
sPyNNaker Documentation

Jan 29, 2020

Contents

1	spynnaker package	3
1.1	Subpackages	3
1.1.1	spynnaker.pyNN package	3
1.1.1.1	Subpackages	3
1.1.1.2	Submodules	288
1.1.1.3	spynnaker.pyNN.abstract_spinnaker_common module	288
1.1.1.4	spynnaker.pyNN.exceptions module	289
1.1.1.5	spynnaker.pyNN.spynnaker_external_device_plugin_manager module	290
1.1.1.6	spynnaker.pyNN.spynnaker_simulator_interface module	293
1.1.1.7	Module contents	293
1.2	Submodules	293
1.3	spynnaker.gsyn_tools module	293
1.4	spynnaker.plot_utils module	294
1.5	spynnaker.spike_checker module	294
1.6	Module contents	294
2	Indices and tables	295
	Python Module Index	297
	Index	303

These pages document the python code for the [sPyNNaker](#) module which is part of the [SpiNNaker](#) Project.

This code depends on [SpiNNUtils](#), [SpiNNMachine](#), [SpiNNStorageHandlers](#), [SpiNNMan](#), [PACMAN](#), [DataSpecification](#), [SpiNNFrontEndCommon](#) ([Combined_documentation](#)).

Contents:

1.1 Subpackages

1.1.1 spynnaker.pyNN package

1.1.1.1 Subpackages

spynnaker.pyNN.connections package

Submodules

spynnaker.pyNN.connections.ethernet_command_connection module

class spynnaker.pyNN.connections.ethernet_command_connection.**EthernetCommandConnection** (*trans-*
com-
man-
lo-
cal_
lo-
cal_

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** (*trans*

la-
bel,
live_
lo-
cal_
lo-
cal_

Bases: spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection

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

add_translator (*label, translator*)

spynnaker.pyNN.connections.spynnaker_live_spikes_connection module

class spynnaker.pyNN.connections.spynnaker_live_spikes_connection.**SpynnakerLiveSpikesConne**

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** (*list (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.connections.spynnaker_poisson_control_connection module

class spynnaker.pyNN.connections.spynnaker_poisson_control_connection.**SpynnakerPoissonCont**

Bases: `spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection`

Parameters

- **poisson_labels** (*iterable of str*) – Labels of Poisson populations to be controlled
- **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)
- **control_label_extension** (*str*) – The extra name added to the label of each Poisson source

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_poisson_label (*label*)

add_receive_callback (*label, live_event_callback, translate_key=False*)

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, list(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.
- **translate_key** – True if the key is to be converted to an atom ID, False if the key should stay a key

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*)

Add a callback for the start and resume state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **start_resume_callback** (*function(str, SpynnakerLiveEventConnection) -> None*) – A function to be called when the start or resume 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

set_rate (*label, neuron_id, rate*)

Set the rate of a Poisson neuron within a Poisson source

Parameters

- **label** – The label of the Population to set the rates of
- **neuron_id** – The neuron ID to set the rate of
- **rate** – The rate to set in Hz

set_rates (*label, neuron_id_rates*)

Set the rates of multiple Poisson neurons within a Poisson source

Parameters

- **label** – The label of the Population to set the rates of
- **neuron_id_rates** – A list of tuples of (neuron ID, rate) to be set

Module contents

```
class spynnaker.pyNN.connections.EthernetCommandConnection (translator,      com-  
mand_containers=None,  
local_host=None, lo-  
cal_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,      label,  
live_packet_gather_label,  
local_host=None, lo-  
cal_port=None)
```

Bases: `spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection`

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

add_translator (*label, translator*)

```
class spynnaker.pyNN.connections.SpynnakerLiveSpikesConnection (receive_labels=None,
                                                                send_labels=None,
                                                                lo-
                                                                cal_host=None,
                                                                lo-
                                                                cal_port=19999,
                                                                live_packet_gather_label='LiveSpikeRec
```

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** (*list (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

```
class spynnaker.pyNN.connections.SpynnakerPoissonControlConnection (poisson_labels=None,
                                                                lo-
                                                                cal_host=None,
                                                                lo-
                                                                cal_port=19999,
                                                                con-
                                                                trol_label_extension='_control')
```

Bases: `spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection`

Parameters

- **poisson_labels** (*iterable of str*) – Labels of Poisson populations to be controlled
- **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)
- **control_label_extension** (*str*) – The extra name added to the label of each Poisson source

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_poisson_label (*label*)

add_receive_callback (*label, live_event_callback, translate_key=False*)

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, list(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.
- **translate_key** – True if the key is to be converted to an atom ID, False if the key should stay a key

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*)

Add a callback for the start and resume state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **start_resume_callback** (function(*str*, *SpynnakerLiveEventConnection*) -> None) – A function to be called when the start or resume 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

set_rate (*label, neuron_id, rate*)

Set the rate of a Poisson neuron within a Poisson source

Parameters

- **label** – The label of the Population to set the rates of
- **neuron_id** – The neuron ID to set the rate of
- **rate** – The rate to set in Hz

set_rates (*label, neuron_id_rates*)

Set the rates of multiple Poisson neurons within a Poisson source

Parameters

- **label** – The label of the Population to set the rates of
- **neuron_id_rates** – A list of tuples of (neuron ID, rate) to be set

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_ethernet

Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Leaky integrate and fire neuron with an exponentially decaying current input

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_spinnaker_link
 Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Control module for a PushBot connected to a SpiNNaker Link

Module contents

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.PushBotLifSpinnakerLink
 Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Leaky integrate and fire neuron with an exponentially decaying current input

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.PushBotLifSpinnakerLink
 Bases: *spynnaker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl*

Control module for a PushBot connected to a SpiNNaker Link

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_device

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_send_type

The type of data to be sent.

Return type *SendType*

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_
```

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice*, *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*

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

Return type iterable([MultiCastCommand](#))

set_command_protocol (*command_protocol*)

Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

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_
```

Bases: `spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice`, `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`

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

Return type iterable([MultiCastCommand](#))

set_command_protocol (*command_protocol*)

Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

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_r
```

Bases: `spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice`, `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`

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

Return type iterable([MultiCastCommand](#))

set_command_protocol (*command_protocol*)

Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

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_

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_s

Bases: `spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice`, `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`

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

Return type iterable([MultiCastCommand](#))

set_command_protocol (*command_protocol*)

Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

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_connection

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

Bases: `spynnaker.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator`

Translates packets between PushBot Multicast packets and PushBot Wi-Fi Commands

Parameters

- **protocol** – The instance of the PushBot protocol to get keys from
- **pushbot_wifi_connection** – A Wi-Fi 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.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_connection

Bases: `spinnman.connections.abstract_classes.connection.Connection`, `spinnman.connections.abstract_classes.listenable.Listenable`

A connection to a PushBot via Wi-Fi.

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

RECV_SIZE = 1024

close()

See `spinnman.connections.Connection.close()`

get_receive_method()

Get the method that receives for this connection

is_connected()

See `spinnman.connections.Connection.is_connected()`

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

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** (*float or None*) – The timeout, or None to wait forever

Returns The data received

Return type `bytes`

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.PushBotEthernetDev`

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_send_type

The type of data to be sent.

Return type *SendType*

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.PushBotEthernetLaser`

Bases: `spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice`, `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`

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

Return type `iterable(MultiCastCommand)`

set_command_protocol (*command_protocol*)

Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetLED**

Bases: [spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice](#), [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](#)

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

Return type iterable([MultiCastCommand](#))

set_command_protocol (*command_protocol*)

Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetMot**

Bases: `spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_device.PushBotEthernetDevice`, `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`

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

Return type iterable(`MultiCastCommand`)

set_command_protocol (*command_protocol*)

Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable(`MultiCastCommand`)

timed_commands

The commands to be sent at given times in the simulation

Return type iterable(`MultiCastCommand`)

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotEthernetRet**

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.**PushBotEthernetSpeaker**

Bases: *spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.
push_bot_ethernet_device.PushBotEthernetDevice, 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*

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

Return type iterable(**MultiCastCommand**)

set_command_protocol (*command_protocol*)
Set the protocol use to send setup and shutdown commands, separately from the protocol used to control the device

start_resume_commands
The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**PushBotRetinaConnect**

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** ([AbstractEthernetTranslator](#))

Bases: [spynnaker.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator](#)

Translates packets between PushBot Multicast packets and PushBot Wi-Fi Commands

Parameters

- **protocol** – The instance of the PushBot protocol to get keys from
- **pushbot_wifi_connection** – A Wi-Fi 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.AbstractEIEIODataElement](#)) – A received multicast packet

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.**get_pushbot_wifi_connect**

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.**PushBotWiFiConnect**

Bases: [spinnman.connections.abstract_classes.connection.Connection](#), [spinnman.connections.abstract_classes.listenable.Listenable](#)

A connection to a PushBot via Wi-Fi.

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

RECV_SIZE = 1024

close()

See `spinnman.connections.Connection.close()`

get_receive_method()

Get the method that receives for this connection

is_connected()

See `spinnman.connections.Connection.is_connected()`

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

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** (*float or None*) – The timeout, or None to wait forever

Returns The data received

Return type bytearray

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.PushBotLaser

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

An enumeration.

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.PushBotLED

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

An enumeration.

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.PushBotMotor

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

An enumeration.

```
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_resolution

Bases: enum.Enum

An enumeration.

```
 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_viewer

Bases: threading.Thread

local_host

local_port

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

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

Bases: *spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice*

An enumeration.

```
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
```

An enumeration.

```
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
```

An enumeration.

```
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
```

An enumeration.

```
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
```

An enumeration.

```
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.PushBotRetinaRes
    Bases: enum.Enum
```

An enumeration.


```

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.PushBotRetinaView

```

Bases: `threading.Thread`

local_host

local_port

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard `run()` method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the `args` and `kwargs` arguments, respectively.

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_device module

```

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_laser_device.PushBotLaserDevice

```

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 = {'board_address': None, 'label': None, 'n_neurons': 1, 'start_
```

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_led_device
module

```
class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_led_device
```

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 = {'board_address': None, 'label': None, 'n_neurons': 1, 'start_
```

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_motor_device module

```
class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spi
```

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 = {'board_address': None, 'label': None, 'n_neurons': 1}
```

spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_retina_device module

```
class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spi
```

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 (graph_mapper)
```

```
routing_info (routing_info)
```

```
start_resume_commands
```

The commands needed when starting or resuming simulation

Return type iterable(*MultiCastCommand*)

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_spinnaker_link_speaker

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 = {'board_address': None, 'label': None, 'n_neurons': 1, 'start_

Module contents

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.**PushBotSpiNNakerLink**

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 = {'board_address': None, 'label': None, 'n_neurons': 1, 'start_

class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.**PushBotSpiNNakerLink**

Bases: `spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_led_device.PushBotEthernetLEDDDevice`, `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 = {'board_address': None, 'label': None, 'n_neurons': 1, 'start_  
class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.PushBotSpiNNa
```

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 = {'board_address': None, 'label': None, 'n_neurons': 1}  
class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.PushBotSpiNNa
```

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 (graph_mapper)
```

```
routing_info (routing_info)
```

```
start_resume_commands
```

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

```
class spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.PushBotSpiNNakerLink
```

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 = {'board_address': None, 'label': None, 'n_neurons': 1, 'start_
```

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.AbstractPushBotOutputDevice
    Bases: enum.Enum
    An enumeration.
    max_value
    min_value
    protocol_property
    send_type
    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.AbstractPushBotRetinaDevice
    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
        Return type iterable(MultiCastCommand)
    start_resume_commands
        The commands needed when starting or resuming simulation
        Return type iterable(MultiCastCommand)
    timed_commands
        The commands to be sent at given times in the simulation
        Return type iterable(MultiCastCommand)
```

Module contents

```
class spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotOutputDevice
    Bases: enum.Enum
    An enumeration.
    max_value
    min_value
    protocol_property
    send_type
    time_between_send
```

```

class spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotRetinaDevice (protocol,
                                                                                   res-
                                                                                   o-
                                                                                   lu-
                                                                                   tion)

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

    Return type iterable(MultiCastCommand)

start_resume_commands
    The commands needed when starting or resuming simulation

    Return type iterable(MultiCastCommand)

timed_commands
    The commands to be sent at given times in the simulation

    Return type iterable(MultiCastCommand)

```

Submodules

spynnaker.pyNN.external_devices_models.abstract_ethernet_controller module

```

class spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.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.pyNN.external_devices_models.
                     AbstractEthernetTranslator

    get_outgoing_partition_ids ()
        Get the partition IDs of messages coming out of the controller

        Return type list(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.**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.
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.**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_scaling_factor
The scaling factor used to send the payload to this device.

Return type int

device_control_send_type
The type of data to be sent.

Return type *SendType*

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.abstract_multicast_controllable_device.**SendType**

Bases: enum.Enum

The data type to be sent in the payload of the multicast packet

SEND_TYPE_ACCUM = 2

SEND_TYPE_FRACT = 4

SEND_TYPE_INT = 0

SEND_TYPE_UACCUM = 3

SEND_TYPE_UFRACT = 5

SEND_TYPE_UINT = 1

spynnaker.pyNN.external_devices_models.arbitrary_fpga_device module

class spynnaker.pyNN.external_devices_models.arbitrary_fpga_device.**ArbitraryFPGADevice**(*n_neurons*, *fpga_label*, *fpga_board_label*)

Bases: pacman.model.graphs.application.application_fpga_vertex.
ApplicationFPGAVertex, spinn_front_end_common.abstract_models.impl.
provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl

spynnaker.pyNN.external_devices_models.external_device_lif_control module

class spynnaker.pyNN.external_devices_models.external_device_lif_control.**ExternalDeviceLifControl**

Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
AbstractPyNNNeuronModelStandard*

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

create_vertex(*n_neurons*, *label*, *constraints*, *spikes_per_second*, *ring_buffer_sigma*, *incoming_spike_buffer_size*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex module

class `spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex.ExternalDev`

Bases: `spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`, `spynnaker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController`, `spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraint.AbstractProvidesOutgoingPartitionConstraints`, `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.

dependent_vertices ()

Return the vertices which this vertex depends upon

Return type `iterable(ApplicationVertex)`

edge_partition_identifiers_for_dependent_vertex (*vertex*)

Return the dependent edge identifiers for a particular dependent vertex.

Parameters **vertex** (`ApplicationVertex`) –

Return type iterable(str)

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.pyNN.external_devices_models.
AbstractEthernetTranslator*

get_outgoing_partition_constraints (partition)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

get_outgoing_partition_ids ()

Get the partition IDs of messages coming out of the controller

Return type list(str)

routing_key_partition_atom_mapping (routing_info, partition)

Returns a list of atom to key mapping.

Parameters

- **routing_info** (*RoutingInfo*) – the routing info object to consider
- **partition** (*AbstractOutgoingEdgePartition*) – the routing partition to handle.

Returns a iterable of tuples of atom IDs to keys.

Return type iterable(tuple(int,int))

spynnaker.pyNN.external_devices_models.external_spinnaker_link_cochlea_device module

class spynnaker.pyNN.external_devices_models.external_spinnaker_link_cochlea_device.**ExternalSpinnakerLinkCochleaDevice**

Bases: pacman.model.graphs.application.application_spinnaker_link_vertex.
ApplicationSpiNNakerLinkVertex, spinn_front_end_common.abstract_models.
impl.provides_key_to_atom_mapping_impl.ProvidesKeyToAtomMappingImpl

spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device module**class** spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.**Ext**

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** –
- **board_address** –

DOWN_POLARITY = 'DOWN'**MERGED_POLARITY** = 'MERGED'**MODE_128** = '128'**MODE_16** = '16'**MODE_32** = '32'**MODE_64** = '64'**UP_POLARITY** = 'UP'**static get_n_neurons** (*mode, polarity*)**get_outgoing_partition_constraints** (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

pause_stop_commands

The commands needed when pausing or stopping simulation

Return type iterable([MultiCastCommand](#))

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable([MultiCastCommand](#))

timed_commands

The commands to be sent at given times in the simulation

Return type iterable([MultiCastCommand](#))

`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_fr`

`spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.get_y_fr`

spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device module

class `spynnaker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.MunichMotor`

Bases: `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_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_constraint.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** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

default_initial_values = {}

default_parameters = {'board_address': None, 'continue_if_not_different': True, 'del

dependent_vertices ()

Return the vertices which this vertex depends upon

Return type *iterable(ApplicationVertex)* Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex (*vertex*)

Return the dependent edge identifiers for a particular dependent vertex.

Parameters **vertex** (*ApplicationVertex*) –

Return type *iterable(str)* Return the dependent edge identifier

generate_data_specification (*spec, placement, routing_info, machine_time_step, time_scale_factor*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – the placement the vertex is located at

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 *ExecutableType*

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type *list(AbstractConstraint)*

get_resources_used_by_atoms (*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *ResourceContainer*

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

n_atoms

The number of atoms in the vertex

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.**MunichRet**

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 = {'board_address': None, 'label': 'MunichRetinaDevice', 'polarit

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

pause_stop_commands

The commands needed when pausing or stopping simulation

Return type iterable(*MultiCastCommand*)

start_resume_commands

The commands needed when starting or resuming simulation

Return type iterable(*MultiCastCommand*)

timed_commands

The commands to be sent at given times in the simulation

Return type iterable(*MultiCastCommand*)

`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.Thresh`

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

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

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.pyNN.external_devices_models.AbstractEthernetTranslator*

get_outgoing_partition_ids ()

Get the partition IDs of messages coming out of the controller

Return type list(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.
AbstractEIEIODataElement) – A received multicast packet

class spynnaker.pyNN.external_devices_models.**ArbitraryFPGADevice** (*n_neurons*,
fpga_link_id,
fpga_id,
board_address=None,
label=None)

Bases: pacman.model.graphs.application.application_fpga_vertex.
ApplicationFPGAVertex, spinn_front_end_common.abstract_models.impl.
ProvidesKeyToAtomMappingImpl.ProvidesKeyToAtomMappingImpl

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_scaling_factor

The scaling factor used to send the payload to this device.

Return type int

device_control_send_type

The type of data to be sent.

Return type *SendType*

`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 (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

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

```
create_vertex(n_neurons, label, constraints, spikes_per_second, ring_buffer_sigma, incoming_spike_buffer_size)
```

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

```
class spynnaker.pyNN.external_devices_models.ExternalCochleaDevice (n_neurons,
                                                                    spin-
                                                                    naker_link,
                                                                    la-
                                                                    bel=None,
                                                                    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
```

```
class spynnaker.pyNN.external_devices_models.ExternalFPGARetinaDevice (mode,
                                                                    retina_key,
                                                                    spin-
                                                                    naker_link_id,
                                                                    po-
                                                                    larity,
                                                                    la-
                                                                    bel=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** –
- **board_address** –

```
DOWN_POLARITY = 'DOWN'
```

```
MERGED_POLARITY = 'MERGED'
```

```
MODE_128 = '128'
```

```
MODE_16 = '16'
```

```
MODE_32 = '32'
```

```
MODE_64 = '64'
```

```
UP_POLARITY = 'UP'
```

```
static get_n_neurons (mode, polarity)
```

```
get_outgoing_partition_constraints (partition)
```

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

```
pause_stop_commands
```

The commands needed when pausing or stopping simulation

Return type iterable(*MultiCastCommand*)

```
start_resume_commands
```

The commands needed when starting or resuming simulation

Return type iterable(*MultiCastCommand*)

```
timed_commands
```

The commands to be sent at given times in the simulation

Return type iterable(*MultiCastCommand*)

```
class spynnaker.pyNN.external_devices_models.MunichMotorDevice(spinnaker_link_id,
                                                             board_address=None,
                                                             speed=30, sample_time=4096,
                                                             up-date_time=512,
                                                             delay_time=5,
                                                             delta_threshold=23,
                                                             continue_if_not_different=True,
                                                             label=None)
```

Bases: `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_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_constraint.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** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str* or *None*) – human readable label for the machine vertex
- **constraints** (*iterable* (*AbstractConstraint*)) – Constraints to be passed on to the machine vertex

default_initial_values = {}

default_parameters = {'board_address': None, 'continue_if_not_different': True, 'del

dependent_vertices ()

Return the vertices which this vertex depends upon

Return type `iterable(ApplicationVertex)` Return the vertices which this vertex depends upon

edge_partition_identifiers_for_dependent_vertex (*vertex*)

Return the dependent edge identifiers for a particular dependent vertex.

