), 687
set_structure_value() (data_specification.data_specification_generator.DataSpecificationGenerator method), 441
set_structure_value() (data_specification.DataSpecificationGenerator method), 479
set_synapse_dynamics() (spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses_parameters.AbstractAcceptsIncomingSynapsesParameters method), 666
set_synapse_dynamics() (spynnaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapsesParameters method), 668
set_synapse_dynamics() (spynnaker.pyNN.models.abstract_populatable_vertex.AbstractPopulatableVertex method), 782
set_synapse_dynamics() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 790
set_synapse_type() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 768
set_synapse_type_parameters() (spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType method), 771
set_threshold_parameters() (spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType method), 774
set_threshold_parameters() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 776

set_timeout() (spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_emergency_timeout_process.Set method), 571
set_timeout() (spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_timeout_process.SetRouterTimeout method), 571
set_up_machine_specifics() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 549
set_up_output_application_data_specifics() (in module spinn_front_end_common.utilities.helpful_functions), 581
set_up_output_application_data_specifics() (in module spinn_utilities.helpful_functions), 9
set_up_report_specifics() (in module spinn_front_end_common.utilities.helpful_functions), 581
set_up_report_specifics() (in module spinn_utilities.helpful_functions), 10
set_up_timings() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 549
set_update_completed() (spinn_front_end_common.interface.buffer_manager.storage_objects.channel_buffer_state.ChannelBufferS method), 516
set_update_completed() (spinn_front_end_common.interface.buffer_manager.storage_objects.ChannelBufferState method), 523
set_value() (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBootValues method), 317
set_value() (spynnaker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable method), 667
set_value() (spynnaker.pyNN.models.abstract_models.AbstractPopulationSettable method), 668
set_value() (spynnaker.pyNN.models.common.simple_population_settable.SimplePopulationSettable method), 673
set_value() (spynnaker.pyNN.models.common.SimplePopulationSettable method), 676
set_value() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 782
set_value() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex method), 790
set_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 758
set_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 760
set_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 763
set_value() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
set_value() (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson method), 798
set_value() (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson method), 805
set_virtual_chip_coordinates() (pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex method), 142
set_virtual_chip_coordinates() (pacman.model.graphs.AbstractVirtualVertex method), 147
set_virtual_chip_coordinates() (pacman.model.graphs.application.application_fpga_vertex.ApplicationFpgaVertex method), 111
set_virtual_chip_coordinates() (pacman.model.graphs.application.application_spinnaker_link_vertex method), 112
set_virtual_chip_coordinates() (pacman.model.graphs.application.ApplicationFPGAVertex method), 116
set_virtual_chip_coordinates() (pacman.model.graphs.application.ApplicationSpiNNakerLinkVertex method), 118
set_virtual_chip_coordinates() (pacman.model.graphs.machine.machine_fpga_vertex.MachineFPGA method), 131
set_virtual_chip_coordinates() (pacman.model.graphs.machine.machine_spinnaker_link_vertex.Mach method), 132
set_virtual_chip_coordinates() (pacman.model.graphs.machine.MachineFPGAVertex method), 135
set_virtual_chip_coordinates() (pacman.model.graphs.machine.MachineSpiNNakerLinkVertex method), 136
set_watch_dog() (spinnman.transceiver.Transceiver method), 405
set_watch_dog_to_chip() (spinnman.transceiver.Transceiver method), 405
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.abstract_conne method), 678
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.AbstractConne method), 687
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_con method), 679
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConn method), 688
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.from_list_con method), 684
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.FromListConn method), 693
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.multapse_con method), 685
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.MultapseConn method), 694
set_weights_and_delays() (spyn-

naker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector method), 685
set_weights_and_delays() (spynnaker.pyNN.models.neural_projections.connectors.OneToOneConnector method), 694
SET_WR_PTR (data_specification.enums.Commands attribute), 418
SET_WR_PTR (data_specification.enums.commands.Commands attribute), 415
set_write_pointer() (data_specification.data_specification_generator.DataSpecificationGenerator method), 442
set_write_pointer() (data_specification.DataSpecificationGenerator method), 480
SetLED (class in spinnman.messages.scp.impl), 300
SetLED (class in spinnman.messages.scp.impl.set_led), 294
SetPacketTypesProcess (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_packet_types_process), 571
SetPower (class in spinnman.messages.scp.impl), 301
SetPower (class in spinnman.messages.scp.impl.set_power), 294
SetReinjectionPacketTypesMessage (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_reinjection_packet_types_message), 568
SetRouterEmergencyTimeoutMessage (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_emergency_timeout_message), 569
SetRouterEmergencyTimeoutProcess (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_emergency_timeout_process), 571
SetRouterTimeoutMessage (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages.set_router_timeout_message), 570
SetRouterTimeoutProcess (class in spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.set_router_timeout_process), 571
SETUP (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 589
SETUP (spinn_front_end_common.utility_models.CommandSenderMachineVertex attribute), 605
shared_message_buffer_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
shared_message_buffer_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
shared_message_count_in_use (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
shared_message_count_in_use (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
shared_message_first_free_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
shared_message_first_free_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
shared_message_maximum_used (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
shared_message_maximum_used (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
SHM_CMD (spinnman.model.enums.mailbox_command.MailboxCommand attribute), 327
SHM_CMD (spinnman.model.enums.MailboxCommand attribute), 331
SHM_IDLE (spinnman.model.enums.mailbox_command.MailboxCommand attribute), 327
SHM_IDLE (spinnman.model.enums.MailboxCommand attribute), 331
SHM_MSG (spinnman.model.enums.mailbox_command.MailboxCommand attribute), 327
SHM_MSG (spinnman.model.enums.MailboxCommand attribute), 331
SHM_NOP (spinnman.model.enums.mailbox_command.MailboxCommand attribute), 327
SHM_NOP (spinnman.model.enums.MailboxCommand attribute), 331
SHM_SIGNAL (spinnman.model.enums.mailbox_command.MailboxCommand attribute), 327
SHM_SIGNAL (spinnman.model.enums.MailboxCommand attribute), 331
SHUTDOWN (spinn_front_end_common.interface.simulator_state.Simulatr attribute), 550
Signal (class in spinnman.messages.scp.enums.signal), 277
signal_handler() (spinn_front_end_common.interface.abstract_spinnaker_ba method), 549
signal_type (spinnman.messages.scp.enums.Signal attribute), 280
signal_type (spinnman.messages.scp.enums.signal.Signal attribute), 277
SignalType (class in spinnman.messages.scp.enums.signal), 277
SimpleMachineVertex (class in pacman.model.graphs.machine), 136
SimpleMachineVertex (class in pacman.model.graphs.machine.simple_machine_vertex), 133
SimplePopulationSettable (class in spynnaker.pyNN.models.common), 676
SimplePopulationSettable (class in spyn-

naker.pyNN.models.common.simple_population_settable), 673
Simulator_State (class in spinn_front_end_common.interface.simulator_states), 550
SimulatorInterface (class in spinn_front_end_common.utilities), 583
SimulatorInterface (class in spinn_front_end_common.utilities.simulator_interface), 582
SINGLE_BIT_FLAG_BIT (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch attribute), 730
SINGLE_BIT_FLAG_BIT (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch attribute), 732
SingleInput (class in pacman.executor.algorithm_decorators), 92
SingleInput (class in pacman.executor.algorithm_decorators.single_input), 90
size (pacman.model.constraints.partitioner_constraints.max_sram_size_partitioner_constraint.MaxSramSizePartitionerSameSizeAsVertexConstraint attribute), 102
size (pacman.model.constraints.partitioner_constraints.MaxSizeAtomsPerVertexConstraint attribute), 103
size (pacman.model.resources.element_free_space.ElementFreeSpace attribute), 153
size (pacman.model.resources.ElementFreeSpace attribute), 160
size (spinn_machine.SDRAM attribute), 54
size (spinn_machine.sdram.SDRAM attribute), 37
size (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIODataHeader attribute), 259
size (spinnman.messages.eieio.data_messages.eieio_data_message.EIEIODataMessage attribute), 261
size (spinnman.messages.eieio.data_messages.EIEIODataHeader attribute), 263
size (spinnman.messages.eieio.data_messages.EIEIODataMessage attribute), 265
size (spinnman.model.heap_element.HeapElement attribute), 341
size (spinnman.model.HeapElement attribute), 356
size (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 815
size() (spynnaker.pyNN.models.pynn_projection_common.PyNNProjectionCommon method), 816
SIZE_OF_A_SDRAM_ENTRY (spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression.mundy_on_chip_router_compression.MundyOnChipRouterCompression attribute), 550
size_of_channel_state() (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState static method), 516
size_of_channel_state() (spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state.ChannelBufferState static method), 523
size_of_region() (spinn_front_end_common.interface.buffer_management.storage_objects.buffering_state.BufferingState static method), 516
size_of_region() (spinn_front_end_common.interface.buffer_management.s static method), 524
Slice (class in pacman.model.graphs.common), 122
Slice (class in pacman.model.graphs.common.slice), 120
slow (spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_synapse_dy attribute), 758
slow (spynnaker.pyNN.models.neuron.synapse_dynamics.PyNNSynapseDy attribute), 762
SmallWorldConnector (class in spynnaker.pyNN.models.neural_projections.connectors), 695
SmallWorldConnector (class in spynnaker.pyNN.models.neural_projections.connectors.small_world_c 686
SocketAddress (class in spinn_front_end_common.utilities.notification_protocol), 563
SocketAddress (class in spinn_front_end_common.utilities.notification_protocol.socket_a 563
SocketAddress (class in spinnman.messages.scp.impl.iptagsetup.IptagSetupSocketAddress), 13
SocketAddressWithChip (class in spinnman.connections), 246
SocketAddressWithChip (class in spinnman.connections.socket_address_with_chip), 244
software_error_count (spinnman.model.cpu_info.CPUInfo attribute), 337
software_error_header (spinnman.model.CPUInfo attribute), 352
software_error_message_address (spinnman.model.cpu_info.CPUInfo attribute), 337
software_source_filename_address (spinnman.model.CPUInfo attribute), 352
software_source_line_number (spinnman.model.cpu_info.CPUInfo attribute), 337
software_source_line_number (spinnman.model.CPUInfo attribute), 352
software_version (spinnman.model.cpu_info.CPUInfo attribute), 337
software_version (spinnman.model.CPUInfo attribute), 353
software_watchdog_count (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBootValues attribute), 319
software_watchdog_count (spinnman.model.cpu_info.CPUInfo attribute), 337
software_watchdog_count (spinnman.model.CPUInfo attribute), 352
sort() (spinn_front_end_common.utilities.utilities.ordered_set.OrderedSet method), 17

method), 210
sort() (pacman.utilities.VertexSorter method), 211
sort_out_downed_chips_cores_links() (in module spinn_front_end_common.utilities.helpful_functions), 581
sort_vertices_by_known_constraints() (in module pacman.utilities.algorithm_utilities.placer_algorithm_utilities), 196
source_chip_x (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
source_chip_x (spinnman.messages.sdp.SDPHeader attribute), 314
source_chip_y (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
source_chip_y (spinnman.messages.sdp.SDPHeader attribute), 314
source_cpu (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
source_cpu (spinnman.messages.sdp.SDPHeader attribute), 314
source_link_id (spinn_machine.Link attribute), 46
source_link_id (spinn_machine.link.Link attribute), 29
source_port (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
source_port (spinnman.messages.sdp.SDPHeader attribute), 314
source_vertex (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex attribute), 808
source_vertex (spynnaker.pyNN.models.utility_models.DelayExtensionVertex attribute), 812
source_x (spinn_machine.Link attribute), 46
source_x (spinn_machine.link.Link attribute), 29
source_y (spinn_machine.Link attribute), 46
source_y (spinn_machine.link.Link attribute), 29
sources (spinnman.model.diagnostic_filter.DiagnosticFilter attribute), 340
sources (spinnman.model.DiagnosticFilter attribute), 355
SOUTH (pacman.utilities.constants.EDGES attribute), 207
SOUTH (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
SOUTH (spinnman.model.enums.P2PTableRoute attribute), 331
SOUTH_WEST (pacman.utilities.constants.EDGES attribute), 207
SOUTH_WEST (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
SOUTH_WEST (spinnman.model.enums.P2PTableRoute attribute), 331
space (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector attribute), 678
space (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector attribute), 688
space_allocated (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
space_allocated (data_specification.DataSpecificationExecutorFunctions attribute), 462
space_available (spinnman.messages.eieio.command_messages.spinnaker_request_buffer_command.SpinnakerRequestBuffer attribute), 252
space_available (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffer attribute), 256
space_read() (spinnman.messages.eieio.command_messages.host_data_read.HostDataRead method), 250
space_read() (spinnman.messages.eieio.command_messages.HostDataRead method), 255
space_to_be_read() (spinnman.messages.eieio.command_messages.spinnaker_request_read_buffer_command.SpinnakerRequestReadBuffer method), 253
space_to_be_read() (spinnman.messages.eieio.command_messages.SpinnakerRequestReadBuffer method), 257
SpallocAllocator (class in spinn_front_end_common.interface.interface_functions), 540
SpallocAllocator (class in spinn_front_end_common.interface.interface_functions.spalloc_allocator), 536
SpallocMaxMachineGenerator (class in spinn_front_end_common.interface.interface_functions), 540
SpallocMaxMachineGenerator (class in spinn_front_end_common.interface.interface_functions.spalloc_max_machine_generator), 536
SPEAKER_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeakerParameters attribute), 640
SPEAKER_ACTIVE_TIME (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeakerParameters attribute), 641
speaker_active_time() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
speaker_active_time() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
speaker_frequency() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
speaker_frequency() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
SPEAKER_MELODY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeakerParameters attribute), 640

SPEAKER_MELODY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeaker attribute), 641
SPEAKER_TONE (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeaker attribute), 640
SPEAKER_TONE (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeaker attribute), 641
SPEAKER_TOTAL_PERIOD (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeaker attribute), 640
SPEAKER_TOTAL_PERIOD (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotSpeaker attribute), 641
speaker_total_period() (spynnaker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol static method), 818
speaker_total_period() (spynnaker.pyNN.protocols.MunichIoEthernetProtocol static method), 823
spec_reader (data_specification.data_specification_executor.DataSpecificationExecutor attribute), 421
spec_reader (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
spec_reader (data_specification.DataSpecificationExecutor attribute), 459
spec_reader (data_specification.DataSpecificationExecutorFunctions attribute), 462
spec_writer (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 443
spec_writer (data_specification.DataSpecificationGenerator attribute), 481
specific_core_resources (pacman.model.resources.pre_allocated_resource_container.PreAllocatedResourceContainer attribute), 155
specific_core_resources (pacman.model.resources.PreAllocatedResourceContainer attribute), 163
specific_iptag_resources (pacman.model.resources.pre_allocated_resource_container.PreAllocatedResourceContainer attribute), 155
specific_iptag_resources (pacman.model.resources.PreAllocatedResourceContainer attribute), 163
specific_reverse_iptag_resources (pacman.model.resources.pre_allocated_resource_container.PreAllocatedResourceContainer attribute), 155
specific_reverse_iptag_resources (pacman.model.resources.PreAllocatedResourceContainer attribute), 163
specific_sdram_usage (pacman.model.resources.pre_allocated_resource_container.PreAllocatedResourceContainer attribute), 155
specific_sdram_usage (pacman.model.resources.PreAllocatedResourceContainer attribute), 163
SpecificBoardTagResource (class in pacman.model.resources.specific_board_tag_resource), 157
SpecificChipSDRAMResource (class in pacman.model.resources.specific_chip_sdram_resource), 163
SpecificChipSDRAMResource (class in pacman.model.resources.specific_chip_sdram_resource), 159
SpecificCoreResource (class in pacman.model.resources), 164
SpecificCoreResource (class in pacman.model.resources.specific_core_resource), 159
SPIKE_HISTORY_REGION (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine attribute), 799
SPIKE_HISTORY_REGION (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVe attribute), 805
SPIKE_RECORDING_REGION (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractP attribute), 779
SPIKE_RECORDING_REGION (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 787
SPIKE_RECORDING_REGION_ID (spynnaker.pyNN.models.spike_source.spike_source_array.SpikeSourc attribute), 794
SPIKE_RECORDING_REGION_ID (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSou attribute), 796
SPIKE_RECORDING_REGION_ID (spynnaker.pyNN.models.spike_source.SpikeSourceArray attribute), 800
SPIKE_RECORDING_REGION_ID (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 802
SPIKE_RECORDING_REGION_ID (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjector attribute), 809
SPIKE_RECORDING_REGION_ID (spynnaker.pyNN.models.utility_models.SpikeInjector attribute), 812
spike_times (spynnaker.pyNN.models.spike_source.spike_source_array.Spi attribute), 795
spike_times (spynnaker.pyNN.models.spike_source.spike_source_from_file attribute), 795
spike_times (spynnaker.pyNN.models.spike_source.SpikeSourceArray attribute), 801
spike_times (spynnaker.pyNN.models.spike_source.SpikeSourceFromFile attribute), 801