Parameters **vertex** (*ApplicationVertex*) –

Return type `iterable(str)` Return the dependent edge identifier

generate_data_specification (*spec*, *placement*, *routing_info*, *machine_time_step*,
time_scale_factor)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – the placement the vertex is located at

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 ExecutableType

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

get_resources_used_by_atoms (*vertex_slice*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *ResourceContainer*

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

n_atoms

The number of atoms in the vertex

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*,
spin-
naker_link_id,
position, *la-*
bel=None, *po-*
larity=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 = {'board_address': None, 'label': 'MunichRetinaDevice', 'polarit
```

```
get_outgoing_partition_constraints(partition)
```

Get constraints to be added to the given edge that comes out of this vertex.

Parameters *partition* (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

```
pause_stop_commands
```

The commands needed when pausing or stopping simulation

Return type iterable(*MultiCastCommand*)

```
start_resume_commands
```

The commands needed when starting or resuming simulation

Return type iterable(*MultiCastCommand*)

```
timed_commands
```

The commands to be sent at given times in the simulation

Return type iterable(*MultiCastCommand*)

```
class spynnaker.pyNN.external_devices_models.ThresholdTypeMulticastDeviceControl(device)
```

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

```
add_parameters(parameters)
```

Add the initial values of the parameters to the parameter holder

Parameters *parameters* (*spinn_utilities.ranged.range_dictionary*.
RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.RangeDictionary) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type *int*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

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.**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 ()

Clear 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, using_extra_monitor_cores, placements=None, monitor_api=None, monitor_placement=None, monitor_cores=None, handle_time_out_configuration=True, fixed_routes=None*)

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.

get_synapse_id_by_target (*target*)

Get the ID of a synapse given the name.

Parameters *target* (*str*) – The name of the synapse

Return type int

set_synapse_dynamics (*synapse_dynamics*)

Set the synapse dynamics of this vertex.

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*)
Determine if this edge should be filtered out

Parameters **graph_mapper** – the mapper between graphs

Returns True if the edge should be filtered

Return type bool

spynnaker.pyNN.models.abstract_models.abstract_population_initializable module

class spynnaker.pyNN.models.abstract_models.abstract_population_initializable.**AbstractPopulationInitializable**
Bases: object

Indicates that this object has properties that can be initialised by a PyNN Population

get_initial_value (*variable, selector=None*)
Gets the value for any variable whose in initialize_parameters.keys
Should return the current value not the default one.
Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added.

Parameters

- **variable** (*str*) – variable name with or without *_init*
- **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in SpiNNUtils.spinn_utilities.ranged.abstract_sized.py

Returns A list or an Object which act like a list

get_initial_values (*selector=None*)
A dict containing the initial values of the state variables.

Parameters **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in SpiNNUtils.spinn_utilities.ranged.abstract_sized.py

initial_values
A dict containing the initial values of the state variables.

initialize (*variable, value*)
Set the initial value of one of the state variables of the neurons in this population.

initialize_parameters
List the parameters that are initializable.
If “foo” is initializable there should be a setter *initialize_foo* and a getter property *foo_init*

Returns list of property names

set_initial_value (*variable, value, selector=None*)
Sets the value for any variable whose in initialize_parameters.keys
Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added

Parameters

- **variable** (*str*) – variable name with or without *_init*
- **value** – New value for the variable
- **selector** – a description of the subrange to accept. Or None for all. See: `_selector_to_ids` in `SpiNNUtils.spinn_utilities.ranged.abstract_sized.py`

Returns A list or an Object which act like a list

spynnaker.pyNN.models.abstract_models.abstract_population_settable module

class spynnaker.pyNN.models.abstract_models.abstract_population_settable.**AbstractPopulationSettable**
 Bases: `spynnaker.pyNN.models.abstract_models.abstract_settable.AbstractSettable`

Indicates that some properties of this object can be accessed from the PyNN population set and get methods.

get_value_by_selector (*selector, key*)

Gets the value for a particular key but only for the selected subset.

Parameters

- **selector** – See `RangedList.get_value_by_selector` as this is just a pass through method
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

n_atoms

” See `ApplicationVertex.n_atoms`

set_value_by_selector (*selector, key, value*)

Sets the value for a particular key but only for the selected subset.

Parameters

- **selector** – See `RangedList.set_value_by_selector` as this is just a pass through method
- **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.**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

spynnaker.pyNN.models.abstract_models.abstract_settable module

class spynnaker.pyNN.models.abstract_models.abstract_settable.**AbstractSettable**
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_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 ()

Clear 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, using_extra_monitor_cores, placements=None, monitor_api=None, monitor_placement=None, monitor_cores=None, handle_time_out_configuration=True, fixed_routes=None*)

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.

get_synapse_id_by_target (*target*)

Get the ID of a synapse given the name.

Parameters **target** (*str*) – The name of the synapse

Return type int

set_synapse_dynamics (*synapse_dynamics*)

Set the synapse dynamics of this vertex.

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.

class spynnaker.pyNN.models.abstract_models.**AbstractFilterableEdge**

Bases: object

An edge that can be filtered

filter_edge (*graph_mapper*)

Determine if this edge should be filtered out

Parameters **graph_mapper** – the mapper between graphs

Returns True if the edge should be filtered

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

get_initial_value (*variable, selector=None*)

Gets the value for any variable whose in initialize_parameters.keys

Should return the current value not the default one.

Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added.

Parameters

- **variable** (*str*) – variable name with or without *_init*
- **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in *SpiNNUtils.spinn_utilities.ranged.abstract_sized.py*

Returns A list or an Object which act like a list

get_initial_values (*selector=None*)

A dict containing the initial values of the state variables.

Parameters **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in *SpiNNUtils.spinn_utilities.ranged.abstract_sized.py*

initial_values

A dict containing the initial values of the state variables.

initialize (*variable, value*)

Set the initial value of one of the state variables of the neurons in this population.

initialize_parameters

List the parameters that are initializable.

If “foo” is initializable there should be a setter *initialize_foo* and a getter property *foo_init*

Returns list of property names

set_initial_value (*variable, value, selector=None*)

Sets the value for any variable whose in initialize_parameters.keys

Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added

Parameters

- **variable** (*str*) – variable name with or without *_init*
- **value** – New value for the variable
- **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in *SpiNNUtils.spiNN_utilities.ranged.abstract_sized.py*

Returns A list or an Object which act like a list

class spynnaker.pyNN.models.abstract_models.**AbstractPopulationSettable**

Bases: *spynnaker.pyNN.models.abstract_models.abstract_settable.AbstractSettable*

Indicates that some properties of this object can be accessed from the PyNN population set and get methods.

get_value_by_selector (*selector, key*)

Gets the value for a particular key but only for the selected subset.

Parameters

- **selector** – See *RangedList.get_value_by_selector* as this is just a pass through method
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

n_atoms

” See *ApplicationVertex.n_atoms*

set_value_by_selector (*selector, key, value*)

Sets the value for a particular key but only for the selected subset.

Parameters

- **selector** – See *RangedList.set_value_by_selector* as this is just a pass through method
- **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.**AbstractSettable**

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.**AbstractWeightUpdatable**

Bases: object

An object the weight of which can be updated

update_weight (*graph_mapper*)

Update the weight

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 a variable (e.g., membrane voltage) can be recorded from this object

clear_recording (*variable*, *buffer_manager*, *placements*, *graph_mapper*)

Clear 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*)

Get the recorded data

Parameters

- **variable** –
- **n_machine_time_steps** –
- **placements** –
- **graph_mapper** –
- **buffer_manager** –
- **machine_time_step** –

Returns

get_neuron_sampling_interval (*variable*)

Returns the current sampling interval for this variable

Parameters *variable* – PyNN name of the variable

Returns Sampling interval in micro seconds

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 variable are being recorded, False otherwise

Return type bool

set_recording (*variable, new_state=True, sampling_interval=None, indexes=None*)

Sets variable 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*)

Clear 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

get_spikes_sampling_interval ()

Return the current sampling interval for spikes

Returns Sampling interval in micro seconds

is_recording_spikes ()

Determine if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

set_recording_spikes (*new_state=True, sampling_interval=None, indexes=None*)

Set spikes to being recorded. If *new_state* is false all other parameters are ignored.

Parameters

- **new_state** (*bool*) – Set if the spikes are recording or not
- **sampling_interval** – The interval at which spikes are recorded. Must be a whole multiple of the timestep None will be taken as the timestep
- **indexes** – The indexes of the neurons that will record spikes. If None the assumption is all neurons are recording

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

set_recording (*new_state, sampling_interval=None*)

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*)

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, n_neurons*)

Bases: object

MAX_RATE = 4294967295

N_BYTES_FOR_TIMESTAMP = 4

N_BYTES_PER_INDEX = 1

N_BYTES_PER_POINTER = 4

N_BYTES_PER_RATE = 4

N_BYTES_PER_SIZE = 4

N_BYTES_PER_VALUE = 4

N_CPU_CYCLES_PER_NEURON = 8

SARK_BLOCK_SIZE = 8

check_indexes (*indexes*)

get_buffered_sdram (*variable, vertex_slice, n_machine_time_steps*)

Returns the SDRAM used for this many timesteps

If required the total is rounded up so the space will always fit

Parameters

- **variable** – The
- **vertex_slice** –

Returns

get_buffered_sdram_per_record (*variable, vertex_slice*)

Return the SDRAM used per record

Parameters

- **variable** –
- **vertex_slice** –

Returns

get_buffered_sdram_per_timestep (*variable, vertex_slice*)

Return the SDRAM used per timestep.

In the case where sampling is used it returns the average for recording and none recording based on the recording rate

Parameters

- **variable** –
- **vertex_slice** –

Returns

get_data (*vertex_slice*)

get_dtcm_usage_in_bytes (*vertex_slice*)

get_global_parameters (*vertex_slice*)

get_index_parameters (*vertex_slice*)

get_matrix_data (*label, buffer_manager, region, placements, graph_mapper, application_vertex, variable, n_machine_time_steps*)

Read 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** –
- **variable** (*str*) – PyNN name for the variable (V, gsy_inh etc.)
- **n_machine_time_steps** –

Returns**get_n_cpu_cycles** (*n_neurons*)**get_neuron_sampling_interval** (*variable*)

Return the current sampling interval for this variable

Parameters **variable** – PyNN name of the variable**Returns** Sampling interval in micro seconds**get_recordable_variables** ()**get_sampling_overflow_sdram** (*vertex_slice*)

Get the extra SDRAM that should be reserved if using per_timestep

This is the extra that must be reserved if per_timestep is an average rather than fixed for every timestep.

When sampling the average * time_steps may not be quite enough. This returns the extra space in the worst case where time_steps is a multiple of sampling rate + 1, and recording is done in the first and last time_step

Parameters **vertex_slice** –**Returns** Highest possible overflow needed**get_sdram_usage_in_bytes** (*vertex_slice*)**get_spikes** (*label, buffer_manager, region, placements, graph_mapper, application_vertex, machine_time_step*)**get_variable_sdram_usage** (*vertex_slice*)**is_recording** (*variable*)**recorded_ids_by_slice** (*vertex_slice*)**recorded_region_ids****recording_variables****set_recording** (*variable, new_state, sampling_interval=None, indexes=None*)**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_time_steps, bytes_per_time*)

Get the size of a recording region in bytes

```
spynnaker.pyNN.models.common.recording_utils.make_missing_string(missing)
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

Module contents

```
class spynnaker.pyNN.models.common.AbstractNeuronRecordable
    Bases: object
```

Indicates that a variable (e.g., membrane voltage) can be recorded from this object

```
clear_recording(variable, buffer_manager, placements, graph_mapper)
```

Clear 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)
```

Get the recorded data

Parameters

- **variable** –
- **n_machine_time_steps** –
- **placements** –
- **graph_mapper** –
- **buffer_manager** –
- **machine_time_step** –

Returns

get_neuron_sampling_interval (*variable*)

Returns the current sampling interval for this variable

Parameters **variable** – PyNN name of the variable

Returns Sampling interval in micro seconds

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 variable are being recorded, False otherwise

Return type bool

set_recording (*variable, new_state=True, sampling_interval=None, indexes=None*)

Sets variable to being recorded

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*)

Clear 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

get_spikes_sampling_interval ()

Return the current sampling interval for spikes

Returns Sampling interval in micro seconds

is_recording_spikes()

Determine if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

set_recording_spikes (*new_state=True, sampling_interval=None, indexes=None*)

Set spikes to being recorded. If new_state is false all other parameters are ignored.

Parameters

- **new_state** (*bool*) – Set if the spikes are recording or not
- **sampling_interval** – The interval at which spikes are recorded. Must be a whole multiple of the timestep None will be taken as the timestep
- **indexes** – The indexes of the neurons that will record spikes. If None the assumption is all neurons are recording

class spynnaker.pyNN.models.common.EIEIOSpikeRecorder

Bases: object

Records spikes using EIEIO format

get_dtcn_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

set_recording (*new_state, sampling_interval=None*)

class spynnaker.pyNN.models.common.NeuronRecorder (*allowed_variables, n_neurons*)

Bases: object

MAX_RATE = 4294967295

N_BYTES_FOR_TIMESTAMP = 4

N_BYTES_PER_INDEX = 1

N_BYTES_PER_POINTER = 4

N_BYTES_PER_RATE = 4

N_BYTES_PER_SIZE = 4

N_BYTES_PER_VALUE = 4

N_CPU_CYCLES_PER_NEURON = 8

SARK_BLOCK_SIZE = 8

check_indexes (*indexes*)

get_buffered_sdram (*variable, vertex_slice, n_machine_time_steps*)

Returns the SDRAM used for this many timesteps

If required the total is rounded up so the space will always fit

Parameters

- **variable** – The

- **vertex_slice** –

Returns

get_buffered_sdram_per_record (*variable*, *vertex_slice*)

Return the SDRAM used per record

Parameters

- **variable** –
- **vertex_slice** –

Returns

get_buffered_sdram_per_timestep (*variable*, *vertex_slice*)

Return the SDRAM used per timestep.

In the case where sampling is used it returns the average for recording and none recording based on the recording rate

Parameters

- **variable** –
- **vertex_slice** –

Returns

get_data (*vertex_slice*)

get_dtcm_usage_in_bytes (*vertex_slice*)

get_global_parameters (*vertex_slice*)

get_index_parameters (*vertex_slice*)

get_matrix_data (*label*, *buffer_manager*, *region*, *placements*, *graph_mapper*, *application_vertex*, *variable*, *n_machine_time_steps*)

Read 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** –
- **variable** (*str*) – PyNN name for the variable (V, gsy_inh etc.)
- **n_machine_time_steps** –

Returns

get_n_cpu_cycles (*n_neurons*)

get_neuron_sampling_interval (*variable*)

Return the current sampling interval for this variable

Parameters **variable** – PyNN name of the variable

Returns Sampling interval in micro seconds

get_recordable_variables()

get_sampling_overflow_sdram(*vertex_slice*)

Get the extra SDRAM that should be reserved if using per_timestep

This is the extra that must be reserved if per_timestep is an average rather than fixed for every timestep.

When sampling the average * time_steps may not be quite enough. This returns the extra space in the worst case where time_steps is a multiple of sampling rate + 1, and recording is done in the first and last time_step

Parameters *vertex_slice* –

Returns Highest possible overflow needed

get_sdram_usage_in_bytes(*vertex_slice*)

get_spikes(*label, buffer_manager, region, placements, graph_mapper, application_vertex, machine_time_step*)

get_variable_sdram_usage(*vertex_slice*)

is_recording(*variable*)

recorded_ids_by_slice(*vertex_slice*)

recorded_region_ids

recording_variables

set_recording(*variable, new_state, sampling_interval=None, indexes=None*)

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*)

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

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.AbstractConnector`

Bases: `object`

Abstract class that all PyNN Connectors extend.

NUMPY_SYNAPSES_DTYPE = `[('source', 'uint32'), ('target', 'uint16'), ('weight', 'float64')]`

create_synaptic_block(`pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info`)

Create a synaptic block from the data.

get_delay_maximum(`synapse_info`)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_delay_variance(`delays`)

Get the variance of the delays.

get_n_connections_from_pre_vertex_maximum(`post_vertex_slice, synapse_info, min_delay=None, max_delay=None`)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_provenance_data (*synapse_info*)

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

get_weight_mean (*weights*)

Get the mean of the weights.

get_weight_variance (*weights*)

Get the variance of the weights.

safe

set_projection_information (*machine_time_step*, *synapse_info*)

set_space (*space*)

Set the space object (allowed after instantiation).

Parameters *space* –

Returns

space

use_direct_matrix (*synapse_info*)

verbose

spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine module

class `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_m`

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector`

Indicates that the connectivity can be generated on the machine

gen_connector_id

Get the id of the connection generator on the machine

Return type `int`

gen_connector_params (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Get the parameters of the on machine generation.

Return type `numpy array of uint32`

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type `int`

gen_delay_params (*delays*, *pre_vertex_slice*, *post_vertex_slice*)

Get the parameters of the delay generator on the machine

Return type numpy array of uint32

gen_delay_params_size_in_bytes (*delays*)

The size of the delay parameters in bytes

Return type int

gen_delays_id (*delays*)

Get the id of the delay generator on the machine

Return type int

gen_weight_params_size_in_bytes (*weights*)

The size of the weight parameters in bytes

Return type int

gen_weights_id (*weights*)

Get the id of the weight generator on the machine

Return type int

gen_weights_params (*weights, pre_vertex_slice, post_vertex_slice*)

Get the parameters of the weight generator on the machine

Return type numpy array of uint32

generate_on_machine (*weights, delays*)

Determine if this instance can generate on the machine.

Default implementation returns True if the weights and delays can be generated on the machine

Return type bool

class spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_ma

Bases: enum.Enum

An enumeration.

ALL_TO_ALL_CONNECTOR = 1

FIXED_NUMBER_POST_CONNECTOR = 5

FIXED_NUMBER_PRE_CONNECTOR = 4

FIXED_PROBABILITY_CONNECTOR = 2

FIXED_TOTAL_NUMBER_CONNECTOR = 3

KERNEL_CONNECTOR = 6

ONE_TO_ONE_CONNECTOR = 0

spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector module

class spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.**AllToAllConn**

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

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, synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type *int*

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Get the parameters of the on machine generation.

Return type *numpy array of uint32*

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type *int*

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

spynnaker.pyNN.models.neural_projections.connectors.array_connector module

class spynnaker.pyNN.models.neural_projections.connectors.array_connector.**ArrayConnector** (*an*

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Make connections using an array of integers based on the IDs of the neurons in the pre- and post-populations.

Parameters `array` – An explicit boolean matrix that specifies the connections between the pre- and post-populations (see PyNN documentation)

create_synaptic_block (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

spynnaker.pyNN.models.neural_projections.connectors.csa_connector module

class spynnaker.pyNN.models.neural_projections.connectors.csa_connector.CSAConnector (*cset*,
safe=True, *callback=None*, *verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Make connections using a Connection Set Algebra (Djurfeldt 2012) description between the neurons in the pre- and post-populations. If you get `TypeError` in Python 3 see: <https://github.com/INCF/csa/issues/10>

Parameters *cset* (' ? ') – A description of the connection set between populations

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

show_connection_set (*n_pre_neurons*, *n_post_neurons*)

spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector
module**class** spynnaker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector`

Make connections using a distribution which varies with distance.

Parameters

- **d_expression** (*bool*) – 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** – 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 or None*) – 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, synapse_info*)
Create a synaptic block from the data.

d_expression

get_delay_maximum (*synapse_info*)
Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)
Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)
Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step, synapse_info*)

spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector module

class spynnaker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.**Fixed**

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

Connects a fixed number of post-synaptic neurons selected at random, to all pre-synaptic neurons.

Parameters

- **n** (*int*) – number of random post-synaptic neurons connected to pre-neurons.
- **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.
- **with_replacement** (*bool*) – this flag determines how the random selection of post-synaptic neurons is performed; if true, then every post-synaptic neuron can be chosen on each occasion, and so multiple connections between neuron pairs are possible; if false, then once a post-synaptic neuron has been connected to a pre-neuron, it can't be connected again.

allow_self_connections

create_synaptic_block (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type `int`

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)

Get the parameters of the on machine generation.

Return type `numpy array of uint32`

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type `int`

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step*, *synapse_info*)

spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector module

class spynnaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.**Fixed**

Bases: [*spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine*](#)

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.
- **with_replacement** (*bool*) – this flag determines how the random selection of pre-synaptic neurons is performed; if true, then every pre-synaptic neuron can be chosen on each occasion, and so multiple connections between neuron pairs are possible; if false, then once a pre-synaptic neuron has been connected to a post-neuron, it can't be connected again.

allow_self_connections

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step*, *synapse_info*)

spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector module

class spynnaker.pyNN.models.neural_projections.connectors.fixed_probability_connector.Fixe

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine*

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*, *synapse_info*)

Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

spynnaker.pyNN.models.neural_projections.connectors.from_list_connector module

class spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.**FromListConnector**

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 at least:

(pre_idx, post_idx)

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.

Additional items per synapse are acceptable but all synapses should have the same number of items.

column_names

conn_list

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_delay_variance (*delays*)

Get the variance of the delays.

get_extra_parameter_names ()

Getter for the names of the extra parameters

get_extra_parameters ()

Getter for the extra parameters.

Returns The extra parameters

get_n_connections (*pre_slices, post_slices, pre_hi, post_hi*)

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

get_weight_mean (*weights*)

Get the mean of the weights.

get_weight_variance (*weights*)

Get the variance of the weights.

spynnaker.pyNN.models.neural_projections.connectors.index_based_probability_connector module

class spynnaker.pyNN.models.neural_projections.connectors.index_based_probability_connector

Bases: [*spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*](#)

Make connections using a probability distribution which varies dependent upon the indices of the pre- and post-populations.

Parameters

- **index_expression** (*string*) – the right-hand side of a valid python expression for probability, involving the indices of the pre and post populations, that can be parsed by `eval()`, that computes a probability dist.
- **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, synapse_info*)
Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)
Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)
Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)
Get the maximum of the weights for this connection.

index_expression

spynnaker.pyNN.models.neural_projections.connectors.kernel_connector module

class spynnaker.pyNN.models.neural_projections.connectors.kernel_connector.**ConvolutionKernelConnector**
Bases: numpy.ndarray

class spynnaker.pyNN.models.neural_projections.connectors.kernel_connector.**KernelConnector**

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

Where the pre- and post-synaptic populations are considered as a 2D array. Connect every post(row, col) neuron to many pre(row, col, kernel) through a (kernel) set of weights and/or delays.

TODO: should these include allow_self_connections and with_replacement?

Parameters

- **shape_pre** – 2D shape of the pre population (rows/height, cols/width, usually the input image shape)
- **shape_post** – 2D shape of the post population (rows/height, cols/width)
- **shape_kernel** – 2D shape of the kernel (rows/height, cols/width)

- **(optional)** (*pre/post_start_coords*) – 2D matrix of size *shape_kernel* describing the weights
- **(optional)** – 2D matrix of size *shape_kernel* describing the delays
- **(optional)** – 2D shape of common coordinate system (for both pre and post, usually the input image sizes)
- **(optional)** – Sampling steps/jumps for pre/post pop $\Leftarrow \Rightarrow$ (*startX*, *endX*, *_stepX_*) None or 2-item array
- **(optional)** – Starting row/col for pre/post sampling $\Leftarrow \Rightarrow$ (*_startX_*, *endX*, *stepX*) None or 2-item array

compute_statistics (*weights*, *delays*, *pre_vertex_slice*, *post_vertex_slice*)

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*, *synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

gen_delay_params (*delays*, *pre_vertex_slice*, *post_vertex_slice*)

Get the parameters of the delay generator on the machine

Return type numpy array of uint32

gen_delay_params_size_in_bytes (*delays*)

The size of the delay parameters in bytes

Return type int

gen_delays_id (*delays*)

Get the id of the delay generator on the machine

Return type int

gen_weight_params_size_in_bytes (*weights*)

The size of the weight parameters in bytes

Return type int

gen_weights_id (*weights*)

Get the id of the weight generator on the machine

Return type int

gen_weights_params (*weights*, *pre_vertex_slice*, *post_vertex_slice*)

Get the parameters of the weight generator on the machine

Return type numpy array of uint32

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_kernel_vals (*vals*)

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

map_to_pre_coords (*post_r*, *post_c*)

post_as_pre (*post_vertex_slice*)

pre_as_post (*coords*)

to_post_coords (*post_vertex_slice*)

`spynnaker.pyNN.models.neural_projections.connectors.kernel_connector.shape2word(sw,
sh)`

spynnaker.pyNN.models.neural_projections.connectors.multipse_connector module

class `spynnaker.pyNN.models.neural_projections.connectors.multipse_connector.MultipseConne`

Bases: `spynnaker.pyNN.models.neural_projections.connectors.
abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

Create a multipse 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** (*int*) – This is the total number of synapses in the connection.
- **allow_self_connections** (*bool*) – Allow a neuron to connect to itself or not.
- **with_replacement** (*bool*) – When selecting, allow a neuron to be re-selected or not.

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_rng_next (*num_synapses, prob_connect*)

Get the required RNGs

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector module

class spynnaker.pyNN.models.neural_projections.connectors.one_to_one_connector.**OneToOneConnector**

Bases: [`spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`](#)

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, synapse_info*)

Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

use_direct_matrix (*synapse_info*)

spynnaker.pyNN.models.neural_projections.connectors.small_world_connector module

class spynnaker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector

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*, *synapse_info*)

Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step*, *synapse_info*)

Module contents

class spynnaker.pyNN.models.neural_projections.connectors.**AbstractConnector** (*safe=True*,
call-back=None,
verbose=False,
rng=None)

Bases: object

Abstract class that all PyNN Connectors extend.

NUMPY_SYNAPSES_DTYPE = [('source', 'uint32'), ('target', 'uint16'), ('weight', 'float64')]

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_delay_variance (*delays*)

Get the variance of the delays.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_provenance_data (*synapse_info*)

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

get_weight_mean (*weights*)

Get the mean of the weights.

get_weight_variance (*weights*)

Get the variance of the weights.

safe

set_projection_information (*machine_time_step*, *synapse_info*)

set_space (*space*)

Set the space object (allowed after instantiation).

Parameters *space* –

Returns

space

use_direct_matrix (*synapse_info*)

verbose

```
class spynnaker.pyNN.models.neural_projections.connectors.AbstractGenerateConnectorOnMachine
```

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector`

Indicates that the connectivity can be generated on the machine

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

gen_delay_params (*delays, pre_vertex_slice, post_vertex_slice*)

Get the parameters of the delay generator on the machine

Return type numpy array of uint32

gen_delay_params_size_in_bytes (*delays*)

The size of the delay parameters in bytes

Return type int

gen_delays_id (*delays*)

Get the id of the delay generator on the machine

Return type int

gen_weight_params_size_in_bytes (*weights*)

The size of the weight parameters in bytes

Return type int

gen_weights_id (*weights*)

Get the id of the weight generator on the machine

Return type int

gen_weights_params (*weights, pre_vertex_slice, post_vertex_slice*)

Get the parameters of the weight generator on the machine

Return type numpy array of uint32

generate_on_machine (*weights, delays*)

Determine if this instance can generate on the machine.

Default implementation returns True if the weights and delays can be generated on the machine

Return type bool

```
class spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector (allow_self_connection
                                                                    safe=True,
                                                                    call-
                                                                    back=None,
                                                                    ver-
                                                                    bose=None)
```

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

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, synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type `int`

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Get the parameters of the on machine generation.

Return type `numpy array of uint32`

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type `int`

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

```
class spynnaker.pyNN.models.neural_projections.connectors.ArrayConnector (array,
                                                                    safe=True,
                                                                    call-
                                                                    back=None,
                                                                    ver-
                                                                    bose=False)
```

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector`

Make connections using an array of integers based on the IDs of the neurons in the pre- and post-populations.

Parameters **array** – An explicit boolean matrix that specifies the connections between the pre- and post-populations (see PyNN documentation)

create_synaptic_block (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)
Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)
Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)
Get the maximum of the weights for this connection.

class spynnaker.pyNN.models.neural_projections.connectors.**CSAConnector** (*cset, safe=True, call-back=None, verbose=False*)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*

Make connections using a Connection Set Algebra (Djurfeldt 2012) description between the neurons in the pre- and post-populations. If you get `TypeError` in Python 3 see: <https://github.com/INCF/csa/issues/10>

Parameters **cset** (' ? ') – A description of the connection set between populations

create_synaptic_block (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)
Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)
Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)
Get the maximum of the weights for this connection.

show_connection_set (*n_pre_neurons, n_post_neurons*)

```
class spynnaker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConn
```

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector`

Make connections using a distribution which varies with distance.

Parameters

- **d_expression** (*bool*) – 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** – 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 or None*) – 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, synapse_info*)
Create a synaptic block from the data.

d_expression

get_delay_maximum (*synapse_info*)
Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)
Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)
Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step, synapse_info*)

```
class spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector (n,
al-
low_self_c
with_repla
safe=True,
call-
back=None
ver-
bose=False
rng=None)
```

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

Connects a fixed number of post-synaptic neurons selected at random, to all pre-synaptic neurons.

Parameters

- **n** (*int*) – number of random post-synaptic neurons connected to pre-neurons.
- **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.
- **with_replacement** (*bool*) – this flag determines how the random selection of post-synaptic neurons is performed; if true, then every post-synaptic neuron can be chosen on each occasion, and so multiple connections between neuron pairs are possible; if false, then once a post-synaptic neuron has been connected to a pre-neuron, it can't be connected again.

allow_self_connections

create_synaptic_block (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type `int`

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)

Get the parameters of the on machine generation.

Return type `numpy array of uint32`

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type `int`

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step*, *synapse_info*)

class spynnaker.pyNN.models.neural_projections.connectors.**FixedNumberPreConnector** (*n*,
allow_self_connections, *with_replacement*, *safe=True*, *call_back=None*, *verbose=False*,
rng=None)

Bases: [spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine](#)

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.
- **with_replacement** (*bool*) – this flag determines how the random selection of pre-synaptic neurons is performed; if true, then every pre-synaptic neuron can be chosen on each occasion, and so multiple connections between neuron pairs are possible; if false, then once a pre-synaptic neuron has been connected to a post-neuron, it can't be connected again.

allow_self_connections

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step*, *synapse_info*)

class spynnaker.pyNN.models.neural_projections.connectors.**FixedProbabilityConnector** (*p_connect*, *allow_self_connections*, *safe*, *call_back*, *reverse*, *bose*, *rng*)

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

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*, *synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*, *min_delay=None*, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

class spynnaker.pyNN.models.neural_projections.connectors.**FromListConnector** (*conn_list*, *safe=True*, *call-back=None*, *verbose=False*, *column_names=None*)

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 at least:

```
(pre_idx, post_idx)
```

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.

Additional items per synapse are acceptable but all synapses should have the same number of items.

column_names

conn_list

create_synaptic_block (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_delay_variance (*delays*)

Get the variance of the delays.

get_extra_parameter_names ()

Getter for the names of the extra parameters

get_extra_parameters ()

Getter for the extra parameters.

Returns The extra parameters

get_n_connections (*pre_slices*, *post_slices*, *pre_hi*, *post_hi*)

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*, *min_delay=None*, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

get_weight_mean (*weights*)

Get the mean of the weights.

get_weight_variance (*weights*)

Get the variance of the weights.

class spynnaker.pyNN.models.neural_projections.connectors.IndexBasedProbabilityConnector (*in*

Bases: [*spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector*](#)

Make connections using a probability distribution which varies dependent upon the indices of the pre- and post-populations.

Parameters

- **index_expression** (*string*) – the right-hand side of a valid python expression for probability, involving the indices of the pre and post populations, that can be parsed by `eval()`, that computes a probability dist.
- **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, synapse_info*)

Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

index_expression

```
class spynnaker.pyNN.models.neural_projections.connectors.MultapseConnector (num_synapses,  

                                                                    allow_  

                                                                    self_connections,  

                                                                    with_replacement=True,  

                                                                    safe=True,  

                                                                    call_  

                                                                    back=None,  

                                                                    ver_  

                                                                    bose=False,  

                                                                    rng=None)
```

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

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** (*int*) – This is the total number of synapses in the connection.
- **allow_self_connections** (*bool*) – Allow a neuron to connect to itself or not.
- **with_replacement** (*bool*) – When selecting, allow a neuron to be re-selected or not.

create_synaptic_block (*pre_slices, pre_slice_index, post_slices, post_slice_index,*
pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info)
 Create a synaptic block from the data.

gen_connector_id
 Get the id of the connection generator on the machine

Return type `int`

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index,*
pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info)
 Get the parameters of the on machine generation.

Return type `numpy array of uint32`

gen_connector_params_size_in_bytes
 The size of the connector parameters in bytes.

Return type `int`

get_delay_maximum (*synapse_info*)
 Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info,*
min_delay=None, max_delay=None)
 Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)
 Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_rng_next (*num_synapses, probab_connect*)
 Get the required RNGs

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

class spynnaker.pyNN.models.neural_projections.connectors.**OneToOneConnector** (*random_number_class*,
safe=True,
call-back=None,
vertex-bose=False)

Bases: *spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine*

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*, *synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*,
pre_vertex_slice, *post_vertex_slice*, *synapse_type*, *synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, *max_delay=None*)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

use_direct_matrix (*synapse_info*)

class spynnaker.pyNN.models.neural_projections.connectors.**SmallWorldConnector** (*degree*,
rewiring,
allow_low_self_connections,
safe=True,
call-back=None,
vertex-bose=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, synapse_info*)
Create a synaptic block from the data.

get_delay_maximum (*synapse_info*)
Get the maximum delay specified by the user in ms, or None if unbounded.

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice, synapse_info, min_delay=None, max_delay=None*)
Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)
Get the maximum of the weights for this connection.

set_projection_information (*machine_time_step, synapse_info*)

class `spynnaker.pyNN.models.neural_projections.connectors.KernelConnector` (*shape_pre, shape_post, shape_kernel, weight_kernel, delay_kernel, shape_common, pre_sample_steps, pre_start_coords, post_sample_steps, post_start_coords, safe, verbose, call-back=None*)

Bases: `spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.AbstractGenerateConnectorOnMachine`

Where the pre- and post-synaptic populations are considered as a 2D array. Connect every post(row, col) neuron to many pre(row, col, kernel) through a (kernel) set of weights and/or delays.

TODO: should these include allow_self_connections and with_replacement?

Parameters

- **shape_pre** – 2D shape of the pre population (rows/height, cols/width, usually the input image shape)
- **shape_post** – 2D shape of the post population (rows/height, cols/width)
- **shape_kernel** – 2D shape of the kernel (rows/height, cols/width)
- **(optional)** (*pre/post_start_coords*) – 2D matrix of size shape_kernel describing the weights
- **(optional)** – 2D matrix of size shape_kernel describing the delays

- **(optional)** – 2D shape of common coordinate system (for both pre and post, usually the input image sizes)
- **(optional)** – Sampling steps/jumps for pre/post pop \Leftrightarrow (startX, endX, _stepX_) None or 2-item array
- **(optional)** – Starting row/col for pre/post sampling \Leftrightarrow (_startX_, endX, stepX) None or 2-item array

compute_statistics (*weights, delays, pre_vertex_slice, post_vertex_slice*)

create_synaptic_block (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)
Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

gen_connector_params (*pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type, synapse_info*)

Get the parameters of the on machine generation.

Return type numpy array of uint32

gen_connector_params_size_in_bytes

The size of the connector parameters in bytes.

Return type int

gen_delay_params (*delays, pre_vertex_slice, post_vertex_slice*)

Get the parameters of the delay generator on the machine

Return type numpy array of uint32

gen_delay_params_size_in_bytes (*delays*)

The size of the delay parameters in bytes

Return type int

gen_delays_id (*delays*)

Get the id of the delay generator on the machine

Return type int

gen_weight_params_size_in_bytes (*weights*)

The size of the weight parameters in bytes

Return type int

gen_weights_id (*weights*)

Get the id of the weight generator on the machine

Return type int

gen_weights_params (*weights, pre_vertex_slice, post_vertex_slice*)

Get the parameters of the weight generator on the machine

Return type numpy array of uint32

get_delay_maximum (*synapse_info*)

Get the maximum delay specified by the user in ms, or None if unbounded.

get_kernel_vals (*vals*)

get_n_connections_from_pre_vertex_maximum (*post_vertex_slice*, *synapse_info*,
min_delay=None, max_delay=None)

Get the maximum number of connections between those from any neuron in the pre vertex to the 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 (*synapse_info*)

Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*synapse_info*)

Get the maximum of the weights for this connection.

map_to_pre_coords (*post_r, post_c*)

post_as_pre (*post_vertex_slice*)

pre_as_post (*coords*)

to_post_coords (*post_vertex_slice*)

Submodules

spynnaker.pyNN.models.neural_projections.delay_afferent_application_edge module

class spynnaker.pyNN.models.neural_projections.delay_afferent_application_edge.DelayAfferentApplicationEdge

Bases: `pacman.model.graphs.application.application_edge.ApplicationEdge`

create_machine_edge (*pre_vertex, post_vertex, label*)

Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (`pacman.model.graphs.machine.MachineVertex`) – The machine vertex at the start of the edge
- **post_vertex** (`pacman.model.graphs.machine.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.graphs.machine.MachineEdge`

spynnaker.pyNN.models.neural_projections.delay_afferent_machine_edge module

class spynnaker.pyNN.models.neural_projections.delay_afferent_machine_edge.DelayAfferentMachineEdge

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*)

Determine if this edge should be filtered out

Parameters **graph_mapper** – the mapper between graphs

Returns True if the edge should be filtered

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.DelayedApplicationEdge

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.graphs.machine.MachineVertex) – The machine vertex at the start of the edge
- **post_vertex** (pacman.model.graphs.machine.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.graphs.machine.MachineEdge

synapse_information

spynnaker.pyNN.models.neural_projections.delayed_machine_edge module

class spynnaker.pyNN.models.neural_projections.delayed_machine_edge.DelayedMachineEdge (*synapse_information*, *pre_vertex*, *post_vertex*, *label*, *weight*)

Bases: pacman.model.graphs.machine.machine_edge.MachineEdge, spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge

filter_edge (*graph_mapper*)

Determine if this edge should be filtered out

Parameters **graph_mapper** – the mapper between graphs

Returns True if the edge should be filtered

Return type bool

spynnaker.pyNN.models.neural_projections.projection_application_edge module

class spynnaker.pyNN.models.neural_projections.projection_application_edge.**ProjectionAppli**

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.graphs.machine.MachineVertex`) – The machine vertex at the start of the edge
- **post_vertex** (`pacman.model.graphs.machine.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.graphs.machine.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.**ProjectionMachineEd**

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*)

Determine if this edge should be filtered out

Parameters **graph_mapper** – the mapper between graphs

Returns True if the edge should be filtered

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*, *pre_population*, *post_population*, *pre_population_view*, *post_population_view*, *rng*, *synapse_dynamics*, *synapse_type*, *weights*, *delays*)

Bases: `object`

Contains the synapse information including the connector, synapse type and synapse dynamics

connector

delays

n_post_neurons

n_pre_neurons

post_population

postpop_is_view

pre_population

prepop_is_view

rng

synapse_dynamics

synapse_type

weights

Module contents

class spynnaker.pyNN.models.neural_projections.**DelayAfferentApplicationEdge** (*prevertex, de-layver-tex, la-bel=None*)

Bases: `pacman.model.graphs.application.application_edge.ApplicationEdge`

create_machine_edge (*pre_vertex, post_vertex, label*)

Create a machine edge between two machine vertices

Parameters

- **pre_vertex** (`pacman.model.graphs.machine.MachineVertex`) – The machine vertex at the start of the edge
- **post_vertex** (`pacman.model.graphs.machine.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.graphs.machine.MachineEdge`

class spynnaker.pyNN.models.neural_projections.**DelayAfferentMachineEdge** (*pre_vertex, post_vertex, la-bel, 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*)

Determine if this edge should be filtered out

Parameters **graph_mapper** – the mapper between graphs

Returns True if the edge should be filtered

Return type bool

update_weight (*graph_mapper*)

Update the weight

class spynnaker.pyNN.models.neural_projections.**DelayedApplicationEdge** (*pre_vertex, post_vertex, synapse_information, la-bel=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.graphs.machine.MachineVertex`) – The machine vertex at the start of the edge
- **post_vertex** (`pacman.model.graphs.machine.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.graphs.machine.MachineEdge`

synapse_information

```
class spynnaker.pyNN.models.neural_projections.DelayedMachineEdge(synapse_information,
                                                                    pre_vertex,
                                                                    post_vertex,
                                                                    la-
                                                                    bel=None,
                                                                    weight=1)
```

Bases: `pacman.model.graphs.machine.machine_edge.MachineEdge`, `spynnaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge`

filter_edge (`graph_mapper`)

Determine if this edge should be filtered out

Parameters **graph_mapper** – the mapper between graphs

Returns True if the edge should be filtered

Return type bool

```
class spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge(pre_vertex,
                                                                            post_vertex,
                                                                            synapse_information,
                                                                            la-
                                                                            bel=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.graphs.machine.MachineVertex`) – The machine vertex at the start of the edge
- **post_vertex** (`pacman.model.graphs.machine.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.graphs.machine.MachineEdge`

delay_edge

n_delay_stages

synapse_information

```

class spynnaker.pyNN.models.neural_projections.ProjectionMachineEdge(synapse_information,
                                                                    pre_vertex,
                                                                    post_vertex,
                                                                    la-
                                                                    bel=None,
                                                                    traf-
                                                                    fic_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)
    Determine if this edge should be filtered out

    Parameters graph_mapper – the mapper between graphs

    Returns True if the edge should be filtered

    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,
                                                                    pre_population,
                                                                    post_population,
                                                                    pre-
                                                                    pop_is_view,
                                                                    post-
                                                                    pop_is_view,
                                                                    rng,
                                                                    synapse_dynamics,
                                                                    synapse_type,
                                                                    weights=None,
                                                                    de-
                                                                    lays=None)

Bases: object

Contains the synapse information including the connector, synapse type and synapse dynamics

connector

delays

n_post_neurons

n_pre_neurons

post_population

postpop_is_view

pre_population

```

`prepop_is_view`
`rng`
`synapse_dynamics`
`synapse_type`
`weights`

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*,
data_type)

Bases: object

get_dataspec_datatype ()

get_value ()

iterator_by_slice (*slice_start*, *slice_stop*, *spec*)
Creates an Iterator.