SpikeInjector (class in spynnaker.pyNN.models.utility_models), 812
SpikeInjector (class in spynnaker.pyNN.models.utility_models.spike_injector), 809
SpikeRecorder (class in spynnaker.pyNN.models.common), 676
SpikeRecorder (class in spynnaker.pyNN.models.common.spike_recorder), 673
spikes_per_second (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 782
spikes_per_second (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 790
spikes_per_second (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager attribute), 785
spikes_per_second (spynnaker.pyNN.models.neuron.SynapticManager attribute), 792
SpikeSourceArray (class in spynnaker.pyNN.models.spike_source), 800
SpikeSourceArray (class in spynnaker.pyNN.models.spike_source.spike_source_array), 793
SpikeSourceFromFile (class in spynnaker.pyNN.models.spike_source), 801
SpikeSourceFromFile (class in spynnaker.pyNN.models.spike_source.spike_source_from_file), 795
SpikeSourcePoisson (class in spynnaker.pyNN.models.spike_source), 801
SpikeSourcePoisson (class in spynnaker.pyNN.models.spike_source.spike_source_poisson), 795
SpikeSourcePoissonMachineVertex (class in spynnaker.pyNN.models.spike_source), 805
SpikeSourcePoissonMachineVertex (class in spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex), 799
SpikeSourcePoissonMachineVertex.POISSON_SPIKE_SOURCE_REGIONS (class in spynnaker.pyNN.models.spike_source), 805
SpikeSourcePoissonMachineVertex.POISSON_SPIKE_SOURCE_REGIONS (class in spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex), 799
spinn5_triad_geometry (spinn_machine.spinnaker_triad_geometry.SpiNNakerTriadGeometry attribute), 39
spinn5_triad_geometry (spinn_machine.SpiNNakerTriadGeometry attribute), 55
spinn_front_end_common (module), 493, 614
spinn_front_end_common.abstract_models (module), 499
spinn_front_end_common.abstract_models.abstract_changable_after_run (module), 495
spinn_front_end_common.abstract_models.abstract_generates_data_specifi (module), 495
spinn_front_end_common.abstract_models.abstract_has_associated_binary (module), 496
spinn_front_end_common.abstract_models.abstract_machine_allocation_co (module), 496
spinn_front_end_common.abstract_models.abstract_provides_incoming_pa (module), 496
spinn_front_end_common.abstract_models.abstract_provides_key_to_atom (module), 497
spinn_front_end_common.abstract_models.abstract_provides_n_keys_for_p (module), 497
spinn_front_end_common.abstract_models.abstract_provides_outgoing_par (module), 497
spinn_front_end_common.abstract_models.abstract_recordable (module), 498
spinn_front_end_common.abstract_models.abstract_rewrites_data_specifica (module), 498
spinn_front_end_common.abstract_models.abstract_send_me_multicast_co (module), 498
spinn_front_end_common.abstract_models.abstract_supports_database_inje (module), 498
spinn_front_end_common.abstract_models.abstract_vertex_with_dependen (module), 499
spinn_front_end_common.abstract_models.impl (module), 494
spinn_front_end_common.abstract_models.impl.machine_data_specable_ve (module), 493
spinn_front_end_common.abstract_models.impl.needs_n_machine_time_st (module), 494
spinn_front_end_common.abstract_models.impl.provides_key_to_atom_ma (module), 494
spinn_front_end_common.common_model_binaries (module), 502
spinn_front_end_common.interface (module), 550
spinn_front_end_common.interface.abstract_spinnaker_base (module), 540
spinn_front_end_common.interface.buffer_management (module), 528
spinn_front_end_common.interface.buffer_management.buffer_manager (module), 524
spinn_front_end_common.interface.buffer_management.buffer_models (module), 505
spinn_front_end_common.interface.buffer_management.buffer_models.abs (module), 502
spinn_front_end_common.interface.buffer_management.buffer_models.abs (module), 503
spinn_front_end_common.interface.buffer_management.buffer_models.sen (module), 504
spinn_front_end_common.interface.buffer_management.recording_utilities

(module), 525
spinn_front_end_common.interface.buffer_management.storage_objects (module), 516
spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data (module), 508
spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region (module), 512
spinn_front_end_common.interface.buffer_management.storage_objects.buffers_sent_deque (module), 514
spinn_front_end_common.interface.buffer_management.storage_objects.channel_buffer_state (module), 515
spinn_front_end_common.interface.buffer_management.storage_objects.end_buffering_state (module), 516
spinn_front_end_common.interface.interface_functions (module), 537
spinn_front_end_common.interface.interface_functions.application_finished (module), 530
spinn_front_end_common.interface.interface_functions.application_runner (module), 530
spinn_front_end_common.interface.interface_functions.buffer_extractor (module), 530
spinn_front_end_common.interface.interface_functions.buffer_manager_creator (module), 530
spinn_front_end_common.interface.interface_functions.chip_buffer_checker (module), 530
spinn_front_end_common.interface.interface_functions.chip_buffer_extractor (module), 531
spinn_front_end_common.interface.interface_functions.chip_provenance_updater (module), 531
spinn_front_end_common.interface.interface_functions.chip_runtime_updater (module), 531
spinn_front_end_common.interface.interface_functions.database_interface (module), 531
spinn_front_end_common.interface.interface_functions.dsg_region_reloader (module), 531
spinn_front_end_common.interface.interface_functions.edge_to_n_keys_mapper (module), 531
spinn_front_end_common.interface.interface_functions.graph_binary_gatherer (module), 531
spinn_front_end_common.interface.interface_functions.graph_data_specification_writer (module), 532
spinn_front_end_common.interface.interface_functions.graph_machine_partitioner (module), 532
spinn_front_end_common.interface.interface_functions.graph_provenance_gatherer (module), 532
spinn_front_end_common.interface.interface_functions.hbp_allocator (module), 532
spinn_front_end_common.interface.interface_functions.hbp_max_machine_generator (module), 532
spinn_front_end_common.interface.interface_functions.host_execute_data_specification (module), 532
spinn_front_end_common.interface.interface_functions.insert_chip_power_monitors_to_graphs (module), 532
spinn_front_end_common.interface.interface_functions.insert_edges_to_live_packet_gatherers (module), 533
spinn_front_end_common.interface.interface_functions.insert_extra_monitors (module), 533
spinn_front_end_common.interface.interface_functions.insert_live_packet_gatherers_to_graph (module), 533
spinn_front_end_common.interface.interface_functions.load_executable_images (module), 533
spinn_front_end_common.interface.interface_functions.load_fixed_routes (module), 533
spinn_front_end_common.interface.interface_functions.locate_executable_start_type (module), 533
spinn_front_end_common.interface.interface_functions.machine_execute_data_specification (module), 533
spinn_front_end_common.interface.interface_functions.machine_generator (module), 534
spinn_front_end_common.interface.interface_functions.notification_protocol_finished (module), 534
spinn_front_end_common.interface.interface_functions.placements_provenance_gatherer (module), 534
spinn_front_end_common.interface.interface_functions.pre_allocate_resource_for_bitfield_generation (module), 535
spinn_front_end_common.interface.interface_functions.pre_allocate_resource_for_chip_power_monitor (module), 535
spinn_front_end_common.interface.interface_functions.preallocate_resource_for_extra_monitor_support (module), 535
spinn_front_end_common.interface.interface_functions.profile_data_gatherer (module), 535
spinn_front_end_common.interface.interface_functions.provenance_json_writer (module), 535
spinn_front_end_common.interface.interface_functions.provenance_xml_writer (module), 535
spinn_front_end_common.interface.interface_functions.router_provenance_gatherer (module), 536
spinn_front_end_common.interface.interface_functions.routing_setup (module), 536
spinn_front_end_common.interface.interface_functions.routing_table_loader_from_machine_report (module), 536
spinn_front_end_common.interface.interface_functions.spalloc_allocator (module), 536
spinn_front_end_common.interface.interface_functions.spalloc_max_machine_generator (module), 536
spinn_front_end_common.interface.interface_functions.tags_loader (module), 536
spinn_front_end_common.interface.interface_functions.tdma_agenda_builder (module), 537
spinn_front_end_common.interface.interface_functions.virtual_machine_generator (module), 537
spinn_front_end_common.interface.profiling (module), 543
spinn_front_end_common.interface.profiling.abstract_has_profile_data (module), 541
spinn_front_end_common.interface.profiling.profile_data (module), 541
spinn_front_end_common.interface.profiling.profile_utils

(module), 542
spinn_front_end_common.interface.provenance (module), 544
spinn_front_end_common.interface.provenance.abstract_provides_local_provenance (module), 543
spinn_front_end_common.interface.provenance.abstract_provides_provenance_data (module), 543
spinn_front_end_common.interface.provenance.pacman_provenance_extractor (module), 544
spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine (module), 544
spinn_front_end_common.interface.simulation (module), 546
spinn_front_end_common.interface.simulation.simulation_utilities (module), 546
spinn_front_end_common.interface.simulator_state (module), 550
spinn_front_end_common.mapping_algorithms (module), 551
spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression (module), 551
spinn_front_end_common.mapping_algorithms.on_chip_router_table_compression.utilities (module), 550
spinn_front_end_common.utilities (module), 583
spinn_front_end_common.utilities.connections (module), 553
spinn_front_end_common.utilities.connections.live_event_connection (module), 551
spinn_front_end_common.utilities.constants (module), 577
spinn_front_end_common.utilities.database (module), 558
spinn_front_end_common.utilities.database.database_connection (module), 555
spinn_front_end_common.utilities.database.database_reader (module), 555
spinn_front_end_common.utilities.database.database_writer (module), 557
spinn_front_end_common.utilities.exceptions (module), 578
spinn_front_end_common.utilities.failed_state (module), 578
spinn_front_end_common.utilities.function_list (module), 579
spinn_front_end_common.utilities.globals_variables (module), 579
spinn_front_end_common.utilities.helpful_functions (module), 579
spinn_front_end_common.utilities.math_constants (module), 582
spinn_front_end_common.utilities.notification_protocol (module), 563
spinn_front_end_common.utilities.notification_protocol.notification_protocol (module), 562
spinn_front_end_common.utilities.notification_protocol.socket_address (module), 563
spinn_front_end_common.utilities.report_functions (module), 565
spinn_front_end_common.utilities.report_functions.energy_report (module), 564
spinn_front_end_common.utilities.report_functions.fixed_route_from_machine (module), 564
spinn_front_end_common.utilities.report_functions.memory_map_on_chip_report_from_machine_impl (module), 564
spinn_front_end_common.utilities.report_functions.memory_map_on_host_chip_report (module), 565
spinn_front_end_common.utilities.report_functions.memory_map_on_host_report (module), 565
spinn_front_end_common.utilities.report_functions.routing_table_from_machine (module), 565
spinn_front_end_common.utilities.scp (module), 566
spinn_front_end_common.utilities.scp.clear_iobuf_process (module), 565
spinn_front_end_common.utilities.scp.scp_clear_iobuf_request (module), 566
spinn_front_end_common.utilities.scp.scp_update_runtime_request (module), 566
spinn_front_end_common.utilities.scp.update_runtime_process (module), 566
spinn_front_end_common.utilities.simulator_interface (module), 582
spinn_front_end_common.utilities.utility_objs (module), 575
spinn_front_end_common.utilities.utility_objs.dpri_flags (module), 572
spinn_front_end_common.utilities.utility_objs.executable_finder (module), 572
spinn_front_end_common.utilities.utility_objs.executable_type (module), 572
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages (module), 570
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages (module), 567
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages (module), 568
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages (module), 568
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages (module), 569
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_messages (module), 570
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes (module), 572
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes (module), 570
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes (module), 571
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes

(module), 571
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.abstract_tag_get_ip_and_port (module), 571
spinn_front_end_common.utilities.utility_objs.extra_monitor_scp_processes.get_reverse_iptag (module), 571
spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters (module), 573
spinn_front_end_common.utilities.utility_objs.provenance_data_item (module), 574
spinn_front_end_common.utilities.utility_objs.re_injection_status (module), 574
spinn_front_end_common.utility_models (module), 603
spinn_front_end_common.utility_models.chip_power_monitor (module), 584
spinn_front_end_common.utility_models.chip_power_monitor_machine_vertex (module), 585
spinn_front_end_common.utility_models.command_sender (module), 587
spinn_front_end_common.utility_models.command_sender_machine_vertex (module), 589
spinn_front_end_common.utility_models.extra_monitor_support (module), 590
spinn_front_end_common.utility_models.extra_monitor_support_machine_vertex (module), 591
spinn_front_end_common.utility_models.live_packet_gather (module), 594
spinn_front_end_common.utility_models.live_packet_gather_machine_vertex (module), 596
spinn_front_end_common.utility_models.multi_cast_command (module), 597
spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source (module), 599
spinn_front_end_common.utility_models.reverse_ip_tag_multi_cast_source_machine_vertex (module), 601
spinn_machine (module), 15, 40
spinn_machine.chip (module), 22
spinn_machine.core_subset (module), 25
spinn_machine.core_subsets (module), 26
spinn_machine.exceptions (module), 27
spinn_machine.fixed_route_entry (module), 28
spinn_machine.link (module), 28
spinn_machine.link_data_objects (module), 17
spinn_machine.link_data_objects.abstract_link_data (module), 16
spinn_machine.link_data_objects.fpga_link_data (module), 17
spinn_machine.link_data_objects.spinnaker_link_data (module), 17
spinn_machine.machine (module), 29
spinn_machine.multicast_routing_entry (module), 33
spinn_machine.processor (module), 34
spinn_machine.router (module), 35
spinn_machine.sdram (module), 37
spinn_machine.spinnaker_triad_geometry (module), 38
spinn_machine.tags (module), 20
spinn_machine.tags.abstract_tag (module), 18
spinn_machine.tags.iptag (module), 18
spinn_machine.tags.reverse_iptag (module), 19
spinn_machine.utilities (module), 22
spinn_machine.utilities.ordered_set (module), 21
spinn_machine.utilities.progress_bar (module), 22
spinn_machine.virtual_machine (module), 39
spinn_storage_handlers (module), 59, 72
spinn_storage_handlers.abstract_classes (module), 65
spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage (module), 59
spinn_storage_handlers.abstract_classes.abstract_byte_reader (module), 61
spinn_storage_handlers.abstract_classes.abstract_byte_writer (module), 62
spinn_storage_handlers.abstract_classes.abstract_context_manager (module), 63
spinn_storage_handlers.abstract_classes.abstract_data_reader (module), 63
spinn_storage_handlers.abstract_classes.abstract_data_writer (module), 64
spinn_storage_handlers.buffered_bytearray_data_storage (module), 69
spinn_storage_handlers.buffered_file_data_storage (module), 70
spinn_storage_handlers.buffered_tempfile_data_storage (module), 70
spinn_storage_handlers.exceptions (module), 71
spinn_storage_handlers.file_data_reader (module), 71
spinn_storage_handlers.file_data_writer (module), 72
spinn_utilities (module), 3, 14
spinn_utilities.abstract_base (module), 6
spinn_utilities.conf_loader (module), 7
spinn_utilities.configs (module), 5
spinn_utilities.configs.camel_case_config_parser (module), 3
spinn_utilities.configs.case_sensitive_parser (module), 4
spinn_utilities.configs.no_config_found_exception (module), 5
spinn_utilities.configs.unexpected_config_exception (module), 5
spinn_utilities.executable_finder (module), 9
spinn_utilities.helpful_functions (module), 9
spinn_utilities.log (module), 10
spinn_utilities.logger_utils (module), 10
spinn_utilities.ordered_set (module), 10
spinn_utilities.overrides (module), 11
spinn_utilities.package_loader (module), 11
spinn_utilities.progress_bar (module), 12
spinn_utilities.safe_eval (module), 12
spinn_utilities.socket_address (module), 13
spinn_utilities.testing (module), 6
spinn_utilities.testing.log_checker (module), 5