Parameters

- **slice_start** – Inclusive start of the range
- **slice_stop** – Exclusive end of the range
- **spec** (*DataSpecificationGenerator*) – The data specification to write to

Returns Iterator

Module contents

class spynnaker.pyNN.models.neural_properties.**NeuronParameter** (*value*, *data_type*)

Bases: object

get_dataspec_datatype ()

get_value ()

iterator_by_slice (*slice_start*, *slice_stop*, *spec*)
Creates an Iterator.

Parameters

- **slice_start** – Inclusive start of the range
- **slice_stop** – Exclusive end of the range
- **spec** (*DataSpecificationGenerator*) – The data specification to write to

Returns Iterator

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.**AbstractAdd**

Bases: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents a possible additional independent input for a model.

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive module

class spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive.**Addition**

Bases: *spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput*

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type *int*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters

- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

i_alpha

i_ca2

tau_ca2

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

Module contents

class spynnaker.pyNN.models.neuron.additional_inputs.**AbstractAdditionalInput** (*data_types*)
Bases: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents a possible additional independent input for a model.

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

class spynnaker.pyNN.models.neuron.additional_inputs.**AdditionalInputCa2Adaptive** (*tau_ca2, i_ca2, i_alpha*)

Bases: *spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input.AbstractAdditionalInput*

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters `n_neurons` (*int*) – The number of neurons to get the cycles for

Return type `int`

get_units (*variable*)

Get the units of the given variable

Parameters `variable` (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the parameters
- **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as `self.struct.field_types`

Return type A list of (single value or list of values or `RangedList`)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters `variable` (*str*) – The name of the variable

Return type `bool`

`i_alpha`

`i_ca2`

`tau_ca2`

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

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.EIFConductanceAlphaPopu`
 Bases: `object`

Exponential integrate and fire neuron with spike triggered and sub-threshold adaptation currents (isfa, ista reps.)

`default_initial_values = {'gsyn_exc': 0.0, 'gsyn_inh': 0.0, 'v': -70.6, 'w': 0.0}`

```
default_parameters = {'a': 4.0, 'b': 0.0805, 'cm': 0.281, 'delta_T': 2.0, 'e_rev_E':
```

spynnaker.pyNN.models.neuron.builds.hh_cond_exp module

```
class spynnaker.pyNN.models.neuron.builds.hh_cond_exp.HHCondExp(**kwargs)
```

Bases: object

Single-compartment Hodgkin-Huxley model with exponentially decaying current input.

```
default_initial_values = {'gsyn_exc': 0.0, 'gsyn_inh': 0.0, 'v': -65.0}
```

```
default_parameters = {'cm': 0.2, 'e_rev_E': 0.0, 'e_rev_I': -80, 'e_rev_K': -90.0, 'e
```

spynnaker.pyNN.models.neuron.builds.if_cond_alpha module

```
class spynnaker.pyNN.models.neuron.builds.if_cond_alpha.IFCondAlpha(**kwargs)
```

Bases: object

Leaky integrate and fire neuron with an alpha-shaped current input.

```
default_initial_values = {'gsyn_exc': 0.0, 'gsyn_inh': 0.0, 'v': -65.0}
```

```
default_parameters = {'cm': 1.0, 'e_rev_E': 0.0, 'e_rev_I': -70.0, 'i_offset': 0, 't
```

spynnaker.pyNN.models.neuron.builds.if_cond_exp_base module

```
class spynnaker.pyNN.models.neuron.builds.if_cond_exp_base.IFCondExpBase(**kwargs)
```

Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.*

AbstractPyNNNeuronModelStandard

Leaky integrate and fire neuron with an exponentially decaying conductance input.

spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc module

```
class spynnaker.pyNN.models.neuron.builds.if_cond_exp_stoc.IFCondExpStoc(**kwargs)
```

Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.*

AbstractPyNNNeuronModelStandard

Leaky integrate and fire neuron with a stochastic threshold.

spynnaker.pyNN.models.neuron.builds.if_curr_alpha module

```
class spynnaker.pyNN.models.neuron.builds.if_curr_alpha.IFCurrAlpha(**kwargs)
```

Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.*

AbstractPyNNNeuronModelStandard

Leaky integrate and fire neuron with an alpha-shaped current-based input.

spynnaker.pyNN.models.neuron.builds.if_curr_delta module

class spynnaker.pyNN.models.neuron.builds.if_curr_delta.**IFCurrDelta**(**kwargs)
 Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.AbstractPyNNNeuronModelStandard*

Leaky integrate and fire neuron with an instantaneous current input

spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base module

class spynnaker.pyNN.models.neuron.builds.if_curr_dual_exp_base.**IFCurrDualExpBase**(**kwargs)
 Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.AbstractPyNNNeuronModelStandard*

Leaky integrate and fire neuron with two exponentially decaying excitatory current inputs, and one exponentially decaying inhibitory current input

spynnaker.pyNN.models.neuron.builds.if_curr_exp_base module

class spynnaker.pyNN.models.neuron.builds.if_curr_exp_base.**IFCurrExpBase**(**kwargs)
 Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.AbstractPyNNNeuronModelStandard*

Leaky integrate and fire neuron with an exponentially decaying current input

spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive module

class spynnaker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive.**IFCurrExpCa2Adaptive**(**kwargs)
 Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.AbstractPyNNNeuronModelStandard*

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

spynnaker.pyNN.models.neuron.builds.if_curr_exp_semd_base module

class spynnaker.pyNN.models.neuron.builds.if_curr_exp_semd_base.**IFCurrExpSEMDBase**(**kwargs)
 Bases: *spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.AbstractPyNNNeuronModelStandard*

Leaky integrate and fire neuron with an exponentially decaying current input, where the excitatory input depends upon the inhibitory input (see <https://www.cit-ec.de/en/nbs/spiking-insect-vision>)

spynnaker.pyNN.models.neuron.builds.if_facets_hardware1 module

class spynnaker.pyNN.models.neuron.builds.if_facets_hardware1.**IFFacetsConductancePopulation**
 Bases: object

Leaky integrate and fire neuron with conductance-based synapses and fixed threshold as it is resembled by the FACETS Hardware Stage 1

default_initial_values = {'v': -65.0}

```
default_parameters = {'e_rev_I': -80, 'g_leak': 40.0, 'tau_syn_E': 30.0, 'tau_syn_I':
```

spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base module

```
class spynnaker.pyNN.models.neuron.builds.izk_cond_exp_base.IzkCondExpBase(**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
               AbstractPyNNNeuronModelStandard
```

spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base module

```
class spynnaker.pyNN.models.neuron.builds.izk_curr_exp_base.IzkCurrExpBase(**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
               AbstractPyNNNeuronModelStandard
```

Module contents

```
class spynnaker.pyNN.models.neuron.builds.EIFConductanceAlphaPopulation(**kwargs)
    Bases: object
```

Exponential integrate and fire neuron with spike triggered and sub-threshold adaptation currents (isfa, ista reps.)

```
default_initial_values = {'gsyn_exc': 0.0, 'gsyn_inh': 0.0, 'v': -70.6, 'w': 0.0}
default_parameters = {'a': 4.0, 'b': 0.0805, 'cm': 0.281, 'delta_T': 2.0, 'e_rev_E':
```

```
class spynnaker.pyNN.models.neuron.builds.HHCondExp(**kwargs)
    Bases: object
```

Single-compartment Hodgkin-Huxley model with exponentially decaying current input.

```
default_initial_values = {'gsyn_exc': 0.0, 'gsyn_inh': 0.0, 'v': -65.0}
default_parameters = {'cm': 0.2, 'e_rev_E': 0.0, 'e_rev_I': -80, 'e_rev_K': -90.0, 'e
```

```
class spynnaker.pyNN.models.neuron.builds.IFCondAlpha(**kwargs)
    Bases: object
```

Leaky integrate and fire neuron with an alpha-shaped current input.

```
default_initial_values = {'gsyn_exc': 0.0, 'gsyn_inh': 0.0, 'v': -65.0}
default_parameters = {'cm': 1.0, 'e_rev_E': 0.0, 'e_rev_I': -70.0, 'i_offset': 0, 't
```

```
class spynnaker.pyNN.models.neuron.builds.IFCondExpBase(**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
               AbstractPyNNNeuronModelStandard
```

Leaky integrate and fire neuron with an exponentially decaying conductance input.

```
class spynnaker.pyNN.models.neuron.builds.IFCurrAlpha(**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
               AbstractPyNNNeuronModelStandard
```

Leaky integrate and fire neuron with an alpha-shaped current-based input.

```
class spynnaker.pyNN.models.neuron.builds.IFCurrDualExpBase(**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
               AbstractPyNNNeuronModelStandard
```

Leaky integrate and fire neuron with two exponentially decaying excitatory current inputs, and one exponentially decaying inhibitory current input

```
class spynnaker.pyNN.models.neuron.builds.IFCurrExpBase (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

Leaky integrate and fire neuron with an exponentially decaying current input

```
class spynnaker.pyNN.models.neuron.builds.IFFacetsConductancePopulation (**kwargs)
    Bases: object
```

Leaky integrate and fire neuron with conductance-based synapses and fixed threshold as it is resembled by the FACETS Hardware Stage 1

```
    default_initial_values = {'v': -65.0}
```

```
    default_parameters = {'e_rev_I': -80, 'g_leak': 40.0, 'tau_syn_E': 30.0, 'tau_syn_I':
```

```
class spynnaker.pyNN.models.neuron.builds.IzkCondExpBase (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

```
class spynnaker.pyNN.models.neuron.builds.IzkCurrExpBase (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

```
class spynnaker.pyNN.models.neuron.builds.IFCondExpStoc (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

Leaky integrate and fire neuron with a stochastic threshold.

```
class spynnaker.pyNN.models.neuron.builds.IFCurrDelta (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

Leaky integrate and fire neuron with an instantaneous current input

```
class spynnaker.pyNN.models.neuron.builds.IFCurrExpCa2Adaptive (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

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

```
class spynnaker.pyNN.models.neuron.builds.IFCurrExpSEMDBase (**kwargs)
    Bases:      spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.
                AbstractPyNNNeuronModelStandard
```

Leaky integrate and fire neuron with an exponentially decaying current input, where the excitatory input depends upon the inhibitory input (see <https://www.cit-ec.de/en/nbs/spiking-insect-vision>)

spynnaker.pyNN.models.neuron.implementations package

Submodules

spynnaker.pyNN.models.neuron.implementations.abstract_neuron_impl module

class spynnaker.pyNN.models.neuron.implementations.abstract_neuron_impl.**AbstractNeuronImpl**

Bases: object

An abstraction of a whole neuron model including all parts

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters *parameters* (spinn_utilities.ranged.range_dictionary.
RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters *state_variables* (spinn_utilities.ranged.
range_dictionary.RangeDictionary) – A holder of the state variables

binary_name

The name of the binary executable of this implementation

:rtype str

get_data (*parameters, state_variables, vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.
RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.
RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcmm_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters *n_neurons* (*int*) – The number of neurons to get the usage for

Return type int

get_global_weight_scale ()

Get the weight scaling required by this model

Return type int

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters *n_neurons* (*int*) – The number of neurons to get the cycles for

Return type int

get_n_synapse_types ()

Get the number of synapse types supported by the model

Return type int

get_recordable_units (*variable*)

Get the units of the given variable that can be recorded

Parameters *variable* (*str*) – The name of the variable

get_recordable_variable_index (*variable*)
Get the index of the variable in the list of variables that can be recorded

Parameters *variable* (*str*) – The name of the variable

Return type int

get_recordable_variables ()
Get the names of the variables that can be recorded in this model

Return type list of str

get_sdram_usage_in_bytes (*n_neurons*)
Get the SDRAM memory usage required

Parameters *n_neurons* (*int*) – The number of neurons to get the usage for

Return type int

get_synapse_id_by_target (*target*)
Get the id of a synapse given the name

Parameters *target* (*str*) – The name of the synapse

Return type int

get_synapse_targets ()
Get the target names of the synapse type

Return type array of str

get_units (*variable*)
Get the units of the given variable

Parameters *variable* (*str*) – The name of the variable

is_conductance_based
Determine if the model uses conductance

Return type bool

is_recordable (*variable*)
Determine if the given variable can be recorded

Parameters *variable* (*str*) – The name of the variable being requested

Return type bool

model_name
The name of the model

Return type str

read_data (*data*, *offset*, *vertex_slice*, *parameters*, *state_variables*)
Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the parameters to update

- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables to update

spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component module

class spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.**Abst**

Bases: object

Represents a component of a standard neural model

Parameters data_types – A list of data types in the component structure, in the order that they appear

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters parameters (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters state_variables (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_data (*parameters, state_variables, vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcmm_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters n_neurons (*int*) – The number of neurons to get the usage for

Return type int

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters n_neurons (*int*) – The number of neurons to get the cycles for

Return type int

get_sdram_usage_in_bytes (*n_neurons*)

Get the SDRAM memory usage required

Parameters n_neurons (*int*) – The number of neurons to get the usage for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters *variable* (*str*) – The name of the variable

get_values (*parameters*, *state_variables*, *vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters *variable* (*str*) – The name of the variable

Return type bool

read_data (*data*, *offset*, *vertex_slice*, *parameters*, *state_variables*)

Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters to update
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables to update

Returns The offset after reading the data

struct

The structure of the component

Return type :py:class:'spynnaker.pyNN.models.neuron.implementations.struct.Struct'

update_values (*values*, *parameters*, *state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

spynnaker.pyNN.models.neuron.implementations.neuron_impl_standard module**class** spynnaker.pyNN.models.neuron.implementations.neuron_impl_standard.**NeuronImplStandard**

Bases: `spynnaker.pyNN.models.neuron.implementations.abstract_neuron_impl.AbstractNeuronImpl`

The standard neuron implementation, consisting of various components

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

binary_name

The name of the binary executable of this implementation

:rtype str

get_data (*parameters, state_variables, vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the parameters
- **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcmm_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters **n_neurons** (*int*) – The number of neurons to get the usage for

Return type int

get_global_weight_scale ()

Get the weight scaling required by this model

Return type int

get_n_cpu_cycles (*n_neurons*)
 Get the number of CPU cycles required to update the state
Parameters *n_neurons* (*int*) – The number of neurons to get the cycles for
Return type *int*

get_n_synapse_types ()
 Get the number of synapse types supported by the model
Return type *int*

get_recordable_units (*variable*)
 Get the units of the given variable that can be recorded
Parameters *variable* (*str*) – The name of the variable

get_recordable_variable_index (*variable*)
 Get the index of the variable in the list of variables that can be recorded
Parameters *variable* (*str*) – The name of the variable
Return type *int*

get_recordable_variables ()
 Get the names of the variables that can be recorded in this model
Return type *list of str*

get_sdram_usage_in_bytes (*n_neurons*)
 Get the SDRAM memory usage required
Parameters *n_neurons* (*int*) – The number of neurons to get the usage for
Return type *int*

get_synapse_id_by_target (*target*)
 Get the id of a synapse given the name
Parameters *target* (*str*) – The name of the synapse
Return type *int*

get_synapse_targets ()
 Get the target names of the synapse type
Return type *array of str*

get_units (*variable*)
 Get the units of the given variable
Parameters *variable* (*str*) – The name of the variable

is_conductance_based
 Determine if the model uses conductance
Return type *bool*

is_recordable (*variable*)
 Determine if the given variable can be recorded
Parameters *variable* (*str*) – The name of the variable being requested
Return type *bool*

model_name
 The name of the model

Return type str

read_data (*data, offset, vertex_slice, parameters, state_variables*)

Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters to update
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables to update

spynnaker.pyNN.models.neuron.implementations.ranged_dict_vertex_slice module

class spynnaker.pyNN.models.neuron.implementations.ranged_dict_vertex_slice.**RangedDictVertexSlice**

Bases: object

A slice of a ranged dict to be used to update values

spynnaker.pyNN.models.neuron.implementations.struct module

class spynnaker.pyNN.models.neuron.implementations.struct.**Struct** (*field_types*)

Bases: object

Represents a C code structure

Parameters **field_types** (list of data_specification.enums.data_type.DataType) – The types of the fields, ordered as they appear in the struct

field_types

The types of the fields, ordered as they appear in the struct

Return type list of data_specification.enums.data_type.DataType

get_data (*values, offset=0, array_size=1*)

Get a numpy array of uint32 of data for the given values

Parameters

- **values** (list of (single value or list of values or RangedList of values)) – A list of values with length the same size as the number of fields returned by field_types
- **offset** – The offset into each of the values where to start
- **array_size** – The number of structs to generate

Return type numpy.array(dtype="uint32")

get_size_in_whole_words (*array_size=1*)

Get the size of the struct in whole words in an array of given size (default 1 item)

Parameters **array_size** – The number of elements in an array of structs

Return type int

numpy_dtype

The numpy data type of the struct

Return type numpy.dtype

read_data (*data*, *offset=0*, *array_size=1*)

Read a bytearray of data and convert to struct values

Parameters

- **data** – The data to be read
- **offset** – Index of the byte at the start of the valid data
- **array_size** – The number of struct elements to read

Returns a list of lists of data values, one list for each struct element

Module contents

class spynnaker.pyNN.models.neuron.implementations.**AbstractStandardNeuronComponent** (*data_types*)

Bases: object

Represents a component of a standard neural model

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_data (*parameters*, *state_variables*, *vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcmm_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters **n_neurons** (*int*) – The number of neurons to get the usage for

Return type int

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type *int*

get_sdram_usage_in_bytes (*n_neurons*)

Get the SDRAM memory usage required

Parameters **n_neurons** (*int*) – The number of neurons to get the usage for

Return type *int*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters
- **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as `self.struct.field_types`

Return type A list of (single value or list of values or `RangedList`)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

read_data (*data, offset, vertex_slice, parameters, state_variables*)

Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters to update
- **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the state variables to update

Returns The offset after reading the data

struct

The structure of the component

Return type :`py:class:'spynaker.pyNN.models.neuron.implementations.struct.Struct'`

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

class spynnaker.pyNN.models.neuron.implementations.**Struct** (*field_types*)

Bases: object

Represents a C code structure

Parameters **field_types** (list of `data_specification.enums.data_type.DataType`) – The types of the fields, ordered as they appear in the struct

field_types

The types of the fields, ordered as they appear in the struct

Return type list of `data_specification.enums.data_type.DataType`

get_data (*values, offset=0, array_size=1*)

Get a numpy array of uint32 of data for the given values

Parameters

- **values** (*list of (single value or list of values or RangedList of values)*) – A list of values with length the same size as the number of fields returned by `field_types`
- **offset** – The offset into each of the values where to start
- **array_size** – The number of structs to generate

Return type `numpy.array(dtype="uint32")`

get_size_in_whole_words (*array_size=1*)

Get the size of the struct in whole words in an array of given size (default 1 item)