spinn_utilities.timer (module), 13
spinnaker_based_data_specification_execution() (spinn_front_end_common.interface.interface_functions.machine_execution_data_specification.SpinnakerBootDataSpecificat method), 533
spinnaker_based_data_specification_execution() (spinn_front_end_common.interface.interface_functions.MachineExecutionDataSpecification method), 539
spinnaker_link_id (pacman.model.graphs.abstract_spinnaker_link_vertex.AbstractSpiNNakerLinkVertex attribute), 141
spinnaker_link_id (pacman.model.graphs.AbstractSpiNNakerLinkVertex attribute), 146
spinnaker_link_id (pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex attribute), 112
spinnaker_link_id (pacman.model.graphs.application.ApplicationSpiNNakerLinkVertex attribute), 118
spinnaker_link_id (pacman.model.graphs.machine.machine_spinnaker_link_vertex.MachineSpiNNakerLinkVertex attribute), 132
spinnaker_link_id (pacman.model.graphs.machine.MachineSpiNNakerLinkVertex attribute), 136
spinnaker_link_id (spinn_machine.link_data_objects.spinnaker_link_data.SpinnakerLinkData attribute), 17
spinnaker_link_id (spinn_machine.link_data_objects.SpinnakerLinkData attribute), 18
spinnaker_links (spinn_machine.Machine attribute), 50
spinnaker_links (spinn_machine.machine.Machine attribute), 33
SPINNAKER_REQUEST_BUFFERS (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
SPINNAKER_REQUEST_READ_DATA (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
SpinnakerBootMessage (class in spinnman.messages.spinnaker_boot), 320
SpinnakerBootMessage (class in spinnman.messages.spinnaker_boot.spinnaker_boot_messages), 315
SpinnakerBootMessages (class in spinnman.messages.spinnaker_boot), 321
SpinnakerBootMessages (class in spinnman.messages.spinnaker_boot.spinnaker_boot_messages), 316
SpinnakerBootOpCode (class in spinnman.messages.spinnaker_boot), 321
SpinnakerBootOpCode (class in spinnman.messages.spinnaker_boot.spinnaker_boot_opcode), 317
SpinnakerBootReceiver (class in spinnman.connections.abstract_classes), 229
SpinnakerBootReceiver (class in spinnman.connections.abstract_classes.spinnaker_boot_receiver), 224
SpinnakerBootSender (class in spinnman.connections.abstract_classes), 229
SpinnakerBootSender (class in spinnman.connections.abstract_classes.spinnaker_boot_sender), 224
SpinnakerLinkData (class in spinn_machine.link_data_objects), 18
SpinnakerLinkData (class in spinn_machine.link_data_objects.spinnaker_link_data), 17
SpinnakerRequestBuffers (class in spinnman.messages.eieio.command_messages), 256
SpinnakerRequestBuffers (class in spinnman.messages.eieio.command_messages.spinnaker_request_buffe 252
SpinnakerRequestSpiNNakerLinkVertex (class in spinnman.messages.eieio.command_messages), 256
SpinnakerRequestReadData (class in spinnman.messages.eieio.command_messages.spinnaker_request_read_ 253
SpiNNakerTriadGeometry (class in spinn_machine), 54
SpiNNakerTriadGeometry (class in spinn_machine.spinnaker_triad_geometry), 38
SpinnFrontEndException, 578
SpinnMachineAlreadyExistsException, 27
SpinnMachineException, 27
SpinnMachineInvalidParameterException, 27
spinnman (module), 217, 410
spinnman.connections (module), 245
spinnman.connections.abstract_classes (module), 225
spinnman.connections.abstract_classes.connection (module), 219
spinnman.connections.abstract_classes.eieio_receiver (module), 219
spinnman.connections.abstract_classes.eieio_sender (module), 220
spinnman.connections.abstract_classes.listenable (module), 220
spinnman.connections.abstract_classes.multicast_receiver (module), 220
spinnman.connections.abstract_classes.multicast_sender (module), 221
spinnman.connections.abstract_classes.scp_receiver (module), 222
spinnman.connections.abstract_classes.scp_sender (module), 222
spinnman.connections.abstract_classes.sdp_receiver

(module), 223
spinnman.connections.abstract_classes.sdp_sender (module), 223
spinnman.connections.abstract_classes.spinnaker_boot_receiver (module), 224
spinnman.connections.abstract_classes.spinnaker_boot_sender (module), 224
spinnman.connections.connection_listener (module), 243
spinnman.connections.scp_request_pipeline (module), 243
spinnman.connections.socket_address_with_chip (module), 244
spinnman.connections.token_bucket (module), 245
spinnman.connections.udp_packet_connections (module), 237
spinnman.connections.udp_packet_connections.bmp_connection (module), 230
spinnman.connections.udp_packet_connections.boot_connection (module), 231
spinnman.connections.udp_packet_connections.eieio_connection (module), 232
spinnman.connections.udp_packet_connections.ip_address_connection (module), 232
spinnman.connections.udp_packet_connections.scamp_connection (module), 233
spinnman.connections.udp_packet_connections.sdp_connection (module), 233
spinnman.connections.udp_packet_connections.udp_connection (module), 234
spinnman.connections.udp_packet_connections.udp_listenable_connection (module), 236
spinnman.connections.udp_packet_connections.utils (module), 236
spinnman.constants (module), 380
spinnman.exceptions (module), 381
spinnman.messages (module), 325
spinnman.messages.eieio (module), 267
spinnman.messages.eieio.abstract_messages (module), 248
spinnman.messages.eieio.abstract_messages.eieio_message (module), 248
spinnman.messages.eieio.command_messages (module), 253
spinnman.messages.eieio.command_messages.database_confirmation (module), 248
spinnman.messages.eieio.command_messages.eieio_command_header (module), 248
spinnman.messages.eieio.command_messages.eieio_command_message (module), 249
spinnman.messages.eieio.command_messages.event_stop_request (module), 250
spinnman.messages.eieio.command_messages.host_data_read (module), 250
spinnman.messages.eieio.command_messages.host_data_read_ack (module), 250
spinnman.messages.eieio.command_messages.host_send_sequenced_data (module), 251
spinnman.messages.eieio.command_messages.notification_protocol_pause_ (module), 251
spinnman.messages.eieio.command_messages.notification_protocol_start_r (module), 251
spinnman.messages.eieio.command_messages.padding_request (module), 251
spinnman.messages.eieio.command_messages.spinnaker_request_buffers (module), 252
spinnman.messages.eieio.command_messages.spinnaker_request_read_data (module), 252
spinnman.messages.eieio.command_messages.start_requests (module), 253
spinnman.messages.eieio.command_messages.stop_requests (module), 253
spinnman.messages.eieio.create_eieio_command (module), 266
spinnman.messages.eieio.create_eieio_data (module), 266
spinnman.messages.eieio.data_messages (module), 262
spinnman.messages.eieio.data_messages.abstract_data_element (module), 257
spinnman.messages.eieio.data_messages.eieio_data_header (module), 258
spinnman.messages.eieio.data_messages.eieio_data_message (module), 259
spinnman.messages.eieio.data_messages.key_data_element (module), 261
spinnman.messages.eieio.data_messages.key_payload_data_element (module), 261
spinnman.messages.eieio.eieio_prefix (module), 266
spinnman.messages.eieio.eieio_type (module), 266
spinnman.messages.multicast_message (module), 324
spinnman.messages.scp (module), 308
spinnman.messages.scp.abstract_messages (module), 271
spinnman.messages.scp.abstract_messages.bmp_request (module), 268
spinnman.messages.scp.abstract_messages.bmp_response (module), 269
spinnman.messages.scp.abstract_messages.scp_request (module), 269
spinnman.messages.scp.abstract_messages.scp_response (module), 270
spinnman.messages.scp.enums (module), 277
spinnman.messages.scp.enums.alloc_free (module), 274
spinnman.messages.scp.enums.bmp_info (module), 274
spinnman.messages.scp.enums.iptag_command (module), 274
spinnman.messages.scp.enums.led_action (module), 275
spinnman.messages.scp.enums.power_command (module), 275
spinnman.messages.scp.enums.scp_command (module),

275
spinnman.messages.scp.enums.scp_result (module), 276
spinnman.messages.scp.enums.signal (module), 277
spinnman.messages.scp.impl (module), 296
spinnman.messages.scp.impl.app_stop (module), 280
spinnman.messages.scp.impl.application_run (module), 281
spinnman.messages.scp.impl.bmp_get_version (module), 281
spinnman.messages.scp.impl.bmp_set_led (module), 281
spinnman.messages.scp.impl.check_ok_response (module), 282
spinnman.messages.scp.impl.count_state (module), 282
spinnman.messages.scp.impl.count_state_response (module), 282
spinnman.messages.scp.impl.fill_request (module), 283
spinnman.messages.scp.impl.fixed_route_init (module), 283
spinnman.messages.scp.impl.fixed_route_read (module), 284
spinnman.messages.scp.impl.flood_fill_data (module), 284
spinnman.messages.scp.impl.flood_fill_end (module), 285
spinnman.messages.scp.impl.flood_fill_start (module), 285
spinnman.messages.scp.impl.get_chip_info (module), 285
spinnman.messages.scp.impl.get_chip_info_response (module), 286
spinnman.messages.scp.impl.get_version (module), 286
spinnman.messages.scp.impl.get_version_response (module), 287
spinnman.messages.scp.impl.iptag_clear (module), 287
spinnman.messages.scp.impl.iptag_get (module), 287
spinnman.messages.scp.impl.iptag_get_info (module), 288
spinnman.messages.scp.impl.iptag_get_info_response (module), 288
spinnman.messages.scp.impl.iptag_set (module), 288
spinnman.messages.scp.impl.iptag_set_tto (module), 289
spinnman.messages.scp.impl.read_adc (module), 289
spinnman.messages.scp.impl.read_fpga_register (module), 289
spinnman.messages.scp.impl.read_link (module), 290
spinnman.messages.scp.impl.read_memory (module), 290
spinnman.messages.scp.impl.reverse_iptag_set (module), 291
spinnman.messages.scp.impl.router_alloc (module), 291
spinnman.messages.scp.impl.router_clear (module), 291
spinnman.messages.scp.impl.router_init (module), 292
spinnman.messages.scp.impl.sdram_alloc (module), 292
spinnman.messages.scp.impl.sdram_de_alloc (module), 293
spinnman.messages.scp.impl.send_signal (module), 293
spinnman.messages.scp.impl.set_led (module), 294
spinnman.messages.scp.impl.set_power (module), 294
spinnman.messages.scp.impl.write_fpga_register (module), 294
spinnman.messages.scp.impl.write_link (module), 295
spinnman.messages.scp.impl.write_memory (module), 295
spinnman.messages.scp.scp_request_header (module), 307
spinnman.messages.scp.scp_response_header (module), 308
spinnman.messages.sdp (module), 312
spinnman.messages.sdp.sdp_flag (module), 309
spinnman.messages.sdp.sdp_header (module), 310
spinnman.messages.sdp.sdp_message (module), 312
spinnman.messages.spinnaker_boot (module), 320
spinnman.messages.spinnaker_boot.spinnaker_boot_message (module), 315
spinnman.messages.spinnaker_boot.spinnaker_boot_messages (module), 316
spinnman.messages.spinnaker_boot.spinnaker_boot_op_code (module), 317
spinnman.messages.spinnaker_boot.system_variable_boot_values (module), 317
spinnman.model (module), 347
spinnman.model.adc_info (module), 331
spinnman.model.bmp_connection_data (module), 332
spinnman.model.chip_info (module), 333
spinnman.model.chip_summary_info (module), 334
spinnman.model.cpu_info (module), 336
spinnman.model.cpu_infos (module), 338
spinnman.model.diagnostic_filter (module), 339
spinnman.model.enums (module), 328
spinnman.model.enums.cpu_state (module), 325
spinnman.model.enums.diagnostic_filter_default_routing_status (module), 325
spinnman.model.enums.diagnostic_filter_destination (module), 326
spinnman.model.enums.diagnostic_filter_emergency_routing_status (module), 326
spinnman.model.enums.diagnostic_filter_packet_type (module), 326
spinnman.model.enums.diagnostic_filter_payload_status (module), 327
spinnman.model.enums.diagnostic_filter_source (module), 327
spinnman.model.enums.mailbox_command (module), 327
spinnman.model.enums.p2p_table_route (module), 327
spinnman.model.enums.run_time_error (module), 328
spinnman.model.executable_targets (module), 340
spinnman.model.heap_element (module), 341

spinnman.model.io_buffer (module), 341
spinnman.model.machine_dimensions (module), 342
spinnman.model.p2p_table (module), 342
spinnman.model.router_diagnostics (module), 343
spinnman.model.version_info (module), 346
spinnman.processes (module), 368
spinnman.processes.abstract_multi_connection_process (module), 361
spinnman.processes.abstract_multi_connection_process_connection_selector (module), 362
spinnman.processes.abstract_process (module), 362
spinnman.processes.abstract_single_connection_process (module), 362
spinnman.processes.application_run_process (module), 362
spinnman.processes.de_alloc_sdram_process (module), 362
spinnman.processes.fill_process (module), 363
spinnman.processes.get_cpu_info_process (module), 363
spinnman.processes.get_heap_process (module), 363
spinnman.processes.get_machine_process (module), 363
spinnman.processes.get_routes_process (module), 364
spinnman.processes.get_tags_process (module), 364
spinnman.processes.get_version_process (module), 364
spinnman.processes.load_fixed_route_routing_entry_process (module), 364
spinnman.processes.load_routes_process (module), 365
spinnman.processes.malloc_sdram_process (module), 365
spinnman.processes.most_direct_connection_selector (module), 365
spinnman.processes.read_fixed_route_routing_entry_process (module), 365
spinnman.processes.read_iobuf_process (module), 366
spinnman.processes.read_memory_process (module), 366
spinnman.processes.read_router_diagnostics_process (module), 366
spinnman.processes.round_robin_connection_selector (module), 367
spinnman.processes.send_single_command_process (module), 367
spinnman.processes.write_memory_flood_process (module), 367
spinnman.processes.write_memory_process (module), 367
spinnman.transceiver (module), 383
spinnman.utilities (module), 380
spinnman.utilities.appid_tracker (module), 378
spinnman.utilities.io (module), 375
spinnman.utilities.io.abstract_io (module), 372
spinnman.utilities.io.file_io (module), 373
spinnman.utilities.io.memory_io (module), 374
spinnman.utilities.locate_connected_machine_ip_address (module), 379
spinnman.utilities.reports (module), 379
spinnman.utilities.utility_functions (module), 379
SpinnmanEIEIOPacketParsingException, 381
SpinnmanException, 381
SpinnmanGenericProcessException, 381
SpinnmanInvalidPacketException, 381
SpinnmanInvalidParameterException, 381
SpinnmanInvalidParameterTypeException, 382
SpinnmanIOException, 381
SpinnmanTimeoutException, 382
SpinnmanUnexpectedResponseCodeException, 382
SpinnmanUnsupportedOperationException, 383
SPOMNIBOT (spynnaker.pyNN.protocols.munich_io_spinnaker_link_proto attribute), 818
SPOMNIBOT (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtoc attribute), 823
spynnaker (module), 618, 838
spynnaker.gsyn_tools (module), 837
spynnaker.plot_utils (module), 838
spynnaker.pyNN (module), 837
spynnaker.pyNN.abstract_spinnaker_common (module), 833
spynnaker.pyNN.connections (module), 620
spynnaker.pyNN.connections.ethernet_command_connection (module), 618
spynnaker.pyNN.connections.ethernet_control_connection (module), 619
spynnaker.pyNN.connections.spynnaker_live_spikes_connection (module), 619
spynnaker.pyNN.exceptions (module), 834
spynnaker.pyNN.external_devices_models (module), 658
spynnaker.pyNN.external_devices_models.abstract_ethernet_controller (module), 650
spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor (module), 650
spynnaker.pyNN.external_devices_models.abstract_ethernet_translator (module), 651
spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_ (module), 651
spynnaker.pyNN.external_devices_models.arbitrary_fpga_device (module), 652
spynnaker.pyNN.external_devices_models.external_device_lif_control (module), 652
spynnaker.pyNN.external_devices_models.external_spinnaker_link_cochlea (module), 653
spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_re (module), 654
spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_ (module), 655
spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_ (module), 657
spynnaker.pyNN.external_devices_models.push_bot (module), 649

spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device (module), 649
spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device (module), 649
spynnaker.pyNN.external_devices_models.push_bot.push_bot_commands (module), 624
spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_protocol (module), 622
spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_protocol_packets (module), 623
spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet (module), 632
spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_command_packet (module), 625
spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_laser_packet (module), 626
spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_led_packet (module), 627
spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_packet (module), 628
spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_retina_packet (module), 629
spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_speaker_packet (module), 629
spynnaker.pyNN.external_devices_models.push_bot.push_bot_exceptions (module), 630
spynnaker.pyNN.external_devices_models.push_bot.push_bot_translator (module), 631
spynnaker.pyNN.external_devices_models.push_bot.push_bot_utilities (module), 631
spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters (module), 640
spynnaker.pyNN.external_devices_models.push_bot.push_bot_laser (module), 639
spynnaker.pyNN.external_devices_models.push_bot.push_bot_led (module), 639
spynnaker.pyNN.external_devices_models.push_bot.push_bot_motor (module), 639
spynnaker.pyNN.external_devices_models.push_bot.push_bot_retina_resolution (module), 639
spynnaker.pyNN.external_devices_models.push_bot.push_bot_retina_vispike (module), 640
spynnaker.pyNN.external_devices_models.push_bot.push_bot_speaker (module), 640
spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link (module), 646
spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_protocol (module), 642
spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_utilities (module), 643
spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_vertex (module), 644
spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_vertex_fixed (module), 644
spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_li (module), 645
spynnaker.pyNN.external_devices_models.threshold_type_multicast_device (module), 658
spynnaker.pyNN.model_binaries (module), 665
spynnaker.pyNN.models (module), 817
spynnaker.pyNN.models.abstract_models (module), 667
spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synap
spynnaker.pyNN.models.abstract_models.abstract_push_bot_lif_spinnaker_link (module), 665
spynnaker.pyNN.models.abstract_models.abstract_contains_units (module), 666
spynnaker.pyNN.models.abstract_models.abstract_filterable_edge (module), 666
spynnaker.pyNN.models.abstract_models.abstract_ethernet_device
spynnaker.pyNN.models.abstract_models.abstract_population_initializable (module), 666
spynnaker.pyNN.models.abstract_models.abstract_ethernet_laser_device
spynnaker.pyNN.models.abstract_models.abstract_population_settable (module), 667
spynnaker.pyNN.models.abstract_models.abstract_ethernet_led_device
spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_ (module), 667
spynnaker.pyNN.models.abstract_models.abstract_ethernet_motor_device
spynnaker.pyNN.models.abstract_models.abstract_weight_updatable (module), 667
spynnaker.pyNN.models.abstract_models.abstract_ethernet_retina_device
spynnaker.pyNN.models.common (module), 673
spynnaker.pyNN.models.common.abstract_neuron_recordable (module), 669
spynnaker.pyNN.models.common.abstract_spike_recordable (module), 670
spynnaker.pyNN.models.common.abstract_uint32_recorder (module), 670
spynnaker.pyNN.models.common.abstract_spike_recorder (module), 671
spynnaker.pyNN.models.common.gsyn_inhibitory_recorder (module), 671
spynnaker.pyNN.models.common.multi_spike_recorder (module), 672
spynnaker.pyNN.models.common.neuron_recorder (module), 672
spynnaker.pyNN.models.common.recording_utils (module), 672
spynnaker.pyNN.models.common.population_settable (module), 673
spynnaker.pyNN.models.common.vispike_recorder (module), 673
spynnaker.pyNN.models.neural_projections (module), 699
spynnaker.pyNN.models.neural_projections.connectors (module), 686
spynnaker.pyNN.models.neural_projections.abstract_connector (module), 677
spynnaker.pyNN.models.neural_projections.all_to_all_connector (module), 678
spynnaker.pyNN.models.neural_projections.distance_dependent (module), 679
spynnaker.pyNN.models.neural_projections.fixed_number_post_ (module), 680

spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector (module), 681
spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector (module), 682
spynnaker.pyNN.models.neural_projections.connectors.from_file_connector (module), 683
spynnaker.pyNN.models.neural_projections.connectors.from_list_connector (module), 683
spynnaker.pyNN.models.neural_projections.connectors.multapse_connector (module), 684
spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector (module), 685
spynnaker.pyNN.models.neural_projections.connectors.small_world_connector (module), 686
spynnaker.pyNN.models.neural_projections.delay_afferent_application_edge (module), 696
spynnaker.pyNN.models.neural_projections.delay_afferent_machine_edge (module), 696
spynnaker.pyNN.models.neural_projections.delayed_application_edge (module), 696
spynnaker.pyNN.models.neural_projections.delayed_machine_edge (module), 697
spynnaker.pyNN.models.neural_projections.projection_application_edge (module), 697
spynnaker.pyNN.models.neural_projections.projection_machine_edge (module), 698
spynnaker.pyNN.models.neural_projections.synapse_information (module), 698
spynnaker.pyNN.models.neural_properties (module), 701
spynnaker.pyNN.models.neural_properties.neural_parameters (module), 701
spynnaker.pyNN.models.neuron (module), 786
spynnaker.pyNN.models.neuron.abstract_population_vertex (module), 778
spynnaker.pyNN.models.neuron.additional_inputs (module), 703
spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input (module), 702
spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive (module), 703
spynnaker.pyNN.models.neuron.builds (module), 718
spynnaker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_ista (module), 705
spynnaker.pyNN.models.neuron.builds.hh_cond_exp (module), 706
spynnaker.pyNN.models.neuron.builds.if_cond_alpha (module), 707
spynnaker.pyNN.models.neuron.builds.if_cond_exp_base (module), 708
spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc (module), 709
spynnaker.pyNN.models.neuron.builds.if_curr_alpha (module), 710
spynnaker.pyNN.models.neuron.builds.if_curr_delta (module), 711
spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base (module), 712
spynnaker.pyNN.models.neuron.builds.if_curr_exp_base (module), 713
spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive (module), 714
spynnaker.pyNN.models.neuron.builds.if_facets_hardware1 (module), 715
spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base (module), 716
spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base (module), 717
spynnaker.pyNN.models.neuron.connection_holder (module), 783
spynnaker.pyNN.models.neuron.input_types (module), 726
spynnaker.pyNN.models.neuron.input_types.abstract_input_type (module), 724
spynnaker.pyNN.models.neuron.input_types.input_type_conductance (module), 725
spynnaker.pyNN.models.neuron.input_types.input_type_current (module), 725
spynnaker.pyNN.models.neuron.master_pop_table_generators (module), 731
spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_mas (module), 727
spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_ (module), 729
spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_ (module), 730
spynnaker.pyNN.models.neuron.neuron_models (module), 738
spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model (module), 733
spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh (module), 735
spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integ (module), 736
spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integ (module), 737
spynnaker.pyNN.models.neuron.plasticity (module), 755
spynnaker.pyNN.models.neuron.plasticity.stdp (module), 755
spynnaker.pyNN.models.neuron.plasticity.stdp.common (module), 743
spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers (module), 743
spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure (module), 744
spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_s (module), 743
spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_s (module), 744

spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure (module), 744
spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence (module), 748
spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence (module), 745
spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_triplet (module), 745
spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.recent (module), 746
spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.vogels_2011 (module), 747
spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.vogels_2011_additive (module), 747
spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.vogels_2011_multiplicative (module), 748
spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence (module), 753
spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence (module), 751
spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.additive_two_term (module), 751
spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative (module), 752
spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet (module), 752
spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multiplicative (module), 753
spynnaker.pyNN.models.neuron.population_machine_vertex (module), 784
spynnaker.pyNN.models.neuron.synapse_dynamics (module), 760
spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics (module), 755
spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics (module), 756
spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics (module), 756
spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_synapse_dynamics (module), 758
spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static (module), 758
spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp (module), 759
spynnaker.pyNN.models.neuron.synapse_io (module), 766
spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io (module), 765
spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based (module), 765
spynnaker.pyNN.models.neuron.synapse_types (module), 770
spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type (module), 767
spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta (module), 768
spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential (module), 769
spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential (module), 769
spynnaker.pyNN.models.neuron.synapse_types.synapse_type_triplet (module), 785
spynnaker.pyNN.models.neuron.threshold_types (module), 775
spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type (module), 773
spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic (module), 774
spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static (module), 775
spynnaker.pyNN.models.pynn_population_common (module), 814
spynnaker.pyNN.models.pynn_projection_common (module), 816
spynnaker.pyNN.models.pynn_weight_dependence_common (module), 816
spynnaker.pyNN.models.spike_additive (module), 800
spynnaker.pyNN.models.spike_source.spike_source_array (module), 793
spynnaker.pyNN.models.spike_source.spike_source_from_file (module), 795
spynnaker.pyNN.models.spike_source.spike_source_poisson (module), 795
spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex (module), 799
spynnaker.pyNN.models.utility_models (module), 810
spynnaker.pyNN.models.utility_models.delay_block (module), 806
spynnaker.pyNN.models.utility_models.delay_extension_machine_vertex (module), 806
spynnaker.pyNN.models.utility_models.delay_extension_vertex (module), 807
spynnaker.pyNN.models.utility_models.spike_injector (module), 809
spynnaker.pyNN.overridden_pacman_functions (module), 817
spynnaker.pyNN.overridden_pacman_functions.graph_edge_filter (module), 817
spynnaker.pyNN.overridden_pacman_functions.graph_edge_weight_update (module), 817
spynnaker.pyNN.overridden_pacman_functions.spynnaker_data_specification (module), 817
spynnaker.pyNN.protocols (module), 822
spynnaker.pyNN.protocols.munich_io_ethernet_protocol (module), 817
spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol (module), 818
spynnaker.pyNN.spynnaker_external_device_plugin_manager

(module), 835
spynnaker.pyNN.spynnaker_simulator_interface (module), 837
spynnaker.pyNN.utilities (module), 833
spynnaker.pyNN.utilities.constants (module), 828
spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider (module), 829
spynnaker.pyNN.utilities.random_stats (module), 828
spynnaker.pyNN.utilities.random_stats.abstract_random_stats (module), 827
spynnaker.pyNN.utilities.running_stats (module), 829
spynnaker.pyNN.utilities.spynnaker_connection_holder_generator (module), 829
spynnaker.pyNN.utilities.spynnaker_failed_state (module), 829
spynnaker.pyNN.utilities.spynnaker_neuron_network_specification_report (module), 830
spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report (module), 830
spynnaker.pyNN.utilities.utility_calls (module), 830
spynnaker.spike_checker (module), 838
SpYNNakerConnectionHolderGenerator (class in spynnaker.pyNN.utilities.spynnaker_connection_holder_generator), 829
SpynnakerDataSpecificationWriter (class in spynnaker.pyNN.overridden_pacman_functions.spynnaker_data_specification_writer), 817
SpynnakerException, 834
SpynnakerExternalDevicePluginManager (class in spynnaker.pyNN.spynnaker_external_device_plugin_manager), 835
SpynnakerFailedState (class in spynnaker.pyNN.utilities.spynnaker_failed_state), 829
SpynnakerLiveSpikesConnection (class in spynnaker.pyNN.connections), 620
SpynnakerLiveSpikesConnection (class in spynnaker.pyNN.connections.spynnaker_live_spikes_connection), 619
SpYNNakerNeuronGraphNetworkSpecificationReport (class in spynnaker.pyNN.utilities.spynnaker_neuron_network_specification_report), 830
SpynnakerSimulatorInterface (class in spynnaker.pyNN.spynnaker_simulator_interface), 837
SpYNNakerSynapticMatrixReport (class in spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report), 830
src1_reg (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
src1_reg (data_specification.DataSpecificationExecutorFunctions attribute), 462
src2_reg (data_specification.data_specification_executor_functions.DataSpecificationExecutorFunctions attribute), 424
src2_reg (data_specification.DataSpecificationExecutorFunctions attribute), 462
stack_pointer (spinnman.model.cpu_info.CPUInfo attribute), 337
stack_pointer (spinnman.model.CPUInfo attribute), 353
standard_deviation (spynnaker.pyNN.utilities.running_stats.RunningStats attribute), 829
START (spinnman.messages.scp.enums.Signal attribute), 280
START (spinnman.messages.scp.enums.signal.Signal attribute), 277
start (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSo attribute), 798
start (spynnaker.pyNN.models.spike_source.SpikeSourcePoisson attribute), 805
start_address (pacman.model.resources.element_free_space.ElementFreeSp attribute), 153
start_address (pacman.model.resources.ElementFreeSpace attribute), 160
start_address (spinn_front_end_common.interface.buffer_management.stora attribute), 516
start_address (spinn_front_end_common.interface.buffer_management.stora attribute), 523
start_address() (spinnman.messages.eieio.command_messages.spinnaker_re method), 253
start_address() (spinnman.messages.eieio.command_messages.SpinnakerRe method), 257
start_conditional() (data_specification.data_specification_generator.DataSpe method), 443
start_conditional() (data_specification.DataSpecificationGenerator method), 481
START_CONSTRUCTOR (data_specification.enums.Commands attribute), 418
START_CONSTRUCTOR (data_specification.enums.commands.Commands attribute), 415
start_function() (data_specification.data_specification_generator.DataSpecif method), 443
start_function() (data_specification.DataSpecificationGenerator method), 481
start_loop() (data_specification.data_specification_generator.DataSpecificat method), 444
start_loop() (data_specification.DataSpecificationGenerator method), 482
start_resume_commands (spinn_front_end_common.abstract_models.abstra attribute), 498
start_resume_commands (spinn_front_end_common.abstract_models.Abstr attribute), 501
start_resume_commands (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fp attribute), 654

start_resume_commands (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
start_resume_commands (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
start_resume_commands (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 665
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_model.AbstractPushBotRetinaDevice attribute), 649
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotRetinaDevice attribute), 650
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetLaserDevice attribute), 627
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetLaserDevice attribute), 628
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice attribute), 628
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice attribute), 630
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetLaserDevice attribute), 634
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetLaserDevice attribute), 634
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice attribute), 635
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice attribute), 636
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_retina_device.PushBotSpinnaker attribute), 644
start_resume_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_retina_device.PushBotSpinnaker attribute), 648
START_RESUME_NOTIFICATION (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
START_SENDING_REQUESTS (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
START_STRUCT (data_specification.enums.Commands attribute), 418
START_STRUCT (data_specification.enums.commands.Commands attribute), 415
START_TIME (spinn_front_end_common.interface.profiling.profile_data.P attribute), 541
start_timing() (spinn_utilities.timer.Timer method), 13
Start_Requests (class in spinnman.messages.eieio.command_messages), 257
Start_Requests (class in spinnman.messages.eieio.command_messages.start_requests), 253
state (spinnman.model.CPUInfo attribute), 353
static_get_binary_file_name() (spinn_front_end_common.utility_models.extra_monitor_support static method), 593
static_get_binary_start_type() (spinn_front_end_common.utility_models.ExtraMonitorSupport static method), 593
static_resources_required() (spinn_front_end_common.utility_models.ExtraMonitorSupport static method), 593
status_map (spinnman.messages.spinnaker_boot.system_variable_boot_valu attribute), 319
status_map (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
std() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
std() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
STOP (spinnman.messages.scp.enums.Signal attribute), 280
STOP (spinnman.messages.scp.enums.Signal attribute), 277
stop() (spinn_front_end_common.interface.abstract_spinnaker_base.Abstrac method), 549
stop() (spinn_front_end_common.interface.buffer_management.buffer_man method), 525
stop() (spinn_front_end_common.interface.buffer_management.BufferMana method), 530
stop() (spinn_front_end_common.interface.interface_functions.notification_ method), 534
stop() (spinn_front_end_common.interface.interface_functions.Notification method), 539
stop() (spinn_front_end_common.utilities.FailedState method), 579
stop() (spinn_front_end_common.utilities.FailedState method), 583
stop() (spinn_front_end_common.utilities.simulator_interface.SimulatorInte method), 583
stop() (spinn_front_end_common.utilities.SimulatorInterface method), 584
stop() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerC method), 834
stop_application() (spinnman.transceiver.Transceiver

method), 405
STOP_PAUSE_NOTIFICATION (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
STOP_SENDING_REQUESTS (spinnman.constants.EIEIO_COMMAND_IDS attribute), 380
StopRequests (class in spinnman.messages.eieio.command_messages), 257
StopRequests (class in spinnman.messages.eieio.command_messages.stop_requests), 253
store_data_in_region_buffer() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 511
store_data_in_region_buffer() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 519
store_end_buffering_sequence_number() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 511
store_end_buffering_sequence_number() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 519
store_end_buffering_state() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 512
store_end_buffering_state() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 519
store_last_received_packet_from_core() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 512
store_last_received_packet_from_core() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 520
store_last_sent_packet_to_core() (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_receiving_data.BufferedReceivingData method), 512
store_last_sent_packet_to_core() (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedReceivingData method), 520
strip_sdp (pacman.model.resources.iptag_resource.IPtagResource attribute), 154
strip_sdp (pacman.model.resources.IPtagResource attribute), 161
strip_sdp (pacman.model.resources.specific_board_iptag_resource.SpecificBoardIPtagResource attribute), 157
strip_sdp (spinn_front_end_common.utilities.utility_objs.live_packet_gather_parameters.LivePacketGatherParameters attribute), 573
strip_sdp (spinn_front_end_common.utilities.utility_objs.LivePacketGatherParameters attribute), 577
strip_sdp (spinn_machine.tags.IPTag attribute), 20
strip_sdp (spinn_machine.tags.iptag.IPTag attribute), 19
STRUCT_ELEM (data_specification.enums.Commands attribute), 418
STRUCT_ELEM (data_specification.enums.commands.Commands attribute), 415
struct_slot (data_specification.data_specification_generator.DataSpecificatio attribute), 445
struct_slot (data_specification.DataSpecificationGenerator attribute), 483
struct_slots (data_specification.data_specification_executor_functions.DataS attribute), 424
struct_slots (data_specification.DataSpecificationExecutorFunctions attribute), 462
struct_type (spinnman.processes.fill_process.FillDataType attribute), 363
struct_type (spinnman.processes.FillDataType attribute), 369
structure (spyn_objects.BufferedReceivingData attribute), 815
SUBTRACT (data_specification.enums.arithemetic_operation.ArithmeticOp attribute), 414
SUBTRACT (data_specification.enums.ArithmeticOperation attribute), 417
SUPPORTED_TAGS (class in pacman.utilities.utility_objs.flexi_field), 199
supports_injection() (in module pacman.executors.injection_decorator), 95
supports_sends_message() (spinnman.connections.udp_packet_connections.ip_address_connection method), 232
supports_sends_message() (spinnman.connections.udp_packet_connections.IPAddressesConnectio method), 241
SURPLUS_DATA_ENTRIES (spinn_front_end_common.mapping_algorithms.on_chip_router_ attribute), 550
SVC (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
SVC (spinnman.model.enums.RunTimeError attribute), 330
SWERR (spinnman.model.enums.run_time_error.RunTimeError attribute), 328
SWERR (spinnman.model.enums.RunTimeError attribute), 330
SWITCH_FOCUS (data_specification.enums.Commands attribute), 418
SWITCH_FOCUS (data_specification.enums.commands.Commands attribute), 415
switch_write_focus() (data_specification.data_specification_generator.DataS method), 445
switch_write_focus() (data_specification.DataSpecificationGenerator method), 483
synapse_dynamics (spynnaker.pyNN.models.neural_projections.synapse_information.Syna

attribute), 698
synapse_dynamics (spynnaker.pyNN.models.neural_projections.SynapseInformation attribute), 701
synapse_dynamics (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 783
synapse_dynamics (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 790
synapse_dynamics (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager attribute), 785
synapse_dynamics (spynnaker.pyNN.models.neuron.SynapticManager attribute), 792
SYNAPSE_DYNAMICS (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 828
synapse_information (spynnaker.pyNN.models.neural_projections.projection_application_edge.ProjectionApplicationEdge attribute), 698
synapse_information (spynnaker.pyNN.models.neural_projections.projection_machine_edge.ProjectionMachineEdge attribute), 698
synapse_information (spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge attribute), 700
synapse_information (spynnaker.pyNN.models.neural_projections.ProjectionMachineEdge attribute), 701
SYNAPSE_PARAMS (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 828
synapse_type (spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses attribute), 666
synapse_type (spynnaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses attribute), 668
synapse_type (spynnaker.pyNN.models.neural_projections.synapse_information.SynapseInformation attribute), 698
synapse_type (spynnaker.pyNN.models.neural_projections.SynapseInformation attribute), 701
synapse_type (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 783
synapse_type (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 790
synapse_type (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager attribute), 785
synapse_type (spynnaker.pyNN.models.neuron.SynapticManager attribute), 792
SynapseDynamicsStatic (class in spynnaker.pyNN.models.neuron.synapse_dynamics), 762
SynapseDynamicsStatic (class in spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics), 758
SynapseDynamicsSTDP (class in spynnaker.pyNN.models.neuron.synapse_dynamics), 763
SynapseDynamicsSTDPVertex (class in spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics), 759
SynapseInformation (class in spynnaker.pyNN.models.neural_projections), 701
SynapseInformation (class in spynnaker.pyNN.models.neural_projections.synapse_information), 698
SynapseIORowBased (class in spynnaker.pyNN.models.neuron.synapse_io), 766
SynapseIORowBased (class in spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based), 765
SynapseRowTwoBitExceptionApplicationEdge 818
SynapseStructureWeightAccumulator (class in spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure), 744
SynapseStructureWeightAccumulator (class in spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure), 745
SynapseStructureWeightOnly (class in spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure), 744
SynapseStructureWeightOnly (class in spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_structure), 745
SynapseTypeDelta (class in spynnaker.pyNN.models.neuron.synapse_types), 773
SynapseTypeDelta (class in spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta), 768
SynapseTypeDualExponential (class in spynnaker.pyNN.models.neuron.synapse_types), 771
SynapseTypeDualExponential (class in spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential), 769
SynapseTypeExponential (class in spynnaker.pyNN.models.neuron.synapse_types), 772
SynapseTypeExponential (class in spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential), 769
SYNAPTIC_MATRIX (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 828

synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence attribute), 745
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence attribute), 748
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.pfister_spike_triplet.TimingDependencePfisterSpikeTriplet attribute), 746
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent.TimingDependenceRecurrent attribute), 746
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair.TimingDependenceSpikeNearestPair attribute), 747
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair.TimingDependenceSpikePair attribute), 747
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.vogels_2011.TimingDependenceVogels2011 attribute), 748
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence attribute), 749
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AdditiveTwoPow attribute), 750
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.MultiplicativeTwoPow attribute), 749
synaptic_structure (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_vogels_2011.WeightDependenceVogels2011 attribute), 751
SynapticBlockGenerationException, 834
SynapticBlockReadException, 835
SynapticConfigurationException, 835
SynapticManager (class in spynnaker.pyNN.models.neuron), 791
SynapticManager (class in spynnaker.pyNN.models.neuron.synaptic_manager), 785
SynapticMaxIncomingAtomsSupportException, 835
SYNC (spinn_front_end_common.utilities.utility_objs.executable_type.ExecutableType attribute), 572
SYNC (spinn_front_end_common.utilities.utility_objs.ExecutableType attribute), 576
SYNC0 (spinnman.messages.scp.enums.Signal attribute), 280
SYNC0 (spinnman.messages.scp.enums.signal.Signal attribute), 277
SYNC0 (spinnman.model.enums.cpu_state.CPUState attribute), 325
SYNC0 (spinnman.model.enums.CPUState attribute), 329
SYNC1 (spinnman.messages.scp.enums.Signal attribute), 280
SYNC1 (spinnman.messages.scp.enums.signal.Signal attribute), 277
SYNC1 (spinnman.model.enums.cpu_state.CPUState attribute), 325
SYNC1 (spinnman.model.enums.CPUState attribute), 329
synfire_multiple_lines_spike_checker() (in module spynnaker.spike_checker), 838
synfire_spike_checker() (in module spynnaker.spike_checker), 838
SYSTEM (spinn_front_end_common.utilities.utility_objs.executable_type.ExecutableType attribute), 572
SYSTEM (spinn_front_end_common.utilities.utility_objs.ExecutableType attribute), 576
SYSTEM (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 586
SYSTEM (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 588
system_buffer_words (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBootValues attribute), 319
system_buffer_words (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
system_ram_base_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBootValues attribute), 319
system_ram_base_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 323
system_ram_heap_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBootValues attribute), 319
system_ram_heap_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
system_ram_heap_address (spinnman.model.chip_info.ChipInfo attribute), 334
system_ram_heap_address (spinnman.model.ChipInfo attribute), 341
SYSTEM_REGION (spinn_front_end_common.utility_models.command_sender_machine_vertex.CommandSenderMachineVertex attribute), 589
SYSTEM_REGION (spinn_front_end_common.utility_models.CommandSender attribute), 605
SYSTEM_REGION (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_model.MunichSpinnakerLinkVertex attribute), 655
SYSTEM_REGION (spyn-

naker.pyNN.external_devices_models.MunichMotorDevice attribute), 663
SYSTEM_REGION (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex.POISSON_SPI attribute), 799
SYSTEM_REGION (spynnaker.pyNN.models.spike_source.SpikeSourcePoissonMachineVertex.POISSON_SPIKE_SOURCE_REGIONS attribute), 805
system_sdram_base_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 319
system_sdram_base_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
system_sdram_base_address (spinnman.model.chip_info.ChipInfo attribute), 334
system_sdram_base_address (spinnman.model.ChipInfo attribute), 349
system_sdram_bytes (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 319
system_sdram_bytes (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
system_sdram_heap_address (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 319
system_sdram_heap_address (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
SystemVariableBootValues (class in spinnman.messages.spinnaker_boot.system_variable_boot_values), 317
SystemVariableDefinition (class in spinnman.messages.spinnaker_boot), 321
SystemVariableDefinition (class in spinnman.messages.spinnaker_boot.system_variable_boot_values), 317

T

tag (pacman.model.resources.iptag_resource.IPtagResource attribute), 154
tag (pacman.model.resources.IPtagResource attribute), 161
tag (pacman.model.resources.reverse_iptag_resource.ReverseIPtagResource attribute), 156
tag (pacman.model.resources.ReverseIPtagResource attribute), 162
tag (pacman.model.resources.specific_board_iptag_resource.SpecificBoardIPtagResource attribute), 158
tag (pacman.model.resources.specific_board_reverse_iptag_resource.SpecificBoardReverseIPtagResource attribute), 158
tag (pacman.utilities.utility_objs.Field attribute), 203
tag (pacman.utilities.utility_objs.field.Field attribute), 199
tag (pacman.utilities.utility_objs.flexi_field.FlexiField attribute), 199
tag (pacman.utilities.utility_objs.FlexiField attribute), 204
tag (spinn_front_end_common.utilities.utility_objs.live_packet_gather_para attribute), 573
tag (spinn_front_end_common.utilities.utility_objs.LivePacketGatherParam attribute), 577
tag (spinn_machine.tags.abstract_tag.AbstractTag attribute), 18
tag (spinn_machine.tags.AbstractTag attribute), 20
tag (spinnman.messages.eieio.data_messages.eieio_data_header.EIEIOData attribute), 259
tag (spinnman.messages.eieio.data_messages.EIEIODataHeader attribute), 263
tag (spinnman.messages.sdp.sdp_header.SDPHeader attribute), 311
tag (spinnman.messages.sdp.SDPHeader attribute), 314
tag (spinnman.model.heap_element.HeapElement attribute), 341
tag (spinnman.model.HeapElement attribute), 356
tag_allocator_report() (in module pacman.operations.algorithm_reports.reports), 184
tag_ids (spinn_machine.chip attribute), 43
tag_ids (spinn_machine.chip.Chip attribute), 24
Tags (class in pacman.model.tags), 179
Tags (class in pacman.model.tags.tags), 178
tags (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractS attribute), 549
tags (spinn_front_end_common.interface.profiling.profile_data.ProfileData attribute), 542
tags (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
tags (spinn_front_end_common.utilities.FailedState attribute), 583
tags (spinn_front_end_common.utilities.simulator_interface.SimulatorInterf attribute), 583
tags (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
TagsLoader (class in spinn_front_end_common.interface.interface_function 540
TagsLoader (class in spinn_front_end_common.interface.interface_function 536
take_sample() (spinn_utilities.timer.Timer method), 13
tau (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timi attribute), 748
tau (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tim attribute), 751
tau_ca2 (spynnaker.pyNN.models.neuron.additional_inputs.additional_inpu attribute), 703
tau_ca2 (spynnaker.pyNN.models.neuron.additional_inputs.AdditionalInput

attribute), 704
tau_m (spynnaker.pyNN.models.neuron.neuron_models.neuron_models_leaky_integrate_v.NeuronModelLeakyIntegrateThresholdTypeMa
attribute), 737
tau_m (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate
attribute), 742
tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair
attribute), 746
tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair
attribute), 747
tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeTriplet
attribute), 747
tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair
attribute), 749
tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair
attribute), 750
tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeTriplet
attribute), 749
tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair
attribute), 746
tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair
attribute), 747
tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeTriplet
attribute), 747
tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair
attribute), 749
tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair
attribute), 750
tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeTriplet
attribute), 749
tau_refrac (spynnaker.pyNN.models.neuron.neuron_models.neuron_models_leaky_integrate_v.NeuronModelLeakyIntegrateAndF
attribute), 738
tau_refrac (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire
attribute), 743
tau_syn_E (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential
attribute), 769
tau_syn_E (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential
attribute), 770
tau_syn_E (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential
attribute), 772
tau_syn_E (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential
attribute), 773
tau_syn_E2 (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential
attribute), 769
tau_syn_E2 (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential
attribute), 772
tau_syn_I (spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential
attribute), 769
tau_syn_I (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential
attribute), 770
tau_syn_I (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential
attribute), 772
tau_syn_I (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential
attribute), 773
tau_th (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_stochastic.ThresholdTypeStochastic
attribute), 775
tau_th (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass
attribute), 777
tau_x (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeTriplet
attribute), 746
tau_x (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair
attribute), 749
tau_x (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair
attribute), 746
tau_y (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair
attribute), 749
tau_y (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeTriplet
attribute), 749
TDMAAgendaBuilder (class in spinn_front_end_common.interface.interface_functions), 540
TDMAAgendaBuilder (class in spinn_front_end_common.interface.interface_functions.tdma_age
537
tell() (spinnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamicsStructurePlasticityAbstractTimingDependence
method), 64
tell() (spinnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamicsStructurePlasticityAbstractTimingDependenc
method), 64
tell() (spinnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamicsStructurePlasticityAbstractTimingDependenceSpikeP
method), 69
tell() (spinnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamicsStructurePlasticityAbstractTimingDependenceSpikeT
method), 69
tell() (spinnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamicsStructureWeightDependenceSpikeNearestPairFileDataReader
method), 72
tell() (spinnaker.pyNN.models.neuron.plasticity.stdp.synapse_dynamics.AbstractSynapseDynamicsStructureWeightDependenceSpikeNearestPairFileDataWriter
method), 72
tell() (spinnaker.pyNN.models.neuron.neuron_models.neuron_models_leaky_integrate_and_fire_data_reader.NeuronModelLeakyIntegrateAndF
74
tell() (spinnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFireDataWriter method), 75
tell() (spinnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.AbstractTypeDualExponential
method), 373
tell() (spinnaker.pyNN.models.neuron.synapse_types.AbstractTypeDualExponential method), 376
tell() (spinnman.utilities.io.file_io.FileIO method), 374
tell() (spinnman.utilities.io.FileIO method), 377
tell() (spinnman.utilities.io.memory_io.MemoryIO
method), 376
tell() (spinnman.utilities.io.MemoryIO method), 378
tell() (spinnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.AbstractTypeDualExponential.buffered_data_
method), 60
tell() (spinnaker.pyNN.models.neuron.synapse_types.AbstractTypeDualExponential
method), 66
tell() (spinnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential.storage.Buffer
method), 70
tell() (spinnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential.data_storage.BufferedFile
method), 70
tell() (spinnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential.buffered_tempfile_data_storage.Buffere
method), 71
tell() (spinnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential.storage_handlers.BufferedBytearrayDataStorage
method), 73
tell() (spinnaker.pyNN.models.neuron.threshold_types.threshold_type_stochastic.ThresholdTypeStochastic.storage