Parameters **array_size** – The number of elements in an array of structs

Return type int

numpy_dtype

The numpy data type of the struct

Return type `numpy.dtype`

read_data (*data, offset=0, array_size=1*)

Read a bytearray of data and convert to struct values

Parameters

- **data** – The data to be read
- **offset** – Index of the byte at the start of the valid data
- **array_size** – The number of struct elements to read

Returns a list of lists of data values, one list for each struct element

```
class spynnaker.pyNN.models.neuron.implementations.NeuronImplStandard(model_name,  
                                                                    bi-  
                                                                    nary,  
                                                                    neu-  
                                                                    ron_model,  
                                                                    in-  
                                                                    put_type,  
                                                                    synapse_type,  
                                                                    thresh-  
                                                                    old_type,  
                                                                    addi-  
                                                                    tional_input_type=None)
```

Bases: `spynnaker.pyNN.models.neuron.implementations.abstract_neuron_impl.AbstractNeuronImpl`

The standard neuron implementation, consisting of various components

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

binary_name

The name of the binary executable of this implementation

:rtype str

get_data (*parameters, state_variables, vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcmm_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters **n_neurons** (*int*) – The number of neurons to get the usage for

Return type int

get_global_weight_scale ()

Get the weight scaling required by this model

Return type int

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters `n_neurons` (*int*) – The number of neurons to get the cycles for

Return type `int`

get_n_synapse_types ()

Get the number of synapse types supported by the model

Return type `int`

get_recordable_units (*variable*)

Get the units of the given variable that can be recorded

Parameters `variable` (*str*) – The name of the variable

get_recordable_variable_index (*variable*)

Get the index of the variable in the list of variables that can be recorded

Parameters `variable` (*str*) – The name of the variable

Return type `int`

get_recordable_variables ()

Get the names of the variables that can be recorded in this model

Return type `list of str`

get_sdram_usage_in_bytes (*n_neurons*)

Get the SDRAM memory usage required

Parameters `n_neurons` (*int*) – The number of neurons to get the usage for

Return type `int`

get_synapse_id_by_target (*target*)

Get the id of a synapse given the name

Parameters `target` (*str*) – The name of the synapse

Return type `int`

get_synapse_targets ()

Get the target names of the synapse type

Return type `array of str`

get_units (*variable*)

Get the units of the given variable

Parameters `variable` (*str*) – The name of the variable

is_conductance_based

Determine if the model uses conductance

Return type `bool`

is_recordable (*variable*)

Determine if the given variable can be recorded

Parameters `variable` (*str*) – The name of the variable being requested

Return type `bool`

model_name

The name of the model

Return type `str`

read_data (*data, offset, vertex_slice, parameters, state_variables*)

Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters to update
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables to update

class spynnaker.pyNN.models.neuron.implementations.**AbstractNeuronImpl**

Bases: object

An abstraction of a whole neuron model including all parts

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

binary_name

The name of the binary executable of this implementation

:rtype str

get_data (*parameters, state_variables, vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcn_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters **n_neurons** (*int*) – The number of neurons to get the usage for

Return type int

get_global_weight_scale ()

Get the weight scaling required by this model

Return type int

get_n_cpu_cycles (*n_neurons*)
 Get the number of CPU cycles required to update the state
Parameters *n_neurons* (*int*) – The number of neurons to get the cycles for
Return type *int*

get_n_synapse_types ()
 Get the number of synapse types supported by the model
Return type *int*

get_recordable_units (*variable*)
 Get the units of the given variable that can be recorded
Parameters *variable* (*str*) – The name of the variable

get_recordable_variable_index (*variable*)
 Get the index of the variable in the list of variables that can be recorded
Parameters *variable* (*str*) – The name of the variable
Return type *int*

get_recordable_variables ()
 Get the names of the variables that can be recorded in this model
Return type *list of str*

get_sdram_usage_in_bytes (*n_neurons*)
 Get the SDRAM memory usage required
Parameters *n_neurons* (*int*) – The number of neurons to get the usage for
Return type *int*

get_synapse_id_by_target (*target*)
 Get the id of a synapse given the name
Parameters *target* (*str*) – The name of the synapse
Return type *int*

get_synapse_targets ()
 Get the target names of the synapse type
Return type *array of str*

get_units (*variable*)
 Get the units of the given variable
Parameters *variable* (*str*) – The name of the variable

is_conductance_based
 Determine if the model uses conductance
Return type *bool*

is_recordable (*variable*)
 Determine if the given variable can be recorded
Parameters *variable* (*str*) – The name of the variable being requested
Return type *bool*

model_name
 The name of the model

Return type str

read_data (*data, offset, vertex_slice, parameters, state_variables*)

Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters to update
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables to update

class spynnaker.pyNN.models.neuron.implementations.**RangedDictVertexSlice** (*ranged_dict, vertex_slice*)

Bases: object

A slice of a ranged dict to be used to update values

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** (*data_types*)
Bases: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents a possible input type for a neuron model (e.g., current).

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

get_global_weight_scale ()
Get the global weight scaling value.

Returns The global weight scaling value

Return type float

spynnaker.pyNN.models.neuron.input_types.input_type_conductance module

class spynnaker.pyNN.models.neuron.input_types.input_type_conductance.**InputTypeConductance**
Bases: *spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType*

The conductance input type

add_parameters (*parameters*)
Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)
Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

e_rev_E

e_rev_I

get_global_weight_scale ()
Get the global weight scaling value.

Returns The global weight scaling value

Return type float

get_n_cpu_cycles (*n_neurons*)
Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)
Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)
Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)
Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

update_values (*values, parameters, state_variables*)
Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

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*](#)

The current input type

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_global_weight_scale ()

Get the global weight scaling value.

Returns The global weight scaling value

Return type float

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

spynnaker.pyNN.models.neuron.input_types.input_type_current_semd module

class spynnaker.pyNN.models.neuron.input_types.input_type_current_semd.**InputTypeCurrentSEMD**

Bases: `spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType`

The current sEMD input type

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

get_global_weight_scale ()

Get the global weight scaling value.

Returns The global weight scaling value

Return type float

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the parameters
- **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as `self.struct.field_types`

Return type A list of (single value or list of values or `RangedList`)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

inh_input_previous

multiplicator

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

Module contents

class spynnaker.pyNN.models.neuron.input_types.**AbstractInputType** (*data_types*)

Bases: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents a possible input type for a neuron model (e.g., current).

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

get_global_weight_scale ()

Get the global weight scaling value.

Returns The global weight scaling value

Return type float

class spynnaker.pyNN.models.neuron.input_types.**InputTypeConductance** (*e_rev_E, e_rev_I*)

Bases: *spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType*

The conductance input type

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the state variables

e_rev_E

e_rev_I

get_global_weight_scale ()

Get the global weight scaling value.

Returns The global weight scaling value

Return type float

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters
- **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

class spynnaker.pyNN.models.neuron.input_types.**InputTypeCurrent**

Bases: *spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType*

The current input type

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.RangeDictionary) – A holder of the state variables

get_global_weight_scale()
Get the global weight scaling value.

Returns The global weight scaling value

Return type float

get_n_cpu_cycles(*n_neurons*)
Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units(*variable*)
Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values(*parameters, state_variables, vertex_slice*)
Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable(*variable*)
Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

update_values(*values, parameters, state_variables*)
Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

class spynnaker.pyNN.models.neuron.input_types.**InputTypeCurrentSEMD**(*multiplicator, inh_input_previous*)

Bases: [*spynnaker.pyNN.models.neuron.input_types.abstract_input_type.AbstractInputType*](#)

The current sEMD input type

add_parameters(*parameters*)
Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)
Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_global_weight_scale ()
Get the global weight scaling value.

Returns The global weight scaling value

Return type float

get_n_cpu_cycles (*n_neurons*)
Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)
Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)
Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)
Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

inh_input_previous

multiplicator

update_values (*values, parameters, state_variables*)
Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

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_factory
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`) – the transceiver object
- **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*)

Complete the master pop table in the data specification.

Parameters

- **spec** – the data specification 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.

Returns a list of constraints

Return type list(`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, key_and_mask, master_pop_table_region, is_single=False*)

Update a data specification with a master pop entry in some form

Parameters

- **spec** – the data specification 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
- **key_and_mask** (`pacman.model.routing_info.BaseKeyAndMask`) – a key_and_mask object used as part of describing an edge that will require being received to be stored in the master pop table; the whole edge will become multiple calls to this function

- **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_binary_search module

class spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.
 Bases: [*spynnaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory.AbstractMasterPopTableFactory*](#)

Master population table, implemented as binary search master.

ADDRESS_LIST_DTYPE = '<u4'

ADDRESS_MASK = 2147483392

ADDRESS_SCALE = 16

ADDRESS_SCALED_SHIFT = 4

MASTER_POP_ENTRY_DTYPE = [('key', '<u4'), ('mask', '<u4'), ('start', '<u2'), ('count',

ROW_LENGTH_MASK = 255

SINGLE_BIT_FLAG_BIT = 2147483648

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*](#)) – the transceiver object
- **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*)

Complete the master pop table in the data specification.

Parameters

- **spec** – the data specification to write the master pop entry to
- **master_pop_table_region** – the region to which the master pop table is being stored

get_allowed_row_length(*row_length*)

Parameters *row_length* – the row length being considered

Returns the row length available

get_edge_constraints()

Gets the constraints for this table on edges coming in to a vertex.

Returns a list of constraints

Return type `list(pacman.model.constraints.AbstractConstraint)`

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*)

Get the size of the master population table in SDRAM

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 ()

Initialise the master pop data structure.

Return type `None`

update_master_population_table (*spec, block_start_addr, row_length, key_and_mask, master_pop_table_region, is_single=False*)

Add an 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.

Returns The index of the entry, to be used to retrieve it

Return type `int`

Module contents

```
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`

Master population table, implemented as binary search master.

```
ADDRESS_LIST_DTYPE = '<u4'
```

```
ADDRESS_MASK = 2147483392
```

```
ADDRESS_SCALE = 16
```

```
ADDRESS_SCALED_SHIFT = 4
```

```
MASTER_POP_ENTRY_DTYPE = [('key', '<u4'), ('mask', '<u4'), ('start', '<u2'), ('count',
```


ROW_LENGTH_MASK = 255

SINGLE_BIT_FLAG_BIT = 2147483648

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`) – the transceiver object
- **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*)

Complete the master pop table in the data specification.

Parameters

- **spec** – the data specification to write the master pop entry to
- **master_pop_table_region** – the region to which the master pop table is being stored

get_allowed_row_length (*row_length*)

Parameters **row_length** – the row length being considered

Returns the row length available

get_edge_constraints ()

Gets the constraints for this table on edges coming in to a vertex.

Returns a list of constraints

Return type `list(pacman.model.constraints.AbstractConstraint)`

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*)

Get the size of the master population table in SDRAM

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 ()

Initialise the master pop data structure.

Return type `None`

update_master_population_table (*spec, block_start_addr, row_length, key_and_mask, master_pop_table_region, is_single=False*)

Add an 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.

Returns The index of the entry, to be used to retrieve it

Return type int

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: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents a neuron model.

Parameters

- **data_types** – A list of data types in the neuron structure, in the order that they appear
- **global_data_types** – A list of data types in the neuron global structure, in the order that they appear

get_data (*parameters, state_variables, vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters
- **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcn_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters **n_neurons** (*int*) – The number of neurons to get the usage for

Return type int

get_global_values()

Get the global values to be written to the machine for this model

Returns A list with the same length as `self.global_struct.field_types`

Return type A list of single values

get_sdram_usage_in_bytes(*n_neurons*)

Get the SDRAM memory usage required

Parameters *n_neurons* (*int*) – The number of neurons to get the usage for

Return type *int*

global_struct

Get the global parameters structure

read_data(*data, offset, vertex_slice, parameters, state_variables*)

Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the parameters to update
- **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the state variables to update

Returns The offset after reading the data

spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh module

```
class spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh(a,
                                                                                   b,
                                                                                   c,
                                                                                   d,
                                                                                   v_init,
                                                                                   u_init,
                                                                                   i_offset)
```

Bases: `spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel`

a

add_parameters(*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters *parameters* (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables(*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters *state_variables* (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

b

c

d

get_global_values (*machine_time_step*)

Get the global values to be written to the machine for this model

Returns A list with the same length as self.global_struct.field_types

Return type A list of single values

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

i_offset

u_init

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

v_init

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.**Neu**

Bases: `spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel`

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

cm

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type *int*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

i_offset

tau_m

tau_refrac

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

v_init

v_reset

v_rest

Module contents

class spynnaker.pyNN.models.neuron.neuron_models.**AbstractNeuronModel** (*data_types, global_data_types=None*)
Bases: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents a neuron model.

Parameters

- **data_types** – A list of data types in the neuron structure, in the order that they appear
- **global_data_types** – A list of data types in the neuron global structure, in the order that they appear

get_data (*parameters, state_variables, vertex_slice*)

Get the data to be written to the machine for this model

Parameters

- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters
- **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the state variables
- **vertex_slice** – The slice of the vertex to generate parameters for

Return type numpy array of uint32

get_dtcn_usage_in_bytes (*n_neurons*)

Get the DTCM memory usage required

Parameters **n_neurons** (*int*) – The number of neurons to get the usage for

Return type int

get_global_values ()

Get the global values to be written to the machine for this model

Returns A list with the same length as self.global_struct.field_types

Return type A list of single values

get_sdram_usage_in_bytes (*n_neurons*)

Get the SDRAM memory usage required

Parameters `n_neurons` (*int*) – The number of neurons to get the usage for

Return type `int`

global_struct

Get the global parameters structure

read_data (*data, offset, vertex_slice, parameters, state_variables*)

Read the parameters and state variables of the model from the given data

Parameters

- **data** – The data to be read
- **offset** – The offset where the data should be read from
- **vertex_slice** – The slice of the vertex to read parameters for
- **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the parameters to update
- **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the state variables to update

Returns The offset after reading the data

```
class spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh(a, b, c, d,  
                                                                v_init, u_init,  
                                                                i_offset)
```

Bases: `spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel`

a

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters `parameters` (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters `state_variables` (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

b

c

d

get_global_values (*machine_time_step*)

Get the global values to be written to the machine for this model

Returns A list with the same length as `self.global_struct.field_types`

Return type A list of single values

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters `n_neurons` (*int*) – The number of neurons to get the cycles for

Return type `int`

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the parameters
- **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as `self.struct.field_types`

Return type A list of (single value or list of values or `RangedList`)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type `bool`

i_offset

u_init

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

v_init

```
class spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire (v_init,
                                                                                       v_rest,
                                                                                       tau_m,
                                                                                       cm,
                                                                                       i_offset,
                                                                                       v_reset,
                                                                                       tau_refrac)
```

Bases: `spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel`

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

cm

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type *int*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

i_offset

tau_m

tau_refrac

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

v_init

v_reset

v_rest

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_length)

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers.**get_exp_lut_array**(time_steps, shift, fixed_length, size)

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers.**get_exp_lut_values**(time_steps, shift, fixed_length, size)

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_structure
Bases: object

get_n_half_words_per_connection()
Get the number of bytes for each connection

get_weight_half_word()
The index of the half-word where the weight should be written

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_accumulator

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_half_words_per_connection()
Get the number of bytes for each connection

get_weight_half_word()
The index of the half-word where the weight should be written

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_only

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_half_words_per_connection()
Get the number of bytes for each connection

get_weight_half_word()
The index of the half-word where the weight should be written

Module contents

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.**AbstractSynapseStructure**

Bases: object

get_n_half_words_per_connection()
Get the number of bytes for each connection

get_weight_half_word()
The index of the half-word where the weight should be written

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.**SynapseStructureWeightOnly**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_half_words_per_connection()
Get the number of bytes for each connection

get_weight_half_word()
The index of the half-word where the weight should be written

class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.**SynapseStructureWeightAccumulator**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure.AbstractSynapseStructure*

get_n_half_words_per_connection()
Get the number of bytes for each connection

get_weight_half_word()
The index of the half-word where the weight should be written

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_dependence.  
    Bases: object  
  
    get_parameter_names ()  
        Return the names of the parameters supported by this timing dependency model.  
  
        Return type iterable(str)  
  
    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_pfister_spike_triplet.  
  
    Bases: spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence  
  
    get_parameter_names ()  
        Return the names of the parameters supported by this timing dependency model.  
  
        Return type iterable(str)  
  
    get_parameters_sdram_usage_in_bytes ()  
        Get the amount of SDRAM used by the parameters of this rule  
  
    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

tau_minus**tau_plus****tau_x****tau_y****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_recurrent module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent

Bases: `spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence`

default_parameters = {'accumulator_depression': -6, 'accumulator_potentiation': 6, '...

get_parameter_names ()

Return the names of the parameters supported by this timing dependency model.

Return type iterable(str)

get_parameters_sdram_usage_in_bytes ()

Get the amount of SDRAM used by the parameters of this rule

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_spike_nearest_pair
module

```
class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair
```

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 ()

Return the names of the parameters supported by this timing dependency model.

Return type iterable(str)

get_parameters_sdram_usage_in_bytes ()

Get the amount of SDRAM used by the parameters of this rule

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

tau_minus

tau_plus

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_spike_pair
module

```
class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair
```

Bases: `spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence`

get_parameter_names ()

Return the names of the parameters supported by this timing dependency model.

Return type iterable(str)

get_parameters_sdram_usage_in_bytes()
Get the amount of SDRAM used by the parameters of this rule

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

tau_minus

tau_plus

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_vogels_2011 module

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels_2011

Bases: `spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence`

alpha

default_parameters = {'tau': 20.0}

get_parameter_names()
Return the names of the parameters supported by this timing dependency model.
Return type iterable(str)

get_parameters_sdram_usage_in_bytes()
Get the amount of SDRAM used by the parameters of this rule

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

tau

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

Module contents

```
class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence
    Bases: object

    get_parameter_names ()
        Return the names of the parameters supported by this timing dependency model.

        Return type: iterable(str)

    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.TimingDependenceSpike

    Bases: spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.  
abstract_timing_dependence.AbstractTimingDependence

    get_parameter_names ()
        Return the names of the parameters supported by this timing dependency model.