method), 73
tell_read() (spinn_storage_handlers.BufferedTempfileDataStorage method), 74
tell_write() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage method), 61
tell_write() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage method), 66
tell_write() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedBytearrayDataStorage method), 70
tell_write() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage method), 70
tell_write() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage method), 71
tell_write() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
tell_write() (spinn_storage_handlers.BufferedFileDataStorage method), 73
tell_write() (spinn_storage_handlers.BufferedTempfileDataStorage method), 74
temp_btm (spinnman.model.adc_info.ADCInfo attribute), 332
temp_btm (spinnman.model.ADCInfo attribute), 347
temp_ext_0 (spinnman.model.adc_info.ADCInfo attribute), 332
temp_ext_0 (spinnman.model.ADCInfo attribute), 347
temp_ext_1 (spinnman.model.adc_info.ADCInfo attribute), 332
temp_ext_1 (spinnman.model.ADCInfo attribute), 347
temp_top (spinnman.model.adc_info.ADCInfo attribute), 332
temp_top (spinnman.model.ADCInfo attribute), 347
THRESHOLD (pacman.operations.rigged_algorithms.random_placer.RandomPlacer attribute), 190
ThresholdTypeMaassStochastic (class in spynnaker.pyNN.models.neuron.threshold_types), 777
ThresholdTypeMaassStochastic (class in spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic), 774
ThresholdTypeMulticastDeviceControl (class in spynnaker.pyNN.external_devices_models), 665
ThresholdTypeMulticastDeviceControl (class in spynnaker.pyNN.external_devices_models.threshold_type_multicast_device_control), 658
ThresholdTypeStatic (class in spynnaker.pyNN.models.neuron.threshold_types), 776
ThresholdTypeStatic (class in spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static), 775
time (spinn_front_end_common.utility_models.multi_cast_command.MultiCastCommand attribute), 598
time (spinn_front_end_common.utility_models.MultiCastCommand attribute), 609
time (spinnman.model.cpu_info.CPUInfo attribute), 338
time (spinnman.model.CPUInfo attribute), 353
time_between_send (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot.AbstractPushBotDataStorage attribute), 649
time_between_send (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotDataStorage attribute), 640
TIME_EXPECTED_TO_RUN (spinn_front_end_common.mapping_algorithms.on_chip_router_compression.compressor_provenance.CompressorProvenance attribute), 550
time_millisecond (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 319
time_milliseconds (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
time_phase_scale (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableBoot attribute), 319
time_phase_scale (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
time_scale_factor (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon attribute), 834
time_scale_factor() (spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePlugin static method), 837
timed_commands (spinn_front_end_common.abstract_models.abstract_send_me_multi_cast_command.AbstractSendMeMultiCastCommand attribute), 498
timed_commands (spinn_front_end_common.abstract_models.AbstractSendMeMultiCastCommand attribute), 501
timed_commands (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalSpinnakerLinkFPGARetinaDevice attribute), 654
timed_commands (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
timed_commands (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichSpinnakerLinkRetinaDevice attribute), 658
timed_commands (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 665
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot.AbstractPushBotDataStorage attribute), 649
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotDataStorage attribute), 640
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotMultiCastCommand attribute), 650
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetDevice attribute), 627

timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_led_device.PushBotEthernetLEDDevic attribute), 628
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorD attribute), 628
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_speaker_device.PushBotEthernetSpeak attribute), 630
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_laser_device.PushBotEthernetLaserDevice attribute), 634
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_common.PushBotEthernetLEDDevice attribute), 635
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice attribute), 635
timed_commands (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_speaker_device.PushBotEthernetSpeakerDevice attribute), 636
timeout (spinnman.exceptions.SpinnmanTimeoutException attribute), 382
TIMEOUT_10_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_1280_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_160_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_20_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_2560_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_320_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_40_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_640_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 380
TIMEOUT_80_ms (spinnman.constants.IPTAG_TIME_OUT_WAIT_TIMES attribute), 381
Timer (class in spinn_utilities.timer), 13
TIMER (spinnman.messages.scp.enums.Signal attribute), 280
TIMER (spinnman.messages.scp.enums.signal.Signal attribute), 277
TIMER (spinnman.model.enums.RunTimeError attribute), 328
TIMER (spinnman.model.enums.RunTimeError attribute), 330
TIMER_TIC_HAS_OVERRUN (spinn_front_end_common.interface.provenance.provides_proven attribute), 514
TIMER_TIC_HAS_OVERRUN (spinn_front_end_common.interface.provenance.ProvidesProvena attribute), 545
timescale_factor (spinn_front_end_common.interface.abstract_spinnaker_b attribute), 549
timestamps (spinn_front_end_common.interface.buffer_management.storag attribute), 514
timestamps (spinn_front_end_common.interface.buffer_management.storag attribute), 515
timing_dependence (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 748
timing_dependence (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics attribute), 764
TimingDependencePfisterSpikeTriplet (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 749
TimingDependencePfisterSpikeTriplet (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim 745
TimingDependenceRecurrent (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 750
TimingDependenceRecurrent (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim 746
TimingDependenceSpikeNearestPair (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 750
TimingDependenceSpikeNearestPair (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim 747
TimingDependenceSpikePair (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 749
TimingDependenceSpikePair (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim 747
TimingDependenceVogels2011 (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 751
TimingDependenceVogels2011 (class in spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim 748
TOGGLE (spinnman.messages.scp.enums.led_action.LEDAction

attribute), 275
TOGGLE (spinnman.messages.scp.enums.LEDAction attribute), 279
TokenBucket (class in spinnman.connections), 247
TokenBucket (class in spinnman.connections.token_bucket), 245
tokens (spinnman.connections.token_bucket.TokenBucket attribute), 245
tokens (spinnman.connections.TokenBucket attribute), 247
total_available_user_cores (spinn_machine.Machine attribute), 50
total_available_user_cores (spinn_machine.machine.Machine attribute), 33
total_cores (spinn_machine.Machine attribute), 50
total_cores (spinn_machine.machine.Machine attribute), 33
total_processors (spinnman.model.executable_targets.ExecutableTargets attribute), 341
total_processors (spinnman.model.ExecutableTargets attribute), 355
total_region_size (spinn_front_end_common.interface.buffer_management.storage_objects.buffered_sending_region.BufferedSendingR attribute), 514
total_region_size (spinn_front_end_common.interface.buffer_management.storage_objects.BufferedSendingRegion attribute), 522
TOTAL_REQUIRED_MALLOCS (spinn_front_end_common.utility_models.command_sender.CommandSender attribute), 590
TOTAL_REQUIRED_MALLOCS (spinn_front_end_common.utility_models.CommandSender.CommandSenderMachineVertex attribute), 605
traceback (pacman.exceptions.PacmanAlgorithmFailedToCompleteException attribute), 211
traffic_identifier (pacman.model.resources.iptag_resource.IPtagResource attribute), 154
traffic_identifier (pacman.model.resources.IPtagResource attribute), 161
traffic_identifier (pacman.model.resources.specific_board_iptag_resource.SpecificBoardTagResource attribute), 158
TRAFFIC_IDENTIFIER (spinn_front_end_common.utility_models.live_packet_gather.LivePacketGatherMachineVertex attribute), 596
TRAFFIC_IDENTIFIER (spinn_front_end_common.utility_models.LivePacketGather.LivePacketGatherMachineVertex attribute), 608
traffic_identifier (spinn_machine.tags.IPTag attribute), 20
traffic_identifier (spinn_machine.tags.iptag.IPTag attribute), 19
traffic_type (pacman.model.graphs.abstract_edge.AbstractEdge attribute), 137
traffic_type (pacman.model.graphs.abstract_outgoing_edge_partition.AbstractOutgoingEdgePartition attribute), 141
traffic_type (pacman.model.graphs.AbstractEdge attribute), 142
traffic_type (pacman.model.graphs.AbstractOutgoingEdgePartition attribute), 146
traffic_type (pacman.model.graphs.application.application_edge.Applicatio attribute), 110
traffic_type (pacman.model.graphs.application.ApplicationEdge attribute), 115
traffic_type (pacman.model.graphs.impl.outgoing_edge_partition.OutgoingE attribute), 126
traffic_type (pacman.model.graphs.impl.OutgoingEdgePartition attribute), 129
traffic_type (pacman.model.graphs.machine.machine_edge.MachineEdge attribute), 130
traffic_type (pacman.model.graphs.machine.MachineEdge attribute), 134
traffic_weight (pacman.model.graphs.abstract_outgoing_edge_partition.Abs attribute), 141
traffic_weight (pacman.model.graphs.AbstractOutgoingEdgePartition attribute), 146
traffic_weight (pacman.model.graphs.impl.outgoing_edge_partition.Outgoi attribute), 126
traffic_weight (pacman.model.graphs.impl.OutgoingEdgePartition attribute), 129
traffic_weight (pacman.model.graphs.machine.machine_edge.MachineEdge attribute), 130
traffic_weight (pacman.model.graphs.machine.MachineEdge attribute), 134
Transceiver (class in spinn_machine.transceiver), 33
transceiver (spinn_front_end_common.interface.abstract_spinnaker_base.A attribute), 549
transceiver (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
transceiver (spinn_front_end_common.utilities.FailedState attribute), 583
transceiver (spinn_front_end_common.utilities.simulator_interface.Simulato attribute), 583
transceiver (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
transmission_seq (spinnman.messages.scp.impl.iptag_get_info_response.IPTagGetInfoRe attribute), 288
translate_control_packet() (spinnaker.pyNN.external_devices_models.abstract_ethernet_translato method), 651
translate_control_packet() (spinnaker.pyNN.external_devices_models.AbstractEthernetTranslator method), 659
translate_control_packet() (spinnaker.pyNN.external_devices_models.push_bot.push_bot_etherne method), 631
translate_control_packet() (spinnaker.pyNN.external_devices_models.push_bot.push_bot_etherne method), 637

translate_iobuf_extraction_elements() (in module spinn_front_end_common.utilities.helpful_functions), 582
translate_parameters() (in module spynnaker.pyNN.utilities.utility_calls), 832
TRANSMISSION_EVENT_OVERFLOW (spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFr attribute), 544
TRANSMISSION_EVENT_OVERFLOW (spinn_front_end_common.interface.provenance.ProvidesProvenanceDataFromMachineImpl.PROVENANCE_DATA_ENTR attribute), 545
TTO (spinnman.messages.scp.enums.iptag_command.IPTagCommand attribute), 274
TTO (spinnman.messages.scp.enums.IPTagCommand attribute), 279
turn_left() (pacman.operations.rigged_algorithms.hilbert_state.HilbertState method), 189
turn_off_sensor_reporting() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 821
turn_off_sensor_reporting() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 826
turn_off_sensor_reporting_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 821
turn_off_sensor_reporting_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
turn_right() (pacman.operations.rigged_algorithms.hilbert_state.HilbertState method), 189
txt_indent (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 445
txt_indent (data_specification.DataSpecificationGenerator attribute), 483
type (spinnman.exceptions.SpinnmanInvalidParameterTypeException attribute), 382
TYPES_OF_FIELDS (class in pacman.utilities.algorithm_utilities.field_based_system_utilities), 195

U

U016 (data_specification.enums.data_type.DataType attribute), 416
U016 (data_specification.enums.DataType attribute), 419
U032 (data_specification.enums.data_type.DataType attribute), 416
U032 (data_specification.enums.DataType attribute), 419
U064 (data_specification.enums.data_type.DataType attribute), 416
U064 (data_specification.enums.DataType attribute), 419
U08 (data_specification.enums.data_type.DataType attribute), 416
U08 (data_specification.enums.DataType attribute), 419
U1616 (data_specification.enums.data_type.DataType attribute), 416
U1616 (data_specification.enums.DataType attribute), 419
U3232 (data_specification.enums.data_type.DataType attribute), 416
U3232 (data_specification.enums.DataType attribute), 419
U88 (data_specification.enums.data_type.DataType attribute), 417
U88 (data_specification.enums.DataType attribute), 420
u_init (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh. attribute), 736
u_init (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 741
uart_id (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.Mu attribute), 821
uart_id (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 826
UDPConnection (class in spinnman.connections.udp_packet_connections), 238
UDPConnection (class in spinnman.connections.udp_packet_connections.udp_connection), 234
UDPListenableConnection (class in spinnman.connections.udp_packet_connections), 241
UDPListenableConnection (class in spinnman.connections.udp_packet_connections.udp_listenable_connec 236
UINT16 (data_specification.enums.data_type.DataType attribute), 417
UINT16 (data_specification.enums.DataType attribute), 420
UINT32 (data_specification.enums.data_type.DataType attribute), 417
UINT32 (data_specification.enums.DataType attribute), 420
UINT64 (data_specification.enums.data_type.DataType attribute), 417
UINT64 (data_specification.enums.DataType attribute), 420
UINT8 (data_specification.enums.data_type.DataType attribute), 417
UINT8 (data_specification.enums.DataType attribute), 420
unallocate_resources() (pacman.utilities.utility_objs.resource_tracker.ResourceTracker method), 203
unallocate_resources() (pacman.utilities.utility_objs.ResourceTracker method), 207
UNDEF (spinnman.model.enums.run_time_error.RunTimeError

attribute), 328
UNDEF (spinnman.model.enums.RunTimeError attribute), 330
UnexpectedConfigException, 5
unfilled (data_specification.memory_region.MemoryRegion attribute), 456
unfilled (data_specification.MemoryRegion attribute), 488
UnimplementedDSECommand, 455
UnimplementedDSGCommand, 455
unix_timestamp (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 319
unix_timestamp (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
unset_simulator() (in module spinn_front_end_common.utilities.globals_variables), 579
UP_POLARITY (spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARetinaDevice attribute), 654
UP_POLARITY (spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice attribute), 662
UP_POLARITY (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 657
UP_POLARITY (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 665
update() (spinn_machine.utilities.ordered_set.OrderedSet method), 21
update() (spinn_machine.utilities.progress_bar.ProgressBar method), 22
update() (spinn_utilities.ordered_set.OrderedSet method), 11
update() (spinn_utilities.progress_bar.ProgressBar method), 12
update_buffer() (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex method), 603
update_buffer() (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex method), 614
update_chip_coordinates() (spinnman.connections.udp_packet_connections.scamp_connections.SCAMPConnection method), 233
update_chip_coordinates() (spinnman.connections.udp_packet_connections.SCAMPConnection method), 242
update_extra_inputs() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 549
update_extra_mapping_inputs() (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase method), 549
update_last_operation() (spinn_front_end_common.interface.buffer_manage method), 516
update_last_operation() (spinn_front_end_common.interface.buffer_manage method), 523
update_last_received_sequence_number() (spinn_front_end_common.interface.buffer_management.storage_ method), 514
update_last_received_sequence_number() (spinn_front_end_common.interface.buffer_management.storage_ method), 522
update_live_packet_gather_tracker() (spynnaker.pyNN.spyNNaker_external_device_plugin_manager.SpynnaKerPlugin static method), 837
update_master_population_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract method), 728
update_master_population_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ method), 729
update_master_population_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_ method), 731
update_master_population_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterI method), 731
update_master_population_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.MasterI method), 733
update_read_pointer() (spinn_front_end_common.interface.buffer_manager method), 516
update_read_pointer() (spinn_front_end_common.interface.buffer_manager method), 523
update_runtime() (spinn_front_end_common.utilities.scp.update_runtime_p method), 566
update_runtime() (spinn_front_end_common.utilities.scp.UpdateRuntimePr method), 567
update_sdp_header_for_udp_send() (in module spinnman.connections.udp_packet_connections), 242
update_sdp_header_for_udp_send() (in module spinnman.connections.udp_packet_connections.utils), 234
update_sequence_no_for_core() (spinn_front_end_common.interface.buffer_management.storage_ method), 513
update_sequence_no_for_core() (spinn_front_end_common.interface.buffer_management.storage_ method), 520
update_weight() (spynnaker.pyNN.models.abstract_models.abstract_weight_updatable.A method), 667
update_weight() (spynnaker.pyNN.models.abstract_models.AbstractWeightUpdatable method), 669