        Return type: iterable(str)

    get_parameters_sdram_usage_in_bytes ()
        Get the amount of SDRAM used by the parameters of this rule

    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

    tau_minus
```


get_parameter_names()

Return the names of the parameters supported by this timing dependency model.

Return type iterable(str)

get_parameters_sdram_usage_in_bytes()

Get the amount of SDRAM used by the parameters of this rule

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.**TimingDependenceSpike**

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()

Return the names of the parameters supported by this timing dependency model.

Return type iterable(str)

get_parameters_sdram_usage_in_bytes()

Get the amount of SDRAM used by the parameters of this rule

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

tau_minus

tau_plus

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.**TimingDependenceVogel1**

Bases: `spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence.AbstractTimingDependence`

alpha

default_parameters = {'tau': 20.0}

get_parameter_names()

Return the names of the parameters supported by this timing dependency model.

Return type iterable(str)

get_parameters_sdram_usage_in_bytes()

Get the amount of SDRAM used by the parameters of this rule

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

tau

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.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_minus

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_dependence

Bases: object

get_parameter_names ()

Returns the parameter names

Return type iterable(str)

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.
- **post_population_label** – label of post.

Returns 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_additive

Bases: `spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus`, `spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence`

get_parameter_names ()

Returns the parameter names

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_synapse_types*, *n_weight_terms*)

Get the amount of SDRAM used by the parameters of this rule

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

w_max

w_min

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_triplet module

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus, spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*

A3_minus

A3_plus

default_parameters = {'A3_minus': 0.01, 'A3_plus': 0.01, 'w_max': 1.0, 'w_min': 0.0}

get_parameter_names ()
 Returns the parameter names

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_synapse_types, n_weight_terms*)
 Get the amount of SDRAM used by the parameters of this rule

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

w_max

w_min

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_multiplicative module

class spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multiplicative

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus, spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*

get_parameter_names ()
 Returns the parameter names

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_synapse_types, n_weight_terms*)

Get the amount of SDRAM used by the parameters of this rule

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

w_max

w_min

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

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.**AbstractWeightDependency**

Bases: object

get_parameter_names ()

Returns the parameter names

Return type iterable(str)

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.
- **post_population_label** – label of post.

Returns 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_has_a_plus_a_minus.AbstractHasAPlusAMinus`, `spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence`

get_parameter_names ()

Returns the parameter names

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_synapse_types*, *n_weight_terms*)

Get the amount of SDRAM used by the parameters of this rule

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

w_max

w_min

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.**WeightDependenceMultiplicative**

Bases: `spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus`, `spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence`

get_parameter_names ()

Returns the parameter names

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_synapse_types*, *n_weight_terms*)

Get the amount of SDRAM used by the parameters of this rule

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

w_max

w_min

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.**WeightDependenceAddit**

Bases: *spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus, spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence*

A3_minus

A3_plus

default_parameters = {'A3_minus': 0.01, 'A3_plus': 0.01, 'w_max': 1.0, 'w_min': 0.0}

get_parameter_names()

Returns the parameter names

Return type iterable(str)

get_parameters_sdram_usage_in_bytes(*n_synapse_types, n_weight_terms*)

Get the amount of SDRAM used by the parameters of this rule

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

w_max

w_min

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

Module contents

Module contents

spynnaker.pyNN.models.neuron.structural_plasticity package

Subpackages

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis package

Subpackages

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination package

Submodules

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.abstract_elimination module

```
class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.abstract_elimination.AbstractElimination
    Bases: object

    An elimination rule

    get_parameter_names ()
        Return the names of the parameters supported by this rule

        Return type iterable(str)

    get_parameters_sdram_usage_in_bytes ()
        Get the amount of SDRAM used by the parameters of this rule

    vertex_executable_suffix
        The suffix to be appended to the vertex executable for this rule

    write_parameters (spec, weight_scale)
        Write the parameters of the rule to the spec
```

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.random_by_weight_elimination module

```
class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.random_by_weight_elimination.RandomByWeightElimination
```

Bases: `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.abstract_elimination.AbstractElimination`

Elimination Rule that depends on the weight of a synapse

Parameters

- **threshold** – Below this weight is considered depression, above or equal to this weight is considered potentiation (or the static weight of the connection on static weight connections)
- **prob_elim_depressed** – The probability of elimination if the weight has been depressed (ignored on static weight connections)
- **prob_elim_potentiatiated** – The probability of elimination of the weight has been potentiated or has not changed (and also used on static weight connections)

```
get_parameter_names ()
    Return the names of the parameters supported by this rule

    Return type iterable(str)

get_parameters_sdram_usage_in_bytes ()
    Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix
    The suffix to be appended to the vertex executable for this rule

write_parameters (spec, weight_scale)
    Write the parameters of the rule to the spec
```

Module contents

class `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.AbstractElimination`

Bases: `object`

An elimination rule

get_parameter_names ()

Return the names of the parameters supported by this rule

Return type `iterable(str)`

get_parameters_sdram_usage_in_bytes ()

Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix

The suffix to be appended to the vertex executable for this rule

write_parameters (*spec, weight_scale*)

Write the parameters of the rule to the spec

class `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomElimination`

Bases: `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.abstract_elimination.AbstractElimination`

Elimination Rule that depends on the weight of a synapse

Parameters

- **threshold** – Below this weight is considered depression, above or equal to this weight is considered potentiation (or the static weight of the connection on static weight connections)
- **prob_elim_depressed** – The probability of elimination if the weight has been depressed (ignored on static weight connections)
- **prob_elim_potentiated** – The probability of elimination if the weight has been potentiated or has not changed (and also used on static weight connections)

get_parameter_names ()

Return the names of the parameters supported by this rule

Return type `iterable(str)`

get_parameters_sdram_usage_in_bytes ()

Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix

The suffix to be appended to the vertex executable for this rule

write_parameters (*spec, weight_scale*)

Write the parameters of the rule to the spec

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation package

Submodules

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.abstract_formation module

```
class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.abstract_formation
    Bases: object

    A formation rule

    get_parameter_names()
        Return the names of the parameters supported by this rule

        Return type iterable(str)

    get_parameters_sdram_usage_in_bytes()
        Get the amount of SDRAM used by the parameters of this rule

    vertex_executable_suffix
        The suffix to be appended to the vertex executable for this rule

    write_parameters(spec)
        Write the parameters of the rule to the spec
```

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.distance_dependent_formation module

```
class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.distance_dependent_formation
```

Bases: `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.abstract_formation.AbstractFormation`

Formation rule that depends on the physical distance between neurons

Parameters

- **grid** – (x, y) dimensions of the grid of distance
- **p_form_forward** – The peak probability of formation on feed-forward connections
- **sigma_form_forward** – The spread of probability with distance of formation on feed-forward connections
- **p_form_lateral** – The peak probability of formation on lateral connections
- **sigma_form_lateral** – The spread of probability with distance of formation on lateral connections

distance (x0, x1, metric)

Compute the distance between points x0 and x1 place on the grid using periodic boundary conditions.

Parameters

- **x0** (`np.ndarray of ints`) – first point in space
- **x1** (`np.ndarray of ints`) – second point in space
- **grid** (`np.ndarray of ints`) – shape of grid
- **metric** (`str`) – distance metric, i.e. euclidian or manhattan

Returns the distance

Return type float

generate_distance_probability_array (*probability*, *sigma*)

Generate the exponentially decaying probability LUTs.

Parameters

- **probability** (*float*) – peak probability
- **sigma** (*float*) – spread

Returns distance-dependent probabilities

Return type numpy.ndarray(float)

get_parameter_names ()

Return the names of the parameters supported by this rule

Return type iterable(str)

get_parameters_sdram_usage_in_bytes ()

Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix

The suffix to be appended to the vertex executable for this rule

write_parameters (*spec*)

Write the parameters of the rule to the spec

Module contents

class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.**AbstractFormationRule**

Bases: object

A formation rule

get_parameter_names ()

Return the names of the parameters supported by this rule

Return type iterable(str)

get_parameters_sdram_usage_in_bytes ()

Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix

The suffix to be appended to the vertex executable for this rule

write_parameters (*spec*)

Write the parameters of the rule to the spec

class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.**DistanceDependentFormationRule**

Bases: `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation.abstract_formation.AbstractFormationRule`

Formation rule that depends on the physical distance between neurons

Parameters

- **grid** – (x, y) dimensions of the grid of distance
- **p_form_forward** – The peak probability of formation on feed-forward connections
- **sigma_form_forward** – The spread of probability with distance of formation on feed-forward connections
- **p_form_lateral** – The peak probability of formation on lateral connections
- **sigma_form_lateral** – The spread of probability with distance of formation on lateral connections

distance (*x0, x1, metric*)

Compute the distance between points x0 and x1 place on the grid using periodic boundary conditions.

Parameters

- **x0** (*np.ndarray of ints*) – first point in space
- **x1** (*np.ndarray of ints*) – second point in space
- **grid** (*np.ndarray of ints*) – shape of grid
- **metric** (*str*) – distance metric, i.e. euclidian or manhattan

Returns the distance

Return type float

generate_distance_probability_array (*probability, sigma*)

Generate the exponentially decaying probability LUTs.

Parameters

- **probability** (*float*) – peak probability
- **sigma** (*float*) – spread

Returns distance-dependent probabilities

Return type numpy.ndarray(float)

get_parameter_names ()

Return the names of the parameters supported by this rule

Return type iterable(str)

get_parameters_sdram_usage_in_bytes ()

Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix

The suffix to be appended to the vertex executable for this rule

write_parameters (*spec*)

Write the parameters of the rule to the spec

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection package

Submodules

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.abstract_partner_selection module

```
class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.  
    Bases: object  
    A partner selection rule  
  
    get_parameter_names ()  
        Return the names of the parameters supported by this rule  
        Return type iterable(str)  
  
    get_parameters_sdram_usage_in_bytes ()  
        Get the amount of SDRAM used by the parameters of this rule  
  
    vertex_executable_suffix  
        The suffix to be appended to the vertex executable for this rule  
  
    write_parameters (spec)  
        Write the parameters of the rule to the spec
```

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.last_neuron_selection module

```
class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.  
    Bases: spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.  
    partner_selection.abstract_partner_selection.AbstractPartnerSelection  
    Partner selection that picks a random source neuron from the neurons that spiked in the last timestep  
    Parameters spike_buffer_size – The size of the buffer for holding spikes  
  
    get_parameter_names ()  
        Return the names of the parameters supported by this rule  
        Return type iterable(str)  
  
    get_parameters_sdram_usage_in_bytes ()  
        Get the amount of SDRAM used by the parameters of this rule  
  
    vertex_executable_suffix  
        The suffix to be appended to the vertex executable for this rule  
  
    write_parameters (spec)  
        Write the parameters of the rule to the spec
```

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.random_selection module

```
class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.  
    Bases: spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.  
    partner_selection.abstract_partner_selection.AbstractPartnerSelection  
    Partner selection that picks a random source neuron from all sources  
  
    get_parameter_names ()  
        Return the names of the parameters supported by this rule  
        Return type iterable(str)
```

get_parameters_sdram_usage_in_bytes()
Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix
The suffix to be appended to the vertex executable for this rule

write_parameters(spec)
Write the parameters of the rule to the spec

Module contents

class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.**PartnerSelection**
Bases: object
A partner selection rule

get_parameter_names()
Return the names of the parameters supported by this rule
Return type iterable(str)

get_parameters_sdram_usage_in_bytes()
Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix
The suffix to be appended to the vertex executable for this rule

write_parameters(spec)
Write the parameters of the rule to the spec

class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.**RandomPartnerSelection**
Bases: `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.abstract_partner_selection.AbstractPartnerSelection`
Partner selection that picks a random source neuron from the neurons that spiked in the last timestep
Parameters **spike_buffer_size** – The size of the buffer for holding spikes

get_parameter_names()
Return the names of the parameters supported by this rule
Return type iterable(str)

get_parameters_sdram_usage_in_bytes()
Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix
The suffix to be appended to the vertex executable for this rule

write_parameters(spec)
Write the parameters of the rule to the spec

class spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.**UniformPartnerSelection**
Bases: `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.abstract_partner_selection.AbstractPartnerSelection`
Partner selection that picks a random source neuron from all sources

get_parameter_names()
Return the names of the parameters supported by this rule
Return type iterable(str)

get_parameters_sdram_usage_in_bytes()
 Get the amount of SDRAM used by the parameters of this rule

vertex_executable_suffix
 The suffix to be appended to the vertex executable for this rule

write_parameters(spec)
 Write the parameters of the rule to the spec

Module contents

Module contents

spynnaker.pyNN.models.neuron.synapse_dynamics package

Submodules

spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine module

class spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine.**AbstractGenerateOnMachine**
 Bases: object

A synapse dynamics that can be generated on the machine

gen_matrix_id
 The ID of the on-machine matrix generator

Return type int

gen_matrix_params
 Any parameters required by the matrix generator

Return type numpy array of uint32

gen_matrix_params_size_in_bytes
 The size of the parameters of the matrix generator in bytes

Return type int

generate_on_machine()
 Determines if this instance should be generated on the machine.

Default implementation returns True

Return type bool

class spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine.**MatrixGenerator**
 Bases: enum.Enum

An enumeration.

STATIC_MATRIX = 0

STDP_MATRIX = 1

spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics module

class spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.**AbstractPlasticSynapseDynamics**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics*

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.**AbstractStaticSynapseDynamics**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics*

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.**AbstractSynapseDynamics**

Bases: object

NUMPY_CONNECTORS_DTYPE = [('source', 'uint32'), ('target', 'uint32'), ('weight', 'float64')]

are_weights_signed ()

Determines if the weights are signed values

changes_during_run

Determine if the synapses change during a run

Return type bool

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

delay

The delay of connections

get_delay_maximum (*connector, synapse_info*)

Get the maximum delay for the synapses

get_delay_variance (*connector, delays*)

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 ()

Get the parameter names available from the synapse dynamics components

Return type iterable(str)

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, synapse_info*)

Get the maximum weight for the synapses

get_weight_mean (*connector, synapse_info*)

Get the mean weight for the synapses

get_weight_variance (*connector, weights*)
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

merge (*synapse_dynamics*)
Merge with the given synapse_dynamics and return the result, or error if merge is not possible

set_delay (*delay*)
Set the delay

weight
The weight of connections

write_parameters (*spec, region, machine_time_step, weight_scales*)
Write the synapse parameters to the spec

spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural module

class spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.**AbstractSynapseDynamicsStructural**
Bases: object

elimination
The elimination rule

f_rew
The frequency of rewiring

formation
The formation rule

get_structural_parameters_sdram_usage_in_bytes (*application_graph, app_vertex, n_neurons, n_synapse_types*)
Get the size of the structural parameters

initial_delay
The delay of a formed connection

initial_weight
The weight of a formed connection

partner_selection
The partner selection rule

s_max
The maximum number of synapses

seed
The seed to control the randomness

set_connections (*connections, post_vertex_slice, app_edge, synapse_info, machine_edge*)
Set connections for structural plasticity

write_structural_parameters (*spec, region, machine_time_step, weight_scales, application_graph, app_vertex, post_slice, graph_mapper, routing_info, synapse_indices*)
Write structural plasticity parameters

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.**SynapseDynamicsStatic**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamics, spynnaker.pyNN.models.abstract_models.abstract_settable.AbstractSettable, spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun, spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine.AbstractGenerateOnMachine*

are_weights_signed()

Determines if the weights are signed values

changes_during_run

Determine if the synapses change during a run

Return type bool**delay**

The delay of connections

gen_matrix_id

The ID of the on-machine matrix generator

Return type int**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 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_parameter_names** ()

Get the parameter names available from the synapse dynamics components

Return type iterable(str)**get_parameters_sdram_usage_in_bytes** (*n_neurons, n_synapse_types*)

Get the SDRAM usage of the synapse dynamics parameters in bytes

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.

get_value (*key*)

Get a property Get a property

get_vertex_executable_suffix ()

Get the executable suffix for a vertex for this dynamics

is_same_as (*synapse_dynamics*)

Determines if this synapse dynamics is the same as another

mark_no_changes ()

Marks the point after which changes are reported, so that new changes can be detected before the next check. Marks the point after which changes are reported. Immediately after calling this method, *requires_mapping* should return False.

merge (*synapse_dynamics*)

Merge with the given *synapse_dynamics* and return the result, or error if merge is not possible

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*

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool 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_delay (*delay*)

Set the delay

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

weight

The weight of connections

write_parameters (*spec, region, machine_time_step, weight_scales*)

Write the synapse parameters to the spec

spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp module**class** spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.**SynapseDynamicsS**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.
abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics,
spynnaker.pyNN.models.abstract_models.abstract_settable.AbstractSettable,
spinn_front_end_common.abstract_models.abstract_changable_after_run.
AbstractChangableAfterRun, spynnaker.pyNN.models.neuron.synapse_dynamics.
abstract_generate_on_machine.AbstractGenerateOnMachine*

are_weights_signed()

Determines if the weights are signed values

backprop_delay**changes_during_run**

Determine if the synapses change during a run

Return type bool**delay**

The delay of connections

dendritic_delay_fraction**gen_matrix_id**

The ID of the on-machine matrix generator

Return type int**gen_matrix_params**

Any parameters required by the matrix generator

Return type numpy array of uint32**gen_matrix_params_size_in_bytes**

The size of the parameters of the matrix generator in bytes

Return type int**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 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_parameter_names ()
 Get the parameter names available from the synapse dynamics components

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_neurons, n_synapse_types*)
 Get the SDRAM usage of the synapse dynamics parameters in bytes

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.

get_provenance_data (*pre_population_label, post_population_label*)
 Get the provenance data from this synapse dynamics object

get_value (*key*)
 Get a property Get a property

get_vertex_executable_suffix ()
 Get the executable suffix for a vertex for this dynamics

get_weight_maximum (*connector, synapse_info*)
 Get the maximum weight for the synapses

get_weight_mean (*connector, synapse_info*)
 Get the mean weight for the synapses

get_weight_variance (*connector, weights*)
 Get the variance in weight for the synapses

is_same_as (*synapse_dynamics*)
 Determines if this synapse dynamics is the same as another

mark_no_changes ()
 Marks the point after which changes are reported, so that new changes can be detected before the next check. Marks the point after which changes are reported. Immediately after calling this method, *requires_mapping* should return False.

merge (*synapse_dynamics*)
 Merge with the given *synapse_dynamics* and return the result, or error if merge is not possible

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*

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool 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_delay (*delay*)

Set the delay

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**

The weight of connections

weight_dependence**write_parameters** (*spec*, *region*, *machine_time_step*, *weight_scales*)

Write the synapse parameters to the spec

spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common module

class spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon

Bases: object

Utility class that holds properties of synaptic rewiring both in the presence and absence of STDP.

Written by Petrut Bogdan.

Parameters

- **partner_selection** – The partner selection rule

- **formation** – The formation rule
- **elimination** – The elimination rule
- **f_rew** (*int*) – How many rewiring attempts will be done per second.
- **initial_weight** (*float*) – Initial weight assigned to a newly formed connection
- **initial_delay** (*int* or (*int*, *int*)) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **seed** (*int*) – seed the random number generators

DEFAULT_F_REW = 10000

DEFAULT_INITIAL_DELAY = 1

DEFAULT_INITIAL_WEIGHT = 0

DEFAULT_S_MAX = 32

KEY_ATOM_INFO_SIZE = 20

POST_TO_PRE_ENTRY_SIZE = 4

PRE_POP_INFO_BASE_SIZE = 20

REWIRING_DATA_SIZE = 64

actual_sdram_usage

Actual SDRAM usage (based on what is written to spec).

Returns actual SDRAM usage

Return type int

elimination

f_rew

formation

get_parameter_names ()

get_parameters_sdram_usage_in_bytes (*application_graph*, *app_vertex*, *n_neurons*)

Get SDRAM usage

Returns SDRAM usage

Return type int

get_vertex_executable_suffix ()

initial_delay

initial_weight

is_same_as (*synapse_dynamics*)

n_words_for_plastic_connections (*value*)

Set size of plastic connections in words

n_words_for_static_connections (*value*)

Set size of static connections in words

p_rew

The period of rewiring.

Returns The period of rewiring

Return type int

partner_selection

s_max

seed

set_projection_parameter (*param, value*)

synaptic_data_update (*connections, post_vertex_slice, app_edge, synapse_info, machine_edge*)

Set synaptic data

write_parameters (*spec, region, machine_time_step, weight_scales, application_graph, app_vertex, post_slice, graph_mapper, routing_info, synapse_indices*)

Write the synapse parameters to the spec.

spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static module

class spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStatic

Bases: `spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStatic`, `spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural`

Class that enables synaptic rewiring in the absence of STDP.

It acts as a wrapper around SynapseDynamicsStatic, meaning that rewiring can operate in parallel with static synapses.

Written by Petrut Bogdan.

Parameters

- **partner_selection** – The partner selection rule
- **formation** – The formation rule
- **elimination** – The elimination rule
- **f_rew** (*int*) – How many rewiring attempts will be done per second.

- **initial_weight** (*float*) – Weight assigned to a newly formed connection
- **initial_delay** (*float or (float, float)*) – Delay assigned to a newly formed connection; a single value means a fixed delay value, or a tuple of two values means the delay will be chosen at random from a uniform distribution between the given values
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **seed** (*int*) – seed the random number generators
- **weight** – The weight of connections formed by the connector
- **delay** – The delay of connections formed by the connector

changes_during_run

Determine if the synapses change during a run

Return type bool

elimination

The elimination rule

f_rew

The frequency of rewiring

formation

The formation rule

get_n_words_for_static_connections (*n_connections*)

Get the number of 32-bit words for *n_connections* in a single row

get_parameter_names ()

Get the parameter names available from the synapse dynamics components

Return type iterable(str)

get_structural_parameters_sdram_usage_in_bytes (*application_graph, app_vertex, n_neurons, n_synapse_types*)

Get the size of the structural parameters

get_vertex_executable_suffix ()

Get the executable suffix for a vertex for this dynamics

get_weight_maximum (*connector, synapse_info*)

Get the maximum weight for the synapses

get_weight_mean (*connector, synapse_info*)

Get the mean weight for the synapses

get_weight_variance (*connector, weights*)

Get the variance in weight for the synapses

initial_delay

The delay of a formed connection

initial_weight

The weight of a formed connection

is_same_as (*synapse_dynamics*)

Determines if this synapse dynamics is the same as another

merge (*synapse_dynamics*)

Merge with the given *synapse_dynamics* and return the result, or error if merge is not possible

partner_selection

The partner selection rule

s_max

The maximum number of synapses

seed

The seed to control the randomness

set_connections (*connections, post_vertex_slice, app_edge, synapse_info, machine_edge*)

Set connections for structural plasticity

set_projection_parameter (*param, value*)**write_structural_parameters** (*spec, region, machine_time_step, weight_scales, application_graph, app_vertex, post_slice, graph_mapper, routing_info, synapse_indices*)

Write structural plasticity parameters

spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp module

class spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp.**SynapseDynamicsStructuralSTDP**

Bases: `spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP`, `spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural`

Class that enables synaptic rewiring in the presence of STDP.

It acts as a wrapper around SynapseDynamicsSTDP, meaning rewiring can operate in parallel with STDP synapses.

Written by Petrut Bogdan.

Parameters

- **partner_selection** – The partner selection rule
- **formation** – The formation rule
- **elimination** – The elimination rule
- **timing_dependence** – The STDP timing dependence
- **weight_dependence** – The STDP weight dependence
- **voltage_dependence** – The STDP voltage dependence
- **dendritic_delay_fraction** – The STDP dendritic delay fraction
- **f_rew** (*int*) – How many rewiring attempts will be done per second.
- **initial_weight** (*float*) – Weight assigned to a newly formed connection
- **initial_delay** (*float or (float, float)*) – Delay assigned to a newly formed connection; a single value means a fixed delay value, or a tuple of two values means the delay will be chosen at random from a uniform distribution between the given values
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **seed** (*int*) – seed the random number generators
- **weight** – The weight of connections formed by the connector
- **delay** – The delay of connections formed by the connector

elimination

The elimination rule

f_rew

The frequency of rewiring

formation

The formation rule

get_n_words_for_plastic_connections (*n_connections*)

Get the number of 32-bit words for *n_connections* in a single row

get_parameter_names ()

Get the parameter names available from the synapse dynamics components

Return type iterable(str)

get_structural_parameters_sdram_usage_in_bytes (*application_graph, app_vertex, n_neurons, n_synapse_types*)

Get the size of the structural parameters

get_vertex_executable_suffix ()

Get the executable suffix for a vertex for this dynamics

get_weight_maximum (*connector, synapse_info*)

Get the maximum weight for the synapses

get_weight_mean (*connector, synapse_info*)

Get the mean weight for the synapses

initial_delay

The delay of a formed connection

initial_weight
The weight of a formed connection

is_same_as (*synapse_dynamics*)
Determines if this synapse dynamics is the same as another

merge (*synapse_dynamics*)
Merge with the given synapse_dynamics and return the result, or error if merge is not possible

partner_selection
The partner selection rule

s_max
The maximum number of synapses

seed
The seed to control the randomness

set_connections (*connections, post_vertex_slice, app_edge, synapse_info, machine_edge*)
Set connections for structural plasticity

set_projection_parameter (*param, value*)

write_structural_parameters (*spec, region, machine_time_step, weight_scales, application_graph, app_vertex, post_slice, graph_mapper, routing_info, synapse_indices*)
Write structural plasticity parameters

Module contents

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractSynapseDynamics**
Bases: object

NUMPY_CONNECTORS_DTYPE = [('source', 'uint32'), ('target', 'uint32'), ('weight', 'float64')]

are_weights_signed()
Determines if the weights are signed values

changes_during_run
Determine if the synapses change during a run

Return type bool

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

delay
The delay of connections

get_delay_maximum (*connector, synapse_info*)
Get the maximum delay for the synapses

get_delay_variance (*connector, delays*)
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 ()
Get the parameter names available from the synapse dynamics components
Return type iterable(str)

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, synapse_info*)
Get the maximum weight for the synapses

get_weight_mean (*connector, synapse_info*)
Get the mean weight for the synapses

get_weight_variance (*connector, weights*)
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

merge (*synapse_dynamics*)
Merge with the given synapse_dynamics and return the result, or error if merge is not possible

set_delay (*delay*)
Set the delay

weight
The weight of connections

write_parameters (*spec, region, machine_time_step, weight_scales*)
Write the synapse parameters to the spec

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractGenerateOnMachine**
Bases: object
A synapse dynamics that can be generated on the machine

gen_matrix_id
The ID of the on-machine matrix generator
Return type int

gen_matrix_params
Any parameters required by the matrix generator
Return type numpy array of uint32

gen_matrix_params_size_in_bytes
The size of the parameters of the matrix generator in bytes
Return type int

generate_on_machine()

Determines if this instance should be generated on the machine.

Default implementation returns True

Return type bool

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractStaticSynapseDynamics**

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.
abstract_synapse_dynamics.AbstractSynapseDynamics*

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*

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** (*weight=0.0, delay=1.0, pad_to_length=None*)

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamics, spynnaker.pyNN.models.abstract_models.abstract_settable.AbstractSettable, spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun, spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine.AbstractGenerateOnMachine*

are_weights_signed ()

Determines if the weights are signed values

changes_during_run

Determine if the synapses change during a run

Return type bool

delay

The delay of connections

gen_matrix_id

The ID of the on-machine matrix generator

Return type int

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 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_parameter_names ()

Get the parameter names available from the synapse dynamics components

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_neurons, n_synapse_types*)

Get the SDRAM usage of the synapse dynamics parameters in bytes

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.

get_value (*key*)

Get a property Get a property

get_vertex_executable_suffix ()

Get the executable suffix for a vertex for this dynamics

is_same_as (*synapse_dynamics*)

Determines if this synapse dynamics is the same as another

mark_no_changes ()

Marks the point after which changes are reported, so that new changes can be detected before the next check. Marks the point after which changes are reported. Immediately after calling this method, `requires_mapping` should return False.

merge (*synapse_dynamics*)

Merge with the given `synapse_dynamics` and return the result, or error if merge is not possible

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`

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool 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_delay (*delay*)

Set the delay

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

weight

The weight of connections

write_parameters (*spec, region, machine_time_step, weight_scales*)

Write the synapse parameters to the spec

class spynnaker.pyNN.models.neuron.synapse_dynamics.**SynapseDynamicsSTDP** (*timing_dependence=None, weight_dependence=None, volt-age_dependence=None, dendritic_delay_fraction=1.0, weight=0.0, delay=1.0, pad_to_length=None, backprop_delay=True*)

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics.AbstractPlasticSynapseDynamics, spynnaker.pyNN.models.abstract_models.abstract_settable.AbstractSettable, spinn_front_end_common.abstract_models.abstract_changable_after_run.AbstractChangableAfterRun, spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine.AbstractGenerateOnMachine*

are_weights_signed()
Determines if the weights are signed values

backprop_delay

changes_during_run
Determine if the synapses change during a run
Return type bool

delay
The delay of connections

dendritic_delay_fraction

gen_matrix_id
The ID of the on-machine matrix generator
Return type int

gen_matrix_params
Any parameters required by the matrix generator
Return type numpy array of uint32

gen_matrix_params_size_in_bytes
The size of the parameters of the matrix generator in bytes
Return type int

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 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_parameter_names ()

Get the parameter names available from the synapse dynamics components

Return type iterable(str)

get_parameters_sdram_usage_in_bytes (*n_neurons, n_synapse_types*)

Get the SDRAM usage of the synapse dynamics parameters in bytes

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.

get_provenance_data (*pre_population_label, post_population_label*)

Get the provenance data from this synapse dynamics object

get_value (*key*)

Get a property Get a property

get_vertex_executable_suffix ()

Get the executable suffix for a vertex for this dynamics

get_weight_maximum (*connector, synapse_info*)

Get the maximum weight for the synapses

get_weight_mean (*connector, synapse_info*)

Get the mean weight for the synapses

get_weight_variance (*connector, weights*)

Get the variance in weight for the synapses

is_same_as (*synapse_dynamics*)

Determines if this synapse dynamics is the same as another

mark_no_changes ()

Marks the point after which changes are reported, so that new changes can be detected before the next check. Marks the point after which changes are reported. Immediately after calling this method, *requires_mapping* should return False.

merge (*synapse_dynamics*)

Merge with the given *synapse_dynamics* and return the result, or error if merge is not possible

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*

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool 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_delay (*delay*)

Set the delay

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

The weight of connections

weight_dependence

write_parameters (*spec, region, machine_time_step, weight_scales*)

Write the synapse parameters to the spec

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractSynapseDynamicsStructural**

Bases: object

elimination

The elimination rule

f_rew

The frequency of rewiring

formation

The formation rule

get_structural_parameters_sdram_usage_in_bytes (*application_graph, app_vertex, n_neurons, n_synapse_types*)

Get the size of the structural parameters

initial_delay

The delay of a formed connection

initial_weight

The weight of a formed connection

partner_selection

The partner selection rule

s_max

The maximum number of synapses

seed

The seed to control the randomness

set_connections (*connections, post_vertex_slice, app_edge, synapse_info, machine_edge*)

Set connections for structural plasticity

write_structural_parameters (*spec, region, machine_time_step, weight_scales, application_graph, app_vertex, post_slice, graph_mapper, routing_info, synapse_indices*)

Write structural plasticity parameters

class spynnaker.pyNN.models.neuron.synapse_dynamics.**SynapseDynamicsStructuralCommon** (*partner_selection, formation, elimination, initial_weight, initial_delay, s_max, seed*)

Bases: object

Utility class that holds properties of synaptic rewiring both in the presence and absence of STDP.

Written by Petrut Bogdan.

Parameters

- **partner_selection** – The partner selection rule
- **formation** – The formation rule
- **elimination** – The elimination rule
- **f_rew** (*int*) – How many rewiring attempts will be done per second.
- **initial_weight** (*float*) – Initial weight assigned to a newly formed connection
- **initial_delay** (*int* or (*int*, *int*)) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **seed** (*int*) – seed the random number generators

DEFAULT_F_REW = 10000

DEFAULT_INITIAL_DELAY = 1

DEFAULT_INITIAL_WEIGHT = 0

DEFAULT_S_MAX = 32

KEY_ATOM_INFO_SIZE = 20

POST_TO_PRE_ENTRY_SIZE = 4

PRE_POP_INFO_BASE_SIZE = 20

REWIRING_DATA_SIZE = 64

actual_sdram_usage

Actual SDRAM usage (based on what is written to spec).

Returns actual SDRAM usage

Return type int

elimination

f_rew

formation

get_parameter_names ()

get_parameters_sdram_usage_in_bytes (*application_graph, app_vertex, n_neurons*)

Get SDRAM usage

Returns SDRAM usage

Return type int

get_vertex_executable_suffix ()

initial_delay

initial_weight

is_same_as (*synapse_dynamics*)

n_words_for_plastic_connections (*value*)

Set size of plastic connections in words

n_words_for_static_connections (*value*)

Set size of static connections in words

p_rew

The period of rewiring.

Returns The period of rewiring

Return type int

partner_selection

s_max

seed

set_projection_parameter (*param, value*)

synaptic_data_update (*connections, post_vertex_slice, app_edge, synapse_info, machine_edge*)

Set synaptic data

write_parameters (*spec, region, machine_time_step, weight_scales, application_graph, app_vertex, post_slice, graph_mapper, routing_info, synapse_indices*)

Write the synapse parameters to the spec.

```
class spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic(partner_s
                                                                                   for-
                                                                                   ma-
                                                                                   tion,
                                                                                   elim-
                                                                                   i-
                                                                                   na-
                                                                                   tion,
                                                                                   f_rew=10
                                                                                   ini-
                                                                                   tial_weig
                                                                                   ini-
                                                                                   tial_delay
                                                                                   s_max=3
                                                                                   seed=None
                                                                                   weight=0
                                                                                   de-
                                                                                   lay=1.0)
```

Bases: `spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic`, `spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural`

Class that enables synaptic rewiring in the absence of STDP.

It acts as a wrapper around SynapseDynamicsStatic, meaning that rewiring can operate in parallel with static synapses.

Written by Petrut Bogdan.

Parameters

- **partner_selection** – The partner selection rule
- **formation** – The formation rule
- **elimination** – The elimination rule
- **f_rew** (*int*) – How many rewiring attempts will be done per second.
- **initial_weight** (*float*) – Weight assigned to a newly formed connection
- **initial_delay** (*float or (float, float)*) – Delay assigned to a newly formed connection; a single value means a fixed delay value, or a tuple of two values means the delay will be chosen at random from a uniform distribution between the given values
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **seed** (*int*) – seed the random number generators
- **weight** – The weight of connections formed by the connector
- **delay** – The delay of connections formed by the connector

changes_during_run

Determine if the synapses change during a run

Return type bool

elimination

The elimination rule

f_rew
The frequency of rewiring

formation
The formation rule

get_n_words_for_static_connections (*n_connections*)
Get the number of 32-bit words for *n_connections* in a single row

get_parameter_names ()
Get the parameter names available from the synapse dynamics components
Return type iterable(str)

get_structural_parameters_sdram_usage_in_bytes (*application_graph*, *app_vertex*, *n_neurons*, *n_synapse_types*)
Get the size of the structural parameters

get_vertex_executable_suffix ()
Get the executable suffix for a vertex for this dynamics

get_weight_maximum (*connector*, *synapse_info*)
Get the maximum weight for the synapses

get_weight_mean (*connector*, *synapse_info*)
Get the mean weight for the synapses

get_weight_variance (*connector*, *weights*)
Get the variance in weight for the synapses

initial_delay
The delay of a formed connection

initial_weight
The weight of a formed connection

is_same_as (*synapse_dynamics*)
Determines if this synapse dynamics is the same as another

merge (*synapse_dynamics*)
Merge with the given *synapse_dynamics* and return the result, or error if merge is not possible

partner_selection
The partner selection rule

s_max
The maximum number of synapses

seed
The seed to control the randomness

set_connections (*connections*, *post_vertex_slice*, *app_edge*, *synapse_info*, *machine_edge*)
Set connections for structural plasticity

set_projection_parameter (*param*, *value*)

write_structural_parameters (*spec*, *region*, *machine_time_step*, *weight_scales*, *application_graph*, *app_vertex*, *post_slice*, *graph_mapper*, *routing_info*, *synapse_indices*)
Write structural plasticity parameters

```
class spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralSTDP (partner_selection, formation, elimination, timing_dependence, weight_dependence, voltage_dependence, dendritic_delay_fraction, f_rew=1000, initial_weight=1.0, initial_delay=(1.0, 1.0), s_max=32, seed=None, weight=0.0, delay=1.0, backprop_delay=0.0)
```

Bases: `spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP`, `spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural`

Class that enables synaptic rewiring in the presence of STDP.

It acts as a wrapper around `SynapseDynamicsSTDP`, meaning rewiring can operate in parallel with STDP synapses.

Written by Petrut Bogdan.

Parameters

- **partner_selection** – The partner selection rule
- **formation** – The formation rule
- **elimination** – The elimination rule
- **timing_dependence** – The STDP timing dependence
- **weight_dependence** – The STDP weight dependence
- **voltage_dependence** – The STDP voltage dependence
- **dendritic_delay_fraction** – The STDP dendritic delay fraction
- **f_rew** (*int*) – How many rewiring attempts will be done per second.
- **initial_weight** (*float*) – Weight assigned to a newly formed connection
- **initial_delay** (*float or (float, float)*) – Delay assigned to a newly formed connection; a single value means a fixed delay value, or a tuple of two values means the delay will be chosen at random from a uniform distribution between the given values

- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **seed** (*int*) – seed the random number generators
- **weight** – The weight of connections formed by the connector
- **delay** – The delay of connections formed by the connector

elimination

The elimination rule

f_rew

The frequency of rewiring

formation

The formation rule

get_n_words_for_plastic_connections (*n_connections*)

Get the number of 32-bit words for *n_connections* in a single row

get_parameter_names ()

Get the parameter names available from the synapse dynamics components

Return type iterable(str)

get_structural_parameters_sdram_usage_in_bytes (*application_graph*, *app_vertex*, *n_neurons*, *n_synapse_types*)

Get the size of the structural parameters

get_vertex_executable_suffix ()

Get the executable suffix for a vertex for this dynamics

get_weight_maximum (*connector*, *synapse_info*)

Get the maximum weight for the synapses

get_weight_mean (*connector*, *synapse_info*)

Get the mean weight for the synapses

initial_delay

The delay of a formed connection

initial_weight

The weight of a formed connection

is_same_as (*synapse_dynamics*)

Determines if this synapse dynamics is the same as another

merge (*synapse_dynamics*)

Merge with the given *synapse_dynamics* and return the result, or error if merge is not possible

partner_selection

The partner selection rule

s_max

The maximum number of synapses

seed

The seed to control the randomness

set_connections (*connections*, *post_vertex_slice*, *app_edge*, *synapse_info*, *machine_edge*)

Set connections for structural plasticity

set_projection_parameter (*param*, *value*)

write_structural_parameters (*spec, region, machine_time_step, weight_scales, application_graph, app_vertex, post_slice, graph_mapper, routing_info, synapse_indices*)
Write structural plasticity parameters

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_max_row_info (*synapse_info, post_vertex_slice, n_delay_stages, population_table, machine_time_step, in_edge*)
Get the information about the maximum lengths of delayed and undelayed rows in bytes (including header), words (without header) and number of synapses

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_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, app_edge, machine_edge*)
Get the synapses as an array of words for non-delayed synapses and an array of words for delayed synapses

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*)
Read the synapses for a given projection synapse information object out of the given data

spynnaker.pyNN.models.neuron.synapse_io.max_row_info module

class spynnaker.pyNN.models.neuron.synapse_io.max_row_info.**MaxRowInfo** (*undelayed_max_n_synapses, delayed_max_n_synapses, undelayed_max_bytes, delayed_max_bytes, undelayed_max_words, delayed_max_words*)

Bases: object

Information about the maximums for rows in a synaptic matrix.

delayed_max_bytes

delayed_max_n_synapses

`delayed_max_words`
`undelayed_max_bytes`
`undelayed_max_n_synapses`
`undelayed_max_words`

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 the number of bytes in a block given the max row length and number of rows

get_max_row_info (*synapse_info, post_vertex_slice, n_delay_stages, population_table, machine_time_step, in_edge*)
 Get the information about the maximum lengths of delayed and undelayed rows in bytes (including header), words (without header) and number of synapses

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_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, app_edge, machine_edge*)
 Get the synapses as an array of words for non-delayed synapses and an array of words for delayed synapses

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_time_steps, machine_time_step*)
 Read the synapses for a given projection synapse information object out of the given data

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_max_row_info (*synapse_info, post_vertex_slice, n_delay_stages, population_table, machine_time_step, in_edge*)
 Get the information about the maximum lengths of delayed and undelayed rows in bytes (including header), words (without header) and number of synapses

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_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, app_edge, machine_edge*)

Get the synapses as an array of words for non-delayed synapses and an array of words for delayed synapses

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*)

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 the number of bytes in a block given the max row length and number of rows

get_max_row_info (*synapse_info, post_vertex_slice, n_delay_stages, population_table, machine_time_step, in_edge*)

Get the information about the maximum lengths of delayed and undelayed rows in bytes (including header), words (without header) and number of synapses

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_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, app_edge, machine_edge*)

Get the synapses as an array of words for non-delayed synapses and an array of words for delayed synapses

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_time_steps, machine_time_step*)

Read the synapses for a given projection synapse information object out of the given data

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: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents the synapse types supported.

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

get_n_synapse_types ()

Get the number of synapse types supported.

Returns The number of synapse types supported

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(str)

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_alpha module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_alpha.**SynapseTypeAlpha** (*exc_resp*, *exc_exp*, *tau_syn*, *inh_resp*, *inh_exp*, *tau_syn*)

Bases: `spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType`

add_parameters (*parameters*)
Add the initial values of the parameters to the parameter holder

Parameters *parameters* (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)
Add the initial values of the state variables to the state variables holder

Parameters *state_variables* (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

exc_response

get_n_cpu_cycles (*n_neurons*)
Get the number of CPU cycles required to update the state

Parameters *n_neurons* (*int*) – The number of neurons to get the cycles for

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_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(str)

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

inh_response

tau_syn_E

tau_syn_I

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.**SynapseTypeDelta** (*isyn_ext, isyn_inh*)

Bases: `spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType`

This represents a synapse type with two delta synapses

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

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_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(str)*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

isyn_exc

isyn_inh

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.**SynapseType**

Bases: `spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType`

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

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_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(str)`

get_units (*variable*)

Get the units of the given variable

Parameters *variable* (*str*) – The name of the variable

get_values (*parameters*, *state_variables*, *vertex_slice*, *ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters *variable* (*str*) – The name of the variable

Return type bool

isyn_exc

isyn_exc2

isyn_inh

tau_syn_E

tau_syn_E2

tau_syn_I

update_values (*values*, *parameters*, *state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential

Bases: `spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType`

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

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_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(str)*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

isyn_exc

isyn_inh

tau_syn_E

tau_syn_I

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

spynnaker.pyNN.models.neuron.synapse_types.synapse_type_semd module

class spynnaker.pyNN.models.neuron.synapse_types.synapse_type_semd.**SynapseTypeSEMD** (*tau_syn_E, tau_syn_E2, tau_syn_I, isyn_exc, isyn_exc2, isyn_inh, multiplier, exc2_old*)

Bases: *spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType*

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

exc2_old

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

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_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(str)

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

isyn_exc

isyn_exc2

isyn_inh

multiplicator

tau_syn_E

tau_syn_E2

tau_syn_I

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

Module contents

```

class spynnaker.pyNN.models.neuron.synapse_types.AbstractSynapseType (data_types)
    Bases: spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent
    Represents the synapse types supported.