update_weight() (spynnaker.pyNN.models.neural_projections.delay_afferent_machine_edge.DelayAfferentMachineEdge method), 696
update_weight() (spynnaker.pyNN.models.neural_projections.DelayAfferentMachineEdge method), 699
update_weight() (spynnaker.pyNN.models.neural_projections.projection_machine_edge.ProjectionMachineEdge method), 698
update_weight() (spynnaker.pyNN.models.neural_projections.ProjectionMachineEdge method), 701
UpdateRuntimeProcess (class in spinn_front_end_common.utilities.scp), 567
UpdateRuntimeProcess (class in spinn_front_end_common.utilities.scp.update_runtime_process), 566
UPPER_HALF_WORD (spinnman.messages.eieio.eieio_prefix.EIEIOPrefix attribute), 266
UPPER_HALF_WORD (spinnman.messages.eieio.EIEIOPrefix attribute), 267
use_dest_reg (data_specification.data_specification_executor.DataSpecificationExecutor attribute), 424
use_dest_reg (data_specification.DataSpecificationExecutor attribute), 462
use_payload_prefix (spinn_front_end_common.utilities.utility_objs.LivePacketGather attribute), 574
use_payload_prefix (spinn_front_end_common.utilities.utility_objs.LivePacketGather attribute), 577
use_prefix (spinn_front_end_common.utilities.utility_objs.LivePacketGather attribute), 574
use_prefix (spinn_front_end_common.utilities.utility_objs.LivePacketGather attribute), 577
use_src1_reg (data_specification.data_specification_executor.DataSpecificationExecutor attribute), 424
use_src1_reg (data_specification.DataSpecificationExecutor attribute), 462
use_src2_reg (data_specification.data_specification_executor.DataSpecificationExecutor attribute), 424
use_src2_reg (data_specification.DataSpecificationExecutor attribute), 462
use_virtual_board (spinn_front_end_common.interface.abstract_spinnaker_base.AbstractSpinnakerBase attribute), 549
use_virtual_board (spinn_front_end_common.utilities.failed_state.FailedState attribute), 579
use_virtual_board (spinn_front_end_common.utilities.FailedState attribute), 583
use_virtual_board (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface attribute), 583
use_virtual_board (spinn_front_end_common.utilities.SimulatorInterface attribute), 584
user (spinnman.model.cpu_info.CPUInfo attribute), 338
user (spinnman.model.CPUInfo attribute), 353
USER_0 (spinnman.messages.scp.enums.Signal attribute), 280
USER_0 (spinnman.messages.scp.enums.signal.Signal attribute), 277
user_0 (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
user_0 (spinnman.model.RouterDiagnostics attribute), 359
USER_1 (spinnman.messages.scp.enums.Signal attribute), 280
USER_1 (spinnman.messages.scp.enums.signal.Signal attribute), 277
user_1 (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
user_1 (spinnman.model.RouterDiagnostics attribute), 360
USER_2 (spinnman.messages.scp.enums.Signal attribute), 280
USER_2 (spinnman.messages.scp.enums.signal.Signal attribute), 277
user_2 (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
user_2 (spinnman.model.RouterDiagnostics attribute), 360
USER_3 (spinnman.messages.scp.enums.Signal attribute), 280
USER_3 (spinnman.messages.scp.enums.signal.Signal attribute), 277
user_3 (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
user_3 (spinnman.model.RouterDiagnostics attribute), 360
user_registers (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
user_registers (spinnman.model.RouterDiagnostics attribute), 360
user_temp_0 (spinnman.messages.spinnaker_boot.system_variable_boot_va attribute), 319
user_temp_0 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
user_temp_1 (spinnman.messages.spinnaker_boot.system_variable_boot_va attribute), 319
user_temp_1 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
user_temp_2 (spinnman.messages.spinnaker_boot.system_variable_boot_va attribute), 320
user_temp_2 (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
user_interface (spinnman.messages.spinnaker_boot.system_variable_boot_va attribute), 320
user_interface (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
USES_SIMULATION_INTERFACE

(spinn_front_end_common.utilities.utility_objs.executable_type.ExecutableType attribute), 573
USES_SIMULATION_INTERFACE (spinn_front_end_common.utilities.utility_objs.ExecutableType attribute), 576

V

v_init (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh attribute), 736
v_init (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate attribute), 737
v_init (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 741
v_init (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate attribute), 742
V_RECORDING_REGION (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 779
V_RECORDING_REGION (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 787
v_reset (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAndFire attribute), 738
v_reset (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire attribute), 743
v_rest (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate.NeuronModelLeakyIntegrate attribute), 737
v_rest (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrate attribute), 742
v_thresh (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic.ThresholdTypeMaassStochastic attribute), 775
v_thresh (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static.ThresholdTypeStatic attribute), 775
v_thresh (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic attribute), 777
v_thresh (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic attribute), 777
VALID_CHARS (spinn_front_end_common.interface.interface_functions.provenance_json_writer.ProvenanceJSONWriter attribute), 535
VALID_CHARS (spinn_front_end_common.interface.interface_functions.provenance_xml_writer.ProvenanceXMLWriter attribute), 535
VALID_CHARS (spinn_front_end_common.interface.interface_functions.ProvenanceJSONWriter attribute), 540
VALID_CHARS (spinn_front_end_common.interface.interface_functions.ProvenanceXMLWriter attribute), 540
validate_mars_kiss_64_seed() (in module spynnaker.pyNN.utilities.utility_calls), 832
validate_routes() (in module pacman.operations.multi_cast_router_check_functionality.valid_routes_checker), 186
value (pacman.utilities.utility_objs.Field attribute), 203
value (pacman.utilities.utility_objs.field.Field attribute), 199
value (pacman.utilities.utility_objs.flexi_field.FlexiField attribute), 199
value (pacman.utilities.utility_objs.FlexiField attribute), 204
value (spinn_front_end_common.utilities.utility_objs.provenance_data_item attribute), 574
value (spinn_front_end_common.utilities.utility_objs.ProvenanceDataItem attribute), 577
value (spinn_machine.exceptions.SpinnMachineAlreadyExistsException attribute), 27
value (spinn_machine.exceptions.SpinnMachineInvalidParameterException attribute), 27
value (spinnman.exceptions.SpinnmanInvalidParameterException attribute), 382
var() (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats method), 828
var() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 828
VARIABLE_LONG (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 779
VARIABLE_LONG (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 787
variance (spynnaker.pyNN.utilities.running_stats.RunningStats attribute), 829
verbose (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector attribute), 678
verbose (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector attribute), 688
verify_not_running() (spinn_front_end_common.interface.abstract_spinnaker_interface.AbstractSpinnakerInterface method), 549
verify_not_running() (spinn_front_end_common.utilities.failed_state.FailedState method), 579
verify_not_running() (spinn_front_end_common.utilities.FailedState method), 583
verify_not_running() (spinn_front_end_common.utilities.simulator_interface.SimulatorInterface method), 583
verify_not_running() (spinn_front_end_common.utilities.SimulatorInterface method), 584
VERSION (data_specification.data_specification_generator.DataSpecificationGenerator attribute), 425
VERSION (data_specification.DataSpecificationGenerator attribute), 463
version_info (spinnman.messages.scp.impl.get_version_response.GetVersionResponse attribute), 287
version_number (spinnman.model.version_info.VersionInfo attribute), 346
version_number (spinnman.model.VersionInfo attribute), 361
version_string (spinnman.model.version_info.VersionInfo attribute), 346
version_string (spinnman.model.VersionInfo attribute),

361
VersionInfo (class in spinnman.model), 360
VersionInfo (class in spinnman.model.version_info), 346
vertex (pacman.model.constraints.partitioner_constraints.same_atoms_as_vertex_constraint.SameAtomsAsVertexConstraint attribute), 102
vertex (pacman.model.constraints.partitioner_constraints.SameAtomsAsVertexConstraint attribute), 103
vertex (pacman.model.constraints.placer_constraints.same_chip_as_constraint.SameChipAsConstraint attribute), 105
vertex (pacman.model.constraints.placer_constraints.SameChipAsConstraint attribute), 107
vertex (pacman.model.placements.Placement attribute), 150
vertex (pacman.model.placements.placement.Placement attribute), 148
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence attribute), 745
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence attribute), 748
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_pfister_spike_triplet.TimingDependencePfisterSpikeTriplet attribute), 746
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent.TimingDependenceRecurrent attribute), 746
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair.TimingDependenceSpikeNearestPair attribute), 747
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair.TimingDependenceSpikePair attribute), 748
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels_2011.TimingDependenceVogels2011 attribute), 748
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependencePfisterSpikeTriplet attribute), 750
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceRecurrent attribute), 750
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikeNearestPair attribute), 750
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceSpikePair attribute), 749
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceVogels2011 attribute), 751
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence attribute), 752
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence attribute), 754
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_constraint.SameAtomsAsVertexConstraint attribute), 752
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_constraint.SameAtomsAsVertexConstraint attribute), 753
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_constraint.SameChipAsConstraint attribute), 753
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_constraint.SameChipAsConstraint attribute), 753
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependence attribute), 754
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependence attribute), 755
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependence attribute), 755
vertex_executable_suffix (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_pfister_spike_triplet.TimingDependencePfisterSpikeTriplet attribute), 786
vertex_executable_suffix (spynnaker.pyNN.models.neuron.SynapticManager attribute), 792
VertexBasedRoutingInfoAllocator (class in pacman.operations.routing_info_allocator_algorithms.TimingDependenceSpikeNearestPair attribute), 192
VertexSorter (class in pacman.utilities), 211
VertexSorter (class in pacman.utilities.vertex_sorter), 210
vertices (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 140
vertices (pacman.model.graphs.abstract_graph.AbstractGraph attribute), 145
vertices (pacman.model.graphs.impl.Graph attribute), 125
vertices (pacman.model.graphs.impl.graph.Graph attribute), 125
VIC (spinnman.model.enums.RunTimeError attribute), 328
VIC (spinnman.model.enums.RunTimeError attribute), 330
virtual (spinn_machine.Chip attribute), 43
virtual (spinn_machine.chip.Chip attribute), 24
virtual_chip_x (pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex attribute), 142
virtual_chip_x (pacman.model.graphs.AbstractVirtualVertex attribute), 147
virtual_chip_x (pacman.model.graphs.application.application_fpga_vertex.AbstractWeightDependence attribute), 111
virtual_chip_x (pacman.model.graphs.application.application_fpga_vertex.AbstractWeightDependence attribute), 112

virtual_chip_x (pacman.model.graphs.application.ApplicationFPGAVertex attribute), 116
virtual_chip_x (pacman.model.graphs.application.ApplicationSpiNNakerLinkVertex attribute), 118
virtual_chip_x (pacman.model.graphs.machine.machine_fpga_vertex.MachineFPGAVertex attribute), 131
virtual_chip_x (pacman.model.graphs.machine.machine_spinnaker_link_vertex.MachineSpiNNakerLinkVertex attribute), 132
virtual_chip_x (pacman.model.graphs.machine.MachineFPGAVertex attribute), 135
virtual_chip_x (pacman.model.graphs.machine.MachineSpiNNakerLinkVertex attribute), 136
virtual_chip_y (pacman.model.graphs.abstract_virtual_vertex.AbstractVirtualVertex attribute), 142
virtual_chip_y (pacman.model.graphs.AbstractVirtualVertex attribute), 147
virtual_chip_y (pacman.model.graphs.application.application_fpga_vertex.ApplicationFPGAVertex attribute), 111
virtual_chip_y (pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex attribute), 113
virtual_chip_y (pacman.model.graphs.application.ApplicationFPGAVertex attribute), 116
virtual_chip_y (pacman.model.graphs.application.ApplicationSpiNNakerLinkVertex attribute), 118
virtual_chip_y (pacman.model.graphs.machine.machine_fpga_vertex.MachineFPGAVertex attribute), 131
virtual_chip_y (pacman.model.graphs.machine.machine_spinnaker_link_vertex.MachineSpiNNakerLinkVertex attribute), 133
virtual_chip_y (pacman.model.graphs.machine.MachineFPGAVertex attribute), 135
virtual_chip_y (pacman.model.graphs.machine.MachineSpiNNakerLinkVertex attribute), 136
virtual_core_ids (spinnman.model.chip_info.ChipInfo attribute), 334
virtual_core_ids (spinnman.model.ChipInfo attribute), 349
virtual_key (spinn_front_end_common.utility_models.reverse_ip_tag_multicast_source_machine_vertex.ReverseIPTagMulticastSourceMachineVertex attribute), 603
virtual_key (spinn_front_end_common.utility_models.ReverseIPTagMulticastSourceMachineVertex attribute), 614
virtual_key (spynnaker.pyNN.models.utility_models.spike_injector.SpikeInjectorVertex attribute), 810
virtual_key (spynnaker.pyNN.models.utility_models.SpikeInjectorVertex attribute), 813
virtual_to_physical_core_map (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariableDefinition attribute), 320
virtual_to_physical_core_map (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
VirtualMachine (class in spinn_machine), 55
VirtualMachine (class in spinn_machine.virtual_machine), 39
VirtualMachineGenerator (class in spinn_front_end_common.interface.interface_functions), 541
VirtualMachineGenerator (class in spinn_front_end_common.interface.interface_functions.virtual_m 587
voltage_1_2a (spinnman.model.adc_info.ADCInfo attribute), 332
voltage_1_2a (spinnman.model.ADCInfo attribute), 347
voltage_1_2b (spinnman.model.adc_info.ADCInfo attribute), 332
voltage_1_2b (spinnman.model.ADCInfo attribute), 347
voltage_1_2c (spinnman.model.adc_info.ADCInfo attribute), 332
voltage_1_2c (spinnman.model.ADCInfo attribute), 347
voltage_1_8 (spinnman.model.adc_info.ADCInfo attribute), 332
voltage_1_8 (spinnman.model.ADCInfo attribute), 347
voltage_3_3 (spinnman.model.adc_info.ADCInfo attribute), 332
voltage_3_3 (spinnman.model.ADCInfo attribute), 347
voltage_supply (spinnman.model.adc_info.ADCInfo attribute), 332
voltage_supply (spinnman.model.ADCInfo attribute), 347

W

w_max (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 752
w_max (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 753
w_max (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 753
w_max (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 754
w_max (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 755
w_max (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 755
w_min (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 752
w_min (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 753
w_min (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 753
w_min (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 754
w_min (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 755
w_min (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence attribute), 755
wait_1 (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
wait_1 (spinnman.model.RouterDiagnostics attribute), 360

wait_2 (spinnman.model.router_diagnostics.RouterDiagnostics attribute), 345
wait_2 (spinnman.model.RouterDiagnostics attribute), 360
wait_for_confirmation() (spinn_front_end_common.interface.interface_functions.notification_protocol.NotificationProtocol method), 534
wait_for_confirmation() (spinn_front_end_common.interface.interface_functions.notification_protocol method), 539
wait_for_confirmation() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 562
wait_for_confirmation() (spinn_front_end_common.utilities.notification_protocol.NotificationProtocol method), 563
wait_for_cores_to_be_in_state() (spinnman.transceiver.Transceiver method), 406
wait_till_not_ready() (spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.FakeHBPPortalMachineProvider method), 829
wait_until_ready() (spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.FakeHBPPortalMachineProvider method), 829
warn_once() (in module spinn_utilities.logger_utils), 10
WATCHDOG (spinnman.model.enums.cpu_state.CPUState attribute), 325
WATCHDOG (spinnman.model.enums.CPUState attribute), 329
weight_dependence (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP attribute), 760
weight_dependence (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP attribute), 764
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence attribute), 752
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractWeightDependence attribute), 754
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive.WeightDependenceAdditive attribute), 752
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceAdditive attribute), 753
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multiplicative.WeightDependenceMultiplicative attribute), 753
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditive attribute), 754
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet attribute), 755
weight_maximum (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceMultiplicative attribute), 755
weight_scale (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 783
weight_scale (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 790
WeightDependenceAdditive (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 754
WeightDependenceAdditive (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive), 752
WeightDependenceAdditiveTriplet (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 755
WeightDependenceAdditiveTriplet (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet), 752
WeightDependenceMultiplicative (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 754
WeightDependenceMultiplicative (class in spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multiplicative), 753
WEST (pacman.utilities.constants.EDGES attribute), 208
WEST (spinnman.model.enums.p2p_table_route.P2PTableRoute attribute), 327
WEST (spinnman.model.enums.P2PTableRoute attribute), 331
width (spinnman.model.machine_dimensions.MachineDimensions attribute), 342
width (spinnman.model.MachineDimensions attribute), 357
width (spinnman.model.p2p_table.P2PTable attribute), 343
width (spinnman.model.P2PTable attribute), 358
WITH_PAYLOAD (spinnman.model.enums.diagnostic_filter_payload_status.DiagnosticFilterPayloadStatus attribute), 327
WITH_PAYLOAD (spinnman.model.enums.DiagnosticFilterPayloadStatus attribute), 330
WITHOUT_PAYLOAD (spinnman.model.enums.diagnostic_filter_payload_status.DiagnosticFilterPayloadStatus attribute), 327
WITHOUT_PAYLOAD (spinnman.model.enums.DiagnosticFilterPayloadStatus attribute), 330
WORD (spinnman.processes.FillDataType attribute), 363
WORD (spinnman.processes.FillDataType attribute), 369
work_out_temp_from_machine_details() (in module spinnman.utilities.utility_functions), 379
working_links (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 325

working_links (spinnman.model.ChipSummaryInfo attribute), 350
WRITE (data_specification.enums.Commands attribute), 418
WRITE (data_specification.enums.commands.Commands attribute), 415
write() (spinn_storage_handlers.abstract_classes.abstract_buffered_data_storage.AbstractBufferedDataStorage method), 61
write() (spinn_storage_handlers.abstract_classes.abstract_data_writer.AbstractDataWriter method), 64
write() (spinn_storage_handlers.abstract_classes.AbstractBufferedDataStorage method), 66
write() (spinn_storage_handlers.abstract_classes.AbstractDataWriter method), 69
write() (spinn_storage_handlers.buffered_bytearray_data_storage.BufferedBytearrayDataStorage method), 70
write() (spinn_storage_handlers.buffered_file_data_storage.BufferedFileDataStorage method), 70
write() (spinn_storage_handlers.buffered_tempfile_data_storage.BufferedTempfileDataStorage method), 71
write() (spinn_storage_handlers.BufferedBytearrayDataStorage method), 73
write() (spinn_storage_handlers.BufferedFileDataStorage method), 73
write() (spinn_storage_handlers.BufferedTempfileDataStorage method), 74
write() (spinn_storage_handlers.file_data_writer.FileDataWriter method), 72
write() (spinn_storage_handlers.FileDataWriter method), 75
write() (spinnman.utilities.io.abstract_io.AbstractIO method), 373
write() (spinnman.utilities.io.AbstractIO method), 376
write() (spinnman.utilities.io.file_io.FileIO method), 374
write() (spinnman.utilities.io.FileIO method), 377
write() (spinnman.utilities.io.memory_io.MemoryIO method), 375
write() (spinnman.utilities.io.MemoryIO method), 378
WRITE_ARRAY (data_specification.enums.Commands attribute), 418
WRITE_ARRAY (data_specification.enums.commands.Commands attribute), 415
write_array() (data_specification.data_specification_generator.DataSpecificationGenerator method), 445
write_array() (data_specification.DataSpecificationGenerator method), 483
write_byte() (spinn_storage_handlers.abstract_classes.abstract_byte_writer.AbstractByteWriter method), 62
write_byte() (spinn_storage_handlers.abstract_classes.AbstractByteWriter method), 67
write_bytes() (spinn_storage_handlers.abstract_classes.abstract_byte_writer.AbstractByteWriter method), 63
write_bytes() (spinn_storage_handlers.abstract_classes.AbstractByteWriter method), 68
write_cmd() (data_specification.data_specification_generator.DataSpecificationGenerator method), 446
write_cmd() (data_specification.DataSpecificationGenerator method), 484
write_command_to_files() (data_specification.data_specification_generator.DataSpecificationGenerator method), 446
write_command_to_files() (data_specification.DataSpecificationGenerator method), 484
write_data_spec() (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 786
write_data_spec() (spynnaker.pyNN.models.neuron.SynapticManager method), 792
write_delay_parameters() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 808
write_delay_parameters() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 812
write_exp_lut() (in module spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers), 743
write_finished_file() (in module spinn_front_end_common.utilities.helpful_functions), 582
write_finished_file() (in module spinn_utilities.helpful_functions), 10
write_fpga_register() (spinnman.transceiver.Transceiver method), 406
write_int() (spinn_storage_handlers.abstract_classes.abstract_byte_writer.AbstractByteWriter method), 63
write_int() (spinn_storage_handlers.abstract_classes.AbstractByteWriter method), 68
write_link_memory_from_bytearray() (spinnman.processes.write_memory_process.WriteMemoryProcess method), 367
write_link_memory_from_bytearray() (spinnman.processes.WriteMemoryProcess method), 371
write_link_memory_from_reader() (spinnman.processes.write_memory_process.WriteMemoryProcess method), 367
write_link_memory_from_reader() (spinnman.processes.WriteMemoryProcess method), 371
write_long() (spinn_storage_handlers.abstract_classes.abstract_byte_writer.AbstractByteWriter method), 63
write_long() (spinn_storage_handlers.abstract_classes.AbstractByteWriter method), 68
write_memory() (spinnman.transceiver.Transceiver method), 406

write_memory_flood() (spinnman.transceiver.Transceiver method), 407
write_memory_from_bytearray() (spinnman.processes.write_memory_flood_process.WriteMemoryFloodProcess method), 367
write_memory_from_bytearray() (spinnman.processes.write_memory_process.WriteMemoryProcess method), 367
write_memory_from_bytearray() (spinnman.processes.WriteMemoryFloodProcess method), 371
write_memory_from_bytearray() (spinnman.processes.WriteMemoryProcess method), 371
write_memory_from_reader() (spinnman.processes.write_memory_flood_process.WriteMemoryFloodProcess method), 367
write_memory_from_reader() (spinnman.processes.write_memory_process.WriteMemoryProcess method), 368
write_memory_from_reader() (spinnman.processes.WriteMemoryFloodProcess method), 371
write_memory_from_reader() (spinnman.processes.WriteMemoryProcess method), 371
write_neighbour_memory() (spinnman.transceiver.Transceiver method), 408
WRITE_PARAM (data_specification.enums.Commands attribute), 419
WRITE_PARAM (data_specification.enums.commands.Commands attribute), 415
WRITE_PARAM_COMPONENT (data_specification.enums.Commands attribute), 419
WRITE_PARAM_COMPONENT (data_specification.enums.commands.Commands attribute), 415
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence method), 745
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence method), 749
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_pfister_spike_triplet.TimingDependencePf method), 746
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent_timing_dependence.AbstractSynapseD method), 746
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair_timing_dependence.SpiNNaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair_timing_dependence.TimingDependenceSpi method), 747
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim method), 748
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim method), 748
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tir method), 750
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tir method), 750
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tir method), 751
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tir method), 749
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tir method), 751
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.ab method), 752
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.Ab method), 754
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.we method), 752
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.we method), 753
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.we method), 753
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.We method), 754
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.We method), 755
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.We method), 755
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.We method), 757
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent_timing_dependence.AbstractSynapseD method), 761
write_parameters() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair_timing_dependence.SpiNNaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair_timing_dependence.TimingDependenceSpi method), 759
write_parameters() (spyn-

naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamicsSTDP method), 760
write_parameters() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 763
write_parameters() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 764
write_parameters_per_neuron() (in module spynnaker.pyNN.utilities.utility_calls), 832
write_pointer (data_specification.memory_region.MemoryRegion attribute), 456
write_pointer (data_specification.MemoryRegion attribute), 488
write_profile_region_data() (in module spinn_front_end_common.interface.profiling.profile_utils), 543
write_provenance_header() (pacman.executor.algorithm_classes.abstract_algorithm.AbstractAlgorithm method), 81
write_provenance_header() (pacman.executor.algorithm_classes.AbstractAlgorithm method), 84
write_provenance_header() (pacman.executor.algorithm_classes.external_algorithm.ExternalAlgorithm method), 82
write_provenance_header() (pacman.executor.algorithm_classes.ExternalAlgorithm method), 85
write_provenance_header() (pacman.executor.algorithm_classes.python_class_algorithm.PythonClassAlgorithm method), 83
write_provenance_header() (pacman.executor.algorithm_classes.python_function_algorithm.PythonFunctionAlgorithm method), 83
write_provenance_header() (pacman.executor.algorithm_classes.PythonClassAlgorithm method), 86
write_provenance_header() (pacman.executor.algorithm_classes.PythonFunctionAlgorithm method), 86
write_repeated_value() (data_specification.data_specification_generator.DataSpecificationGenerator method), 447
write_repeated_value() (data_specification.DataSpecificationGenerator method), 485
write_setup_info() (spynnaker.pyNN.models.utility_models.delay_extension_vertex.DelayExtensionVertex method), 808
write_setup_info() (spynnaker.pyNN.models.utility_models.DelayExtensionVertex method), 812
write_short() (spinn_storage_handlers.abstract_classes.abstract_byte_writer.AbstractByteWriter method), 63
write_short() (spinn_storage_handlers.abstract_classes.AbstractByteWriter method), 68
WRITE_STRUCT (data_specification.enums.Commands attribute), 419
WRITE_STRUCT (data_specification.enums.commands.Commands attribute), 415
write_structure() (data_specification.data_specification_generator.DataSpecificationGenerator method), 447
write_structure() (data_specification.DataSpecificationGenerator method), 485
write_value() (data_specification.data_specification_generator.DataSpecificationGenerator method), 448
write_value() (data_specification.DataSpecificationGenerator method), 486
write_value_from_register() (data_specification.data_specification_generator.DataSpecificationGenerator method), 449
write_value_from_register() (data_specification.DataSpecificationGenerator method), 487
WriteFPGARegister (class in spinnman.messages.scp.impl), 305
WriteFPGARegister (class in spinnman.messages.scp.impl.write_fpga_register), 294
WriteLink (class in spinnman.messages.scp.impl), 306
WriteLink (class in spinnman.messages.scp.impl.write_link), 295
WriteMemory (class in spinnman.messages.scp.impl), 306
WriteMemory (class in spinnman.messages.scp.impl.write_memory), 295
WriteMemoryFloodProcess (class in spinnman.processes), 371
WriteMemoryFloodProcess (class in spinnman.processes.write_memory_flood_process), 367
WriteMemoryProcess (class in spinnman.processes), 371
WriteMemoryProcess (class in spinnman.processes.write_memory_process), 367

X

x (pacman.model.constraints.placer_constraints.chip_and_core_constraint.ChipAndCoreConstraint attribute), 104
x (pacman.model.constraints.placer_constraints.ChipAndCoreConstraint attribute), 106
x (pacman.model.constraints.placer_constraints.radial_placement_from_chip_constraint.RadialPlacementFromChipConstraint attribute), 105
x (pacman.model.constraints.placer_constraints.RadialPlacementFromChipConstraint attribute), 106
x (pacman.model.placements.Placement attribute), 150
x (pacman.model.placements.placement.Placement attribute), 148

x (pacman.model.routing_tables.multicast_routing_table.MulticastRoutingTable attribute), 175
x (pacman.model.routing_tables.MulticastRoutingTable attribute), 176
x (pacman.operations.multi_cast_router_check_functionality.valid_routes_check attribute), 185
x (spinn_machine.Chip attribute), 43
x (spinn_machine.chip.Chip attribute), 25
x (spinn_machine.core_subset.CoreSubset attribute), 25
x (spinn_machine.CoreSubset attribute), 43
x (spinnman.messages.eieio.command_messages.spinnaker_request_buffers.SpinnakerRequestBuffers attribute), 252
x (spinnman.messages.eieio.command_messages.spinnaker_request_read_data.SpinnakerRequestReadData attribute), 253
x (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffers attribute), 256
x (spinnman.messages.eieio.command_messages.SpinnakerRequestReadData attribute), 257
x (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariable attribute), 320
x (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
x (spinnman.model.chip_info.ChipInfo attribute), 334
x (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
x (spinnman.model.ChipInfo attribute), 349
x (spinnman.model.ChipSummaryInfo attribute), 350
x (spinnman.model.cpu_info.CPUInfo attribute), 338
x (spinnman.model.CPUInfo attribute), 353
x (spinnman.model.io_buffer.IOBuffer attribute), 342
x (spinnman.model.IOBuffer attribute), 356
x (spinnman.model.version_info.VersionInfo attribute), 346
x (spinnman.model.VersionInfo attribute), 361
x_pos (pacman.operations.rigged_algorithms.hilbert_state.HilbertState attribute), 189
x_size (spinnman.messages.spinnaker_boot.system_variable_boot_values.SystemVariable attribute), 320
x_size (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
x_size (spinnman.model.chip_info.ChipInfo attribute), 334
x_size (spinnman.model.ChipInfo attribute), 349
XOR (data_specification.enums.logic_operation.LogicOperation attribute), 417
XOR (data_specification.enums.LogicOperation attribute), 420

Y

y (pacman.model.constraints.placer_constraints.chip_and_core_constraint.ChipAndCoreConstraint attribute), 104
y (pacman.model.constraints.placer_constraints.ChipAndCoreConstraint attribute), 106
y (pacman.model.constraints.placer_constraints.radial_placement_from_chip.RadialPlacementFromChip attribute), 105
y (pacman.model.constraints.placer_constraints.RadialPlacementFromChipC attribute), 106
y (pacman.model.placements.placement.Placement attribute), 150
y (pacman.model.placements.Placement attribute), 148
y (pacman.model.routing_tables.multicast_routing_table.MulticastRoutingT attribute), 175
y (pacman.model.routing_tables.MulticastRoutingTable attribute), 177
y (pacman.operations.multi_cast_router_check_functionality.valid_routes_c attribute), 185
y (spinn_machine.Chip attribute), 43
y (spinn_machine.chip.Chip attribute), 25
y (spinn_machine.core_subset.CoreSubset attribute), 25
y (spinn_machine.CoreSubset attribute), 44
y (spinnman.messages.eieio.command_messages.spinnaker_request_buffers attribute), 252
y (spinnman.messages.eieio.command_messages.spinnaker_request_read_d attribute), 253
y (spinnman.messages.eieio.command_messages.SpinnakerRequestBuffers attribute), 256
y (spinnman.messages.eieio.command_messages.SpinnakerRequestReadDa attribute), 257
y (spinnman.messages.spinnaker_boot.system_variable_boot_values.Systen attribute), 320
y (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
y (spinnman.model.chip_info.ChipInfo attribute), 334
y (spinnman.model.chip_summary_info.ChipSummaryInfo attribute), 335
y (spinnman.model.ChipInfo attribute), 349
y (spinnman.model.ChipSummaryInfo attribute), 351
y (spinnman.model.cpu_info.CPUInfo attribute), 338
y (spinnman.model.CPUInfo attribute), 353
y (spinnman.model.io_buffer.IOBuffer attribute), 342
y (spinnman.model.IOBuffer attribute), 357
y (spinnman.model.version_info.VersionInfo attribute), 346
y (spinnman.model.VersionInfo attribute), 361
y_pos (pacman.operations.rigged_algorithms.hilbert_state.HilbertState attribute), 189
y_size (spinnman.messages.spinnaker_boot.system_variable_boot_values.S attribute), 320
y_size (spinnman.messages.spinnaker_boot.SystemVariableDefinition attribute), 324
y_size (spinnman.model.chip_info.ChipInfo attribute), 334
y_size (spinnman.model.ChipInfo attribute), 349