        Parameters data_types – A list of data types in the component structure, in the order that they
            appear

get_n_synapse_types ()
    Get the number of synapse types supported.

        Returns The number of synapse types supported

        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(str)

class spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential (tau_syn_E,
                                                                                   tau_syn_E2,
                                                                                   tau_syn_I,
                                                                                   isyn_exc,
                                                                                   isyn_exc2,
                                                                                   isyn_inh)
    Bases: spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType

add_parameters (parameters)
    Add the initial values of the parameters to the parameter holder

        Parameters parameters (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (state_variables)
    Add the initial values of the state variables to the state variables holder

        Parameters state_variables (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_n_cpu_cycles (n_neurons)
    Get the number of CPU cycles required to update the state

        Parameters n_neurons (int) – The number of neurons to get the cycles for

        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_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(str)

get_units (*variable*)
Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)
Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)
Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

isyn_exc

isyn_exc2

isyn_inh

tau_syn_E

tau_syn_E2

tau_syn_I

update_values (*values, parameters, state_variables*)
Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

```

class spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential (tau_syn_E,
                                                                    tau_syn_I,
                                                                    isyn_exc,
                                                                    isyn_inh)
Bases:      spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.
            AbstractSynapseType

add_parameters (parameters)
    Add the initial values of the parameters to the parameter holder

    Parameters parameters      (spinn_utilities.ranged.range_dictionary.
                                RangeDictionary) – A holder of the parameters

add_state_variables (state_variables)
    Add the initial values of the state variables to the state variables holder

    Parameters state_variables      (spinn_utilities.ranged.
                                    range_dictionary.RangeDictionary) – A holder of the state variables

get_n_cpu_cycles (n_neurons)
    Get the number of CPU cycles required to update the state

    Parameters n_neurons (int) – The number of neurons to get the cycles for

    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_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(str)

get_units (variable)
    Get the units of the given variable

    Parameters variable (str) – The name of the variable

get_values (parameters, state_variables, vertex_slice, ts)
    Get the values to be written to the machine for this model

    Parameters
        • parameters      (spinn_utilities.ranged.range_dictionary.
                            RangeDictionary) – The holder of the parameters
        • state_variables  (spinn_utilities.ranged.range_dictionary.
                            RangeDictionary) – The holder of the state variables
        • vertex_slice – The slice of variables being retrieved

    Returns A list with the same length as self.struct.field_types

```

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

isyn_exc

isyn_inh

tau_syn_E

tau_syn_I

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

class spynnaker.pyNN.models.neuron.synapse_types.**SynapseTypeDelta** (*isyn_exc,*
isyn_inh)

Bases: `spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType`

This represents a synapse type with two delta synapses

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

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_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(str)

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

isyn_exc

isyn_inh

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

```
class spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha (exc_response,
                                                                    exc_exp_response,
                                                                    tau_syn_E,
                                                                    inh_response,
                                                                    inh_exp_response,
                                                                    tau_syn_I)
```

Bases: `spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType`

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the state variables

exc_response

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

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_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(str)*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters
- **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as *self.struct.field_types*

Return type A list of (single value or list of values or *RangedList*)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

inh_response

tau_syn_E

tau_syn_I

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

```
class spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeSEMD (tau_syn_E,
                                                                tau_syn_E2,
                                                                tau_syn_I,
                                                                isyn_exc,
                                                                isyn_exc2,
                                                                isyn_inh,
                                                                multiplicator,
                                                                exc2_old)
```

Bases: `spynnaker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType`

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (`spinn_utilities.ranged.range_dictionary.RangeDictionary`) – A holder of the state variables

exc2_old

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

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_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(str)`

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the parameters
- **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as `self.struct.field_types`

Return type A list of (single value or list of values or `RangedList`)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type `bool`

isyn_exc

isyn_exc2

isyn_inh

multiplicator

tau_syn_E

tau_syn_E2

tau_syn_I

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

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.AbstractThresholdType`

Bases: `spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent`

Represents types of threshold for a neuron (e.g., stochastic).

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic module

class spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic.Thresho

Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*

A stochastic threshold

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

du_th

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type *int*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type *bool*

tau_th

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

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*

A threshold that is a static value

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type *int*

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

v_thresh

Module contents

class spynnaker.pyNN.models.neuron.threshold_types.**AbstractThresholdType** (*data_types*)

Bases: *spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent*

Represents types of threshold for a neuron (e.g., stochastic).

Parameters **data_types** – A list of data types in the component structure, in the order that they appear

class spynnaker.pyNN.models.neuron.threshold_types.**ThresholdTypeStatic** (*v_thresh*)

Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*

A threshold that is a static value

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (*spinn_utilities.ranged.range_dictionary.RangeDictionary*) – A holder of the state variables

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

v_thresh

```
class spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic(du_th,  
                                                                              tau_th,  
                                                                              v_thresh)
```

Bases: *spynnaker.pyNN.models.neuron.threshold_types.abstract_threshold_type.AbstractThresholdType*

A stochastic threshold

add_parameters (*parameters*)

Add the initial values of the parameters to the parameter holder

Parameters **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the parameters

add_state_variables (*state_variables*)

Add the initial values of the state variables to the state variables holder

Parameters **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – A holder of the state variables

du_th

get_n_cpu_cycles (*n_neurons*)

Get the number of CPU cycles required to update the state

Parameters **n_neurons** (*int*) – The number of neurons to get the cycles for

Return type int

get_units (*variable*)

Get the units of the given variable

Parameters **variable** (*str*) – The name of the variable

get_values (*parameters, state_variables, vertex_slice, ts*)

Get the values to be written to the machine for this model

Parameters

- **parameters** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the parameters
- **state_variables** (spinn_utilities.ranged.range_dictionary.RangeDictionary) – The holder of the state variables
- **vertex_slice** – The slice of variables being retrieved

Returns A list with the same length as self.struct.field_types

Return type A list of (single value or list of values or RangedList)

has_variable (*variable*)

Determine if this component has a variable by the given name

Parameters **variable** (*str*) – The name of the variable

Return type bool

tau_th

update_values (*values, parameters, state_variables*)

Update the parameters and state variables with the given struct values that have been read from the machine

Parameters

- **values** – The values read from the machine, one for each struct element
- **parameters** – The holder of the parameters to update
- **state_variables** – The holder of the state variables to update

v_thresh

Submodules

spynnaker.pyNN.models.neuron.abstract_population_vertex module

class spynnaker.pyNN.models.neuron.abstract_population_vertex.**AbstractPopulationVertex** (*n_ne*

la-
bel,
con-
strain
max_
spike
ring_
in-
com-
ing_s
neu-
ron_
pynn

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,          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,          spinn_front_end_common.abstract_models.  
abstract_can_reset.AbstractCanReset
```

Underlying vertex model for Neural Populations.

BASIC_MALLOC_USAGE = 2

BYTES_TILL_START_OF_GLOBAL_PARAMETERS = 32

RUNTIME_SDP_PORT_SIZE = 4

SPIKE_RECORDING_REGION = 0

TRAFFIC_IDENTIFIER = 'BufferTraffic'

add_pre_run_connection_holder (*connection_holder, edge, synapse_info*)

Add a connection holder to the vertex to be filled in when the connections are actually generated.

clear_connection_cache ()

Clear the connection data stored in the vertex so far.

clear_recording (*variable, buffer_manager, placements, graph_mapper*)

Clear 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*)

Clear 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

conductance_based

create_machine_vertex (*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str* or *None*) – human readable label for the machine vertex
- **constraints** (*iterable* (*AbstractConstraint*)) – Constraints to be passed on to the machine vertex

describe ()

Get 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.

gen_on_machine (*vertex_slice*)

generate_data_specification (*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *graph_mapper*, *application_graph*, *machine_graph*, *routing_info*, *data_n_time_steps*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – the placement the vertex is located at

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 ExecutableType

get_connection_holders ()

get_connections_from_machine (*transceiver*, *placement*, *edge*, *graph_mapper*, *routing_infos*, *synapse_information*, *machine_time_step*, *using_extra_monitor_cores*, *placements=None*, *monitor_api=None*, *monitor_placement=None*, *monitor_cores=None*, *handle_time_out_configuration=True*, *fixed_routes=None*)

Get the connections from the machine post-run.

get_cpu_usage_for_atoms (*vertex_slice*)

get_data (*variable*, *n_machine_time_steps*, *placements*, *graph_mapper*, *buffer_manager*, *machine_time_step*)

Get the recorded data

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** (*AbstractOutgoingEdgePartition*) – An partition that goes in to this vertex
- **partition** – partition that goes into this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*) Gets the constraints for partitions going into this vertex.

Returns list of constraints

get_initial_value (*variable*, *selector=None*)

Gets the value for any variable whose in initialize_parameters.keys

Should return the current value not the default one.

Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added.

Parameters

- **variable** (*str*) – variable name with our without *_init*
- **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in *SpiNNUtils.spinn_utilities.ranged.abstract_sized.py*

Returns A list or an Object which act like a list

get_maximum_delay_supported_in_ms (*machine_time_step*)

Get the maximum delay supported by this vertex.

get_neuron_sampling_interval (*variable*)

Returns the current sampling interval for this variable

Parameters **variable** – PyNN name of the variable

Returns Sampling interval in micro seconds

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters

- **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex
- **partition** – the partition that leaves this vertex

Returns A list of constraints

Return type `list(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 (*vertex_slice, graph, machine_time_step*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type `ResourceContainer`

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

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_spikes_sampling_interval ()

Return the current sampling interval for spikes

Returns Sampling interval in micro seconds

get_synapse_id_by_target (*target*)

Get the ID of a synapse given the name.

Parameters **target** (*str*) – The name of the synapse

Return type `int`

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.

initialize_parameters

List the parameters that are initializable.

If “foo” is initializable there should be a setter `initialize_foo` and a getter property `foo_init`

Returns list of property names

is_recording (*variable*)

Determines if variable is being recorded

Returns True if variable are being recorded, False otherwise

Return type bool

is_recording_spikes ()

Determine 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, so that new changes can be detected before the next check.

mark_regions_reloaded ()

Indicate that the regions have been reloaded

n_atoms

The number of atoms in the vertex

Return type int

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 (*spec, placement, machine_time_step, time_scale_factor, graph_mapper, routing_info*)

Regenerate the data specification, only generating regions that have changed and need to be reloaded

Parameters

- **spec** (*DataSpecificationGenerator*) – Where to write the regenerated spec
- **placement** (*Placement*) – Where are we regenerating for?

requires_data_generation

True if changes that have been made require that data generation be performed. By default this returns False but can be overridden to indicate changes that require data regeneration.

Return type bool

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool

requires_memory_regions_to_be_reloaded()

Return true if any data region needs to be reloaded

Return type bool

reset_to_first_timestep()

Reset the object to first time step.

ring_buffer_sigma

set_initial_value (*variable, value, selector=None*)

Sets the value for any variable whose in initialize_parameters.keys

Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added

Parameters

- **variable** (*str*) – variable name with our without *_init*
- **value** – New value for the variable
- **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in *SpinnUtils.spinn_utilities.ranged.abstract_sized.py*

Returns A list or an Object which act like a list

set_recording (*variable, new_state=True, sampling_interval=None, indexes=None*)

Sets variable to being recorded

set_recording_spikes (*new_state=True, sampling_interval=None, indexes=None*)

Set spikes to being recorded. If new_state is false all other parameters are ignored.

Parameters

- **new_state** (*bool*) – Set if the spikes are recording or not
- **sampling_interval** – The interval at which spikes are recorded. Must be a whole multiple of the timestep None will be taken as the timestep
- **indexes** – The indexes of the neurons that will record spikes. If None the assumption is all neurons are recording

set_synapse_dynamics (*synapse_dynamics*)

Set the synapse dynamics of this vertex.

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

weight_scale

spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model module

class spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model.**AbstractPyNNNeuronModel** (*model*)

Bases: *spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel*

create_vertex(*n_neurons*, *label*, *constraints*, *spikes_per_second*, *ring_buffer_sigma*, *incoming_spike_buffer_size*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list* or *None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

default_population_parameters = {'incoming_spike_buffer_size': None, 'ring_buffer_sigma': 1.0}

classmethod get_max_atoms_per_core()

Get the maximum number of atoms per core for this model

Return type *int*

classmethod set_model_max_atoms_per_core(*n_atoms*=255)

Set the maximum number of atoms per core for this model

Parameters **n_atoms** (*int* or *None*) – The new maximum, or None for the largest possible

spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard module

class `spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.AbstractPyNNNeuronModelStandard`

Bases: `spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard.AbstractPyNNNeuronModel`

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,  

                                                                    con-  

                                                                    nec-  

                                                                    tions=None,  

                                                                    fixed_values=None,  

                                                                    no-  

                                                                    tify=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.generator_data module

```
class spynnaker.pyNN.models.neuron.generator_data.GeneratorData(synaptic_matrix_offset,
                                                                de-
                                                                layed_synaptic_matrix_offset,
                                                                max_row_n_words,
                                                                max_delayed_row_n_words,
                                                                max_row_n_synapses,
                                                                max_delayed_row_n_synapses,
                                                                pre_slices,
                                                                pre_slice_index,
                                                                post_slices,
                                                                post_slice_index,
                                                                pre_vertex_slice,
                                                                post_vertex_slice,
                                                                synapse_information,
                                                                max_stage,
                                                                ma-
                                                                chine_time_step)
```

Bases: object

Data for each connection of the synapse generator.

BASE_SIZE = 68

gen_data

Get the data to be written for this connection

Return type numpy array of uint32

size

The size of the generated data in bytes

Return type int

spynnaker.pyNN.models.neuron.population_machine_vertex module

```
class spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex(resource_recorder,
                                                                                      label,
                                                                                      constraints)
```

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`

Parameters

- **resources_required** –
- **recorded_region_ids** –

- **label** –
- **constraints** –

class EXTRA_PROVENANCE_DATA_ENTRIES
 Bases: `enum.Enum`

Entries for the provenance data generated by standard neuron models.

BUFFER_OVERFLOW_COUNT = 2

CURRENT_TIMER_TIC = 3

PLASTIC_SYNAPTIC_WEIGHT_SATURATION_COUNT = 4

PRE_SYNAPTIC_EVENT_COUNT = 0

SATURATION_COUNT = 1

N_ADDITIONAL_PROVENANCE_DATA_ITEMS = 5

PROFILE_TAG_LABELS = {0: 'TIMER', 1: 'DMA_READ', 2: 'INCOMING_SPIKE', 3: 'PROCESS_

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*)
 Retrieve the provenance data.

Parameters

- **transceiver** (*Transceiver*) – How to talk to the machine
- **placement** (*Placement*) – Which vertex are we retrieving from, and where was it

Return type `list(ProvenanceDataItem)`

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(int)`

get_recording_region_base_address (*txrx*, *placement*)
 Get the recording region base address

Parameters

- **txrx** (*Transceiver*) – the SpiNNMan instance
- **placement** (*Placement*) – the placement object of the core to find the address of

Returns the base address of the recording region

Return type `int`

is_recording ()
 Deduce if the recorder is actually recording

Return type `bool`

resources_required
 The resources required by the vertex

Return type `ResourceContainer`

spynnaker.pyNN.models.neuron.synaptic_manager module

```
class spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager(n_synapse_types,  
                                                                    ring_buffer_sigma,  
                                                                    spikes_per_second,  
                                                                    config,  
                                                                    popula-  
                                                                    tion_table_type=None,  
                                                                    synapse_io=None)
```

Bases: object

Deals with synapses

add_pre_run_connection_holder (*connection_holder, edge, synapse_info*)

changes_during_run

clear_connection_cache ()

gen_on_machine (*vertex_slice*)

True if the synapses should be generated on the machine

get_connection_holders ()

get_connections_from_machine (*transceiver, placement, machine_edge, graph_mapper,*
 routing_infos, synapse_info, machine_time_step, us-
 ing_extra_monitor_cores, placements=None, mon-
 itor_api=None, monitor_placement=None, moni-
 tor_cores=None, handle_time_out_configuration=True,
 fixed_routes=None)

get_dtcmm_usage_in_bytes ()

get_incoming_partition_constraints ()

get_maximum_delay_supported_in_ms (*machine_time_step*)

get_n_cpu_cycles ()

get_sdram_usage_in_bytes (*vertex_slice, machine_time_step, application_graph, app_vertex*)

ring_buffer_sigma

spikes_per_second

synapse_dynamics

vertex_executable_suffix

write_data_spec (*spec, application_vertex, post_vertex_slice, machine_vertex, placement, ma-*
 chine_graph, application_graph, routing_info, graph_mapper, weight_scale, ma-
 chine_time_step)

Module contents

```
class spynnaker.pyNN.models.neuron.AbstractPopulationVertex (n_neurons, label, constraints, max_atoms_per_core, spikes_per_second, ring_buffer_sigma, incoming_spike_buffer_size, neuron_impl, pynn_model)
```

```
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.AbstractProvidesOutgoingPartitionConstraints,
    spinn_front_end_common.abstract_models.abstract_provides_incoming_partition_constraints.
    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,
    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,
    spinn_front_end_common.abstract_models.
    abstract_can_reset.AbstractCanReset
```

Underlying vertex model for Neural Populations.

```
BASIC_MALLOC_USAGE = 2
```

```
BYTES_TILL_START_OF_GLOBAL_PARAMETERS = 32
```

```
RUNTIME_SDP_PORT_SIZE = 4
```

```
SPIKE_RECORDING_REGION = 0
```

```
TRAFFIC_IDENTIFIER = 'BufferTraffic'
```

```
add_pre_run_connection_holder (connection_holder, edge, synapse_info)
```

Add a connection holder to the vertex to be filled in when the connections are actually generated.

```
clear_connection_cache ()
```

Clear the connection data stored in the vertex so far.

```
clear_recording (variable, buffer_manager, placements, graph_mapper)
```

Clear 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*)

Clear 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

conductance_based

create_machine_vertex (*vertex_slice, resources_required, label=None, constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

describe ()

Get 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.

gen_on_machine (*vertex_slice*)

generate_data_specification (*spec, placement, machine_time_step, time_scale_factor, graph_mapper, application_graph, machine_graph, routing_info, data_n_time_steps*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – the placement the vertex is located at

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 ExecutableType

get_connection_holders ()

get_connections_from_machine (*transceiver, placement, edge, graph_mapper, routing_infos, synapse_information, machine_time_step, using_extra_monitor_cores, placements=None, monitor_api=None, monitor_placement=None, monitor_cores=None, handle_time_out_configuration=True, fixed_routes=None*)

Get the connections from the machine post-run.

get_cpu_usage_for_atoms (*vertex_slice*)

get_data (*variable, n_machine_time_steps, placements, graph_mapper, buffer_manager, machine_time_step*)

Get the recorded data

Parameters

- **variable** –
- **n_machine_time_steps** –
- **placements** –
- **graph_mapper** –
- **buffer_manager** –
- **machine_time_step** –

Returns

get_dtcn_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** (*AbstractOutgoingEdgePartition*) – An partition that goes in to this vertex
- **partition** – partition that goes into this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*) Gets the constraints for partitions going into this vertex.

Returns list of constraints

get_initial_value (*variable, selector=None*)

Gets the value for any variable whose in initialize_parameters.keys

Should return the current value not the default one.

Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added.

Parameters

- **variable** (*str*) – variable name with our without *_init*
- **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in *SpiNNUtils.spinn_utilities.ranged.abstract_sized.py*

Returns A list or an Object which act like a list

get_maximum_delay_supported_in_ms (*machine_time_step*)

Get the maximum delay supported by this vertex.

get_neuron_sampling_interval (*variable*)

Returns the current sampling interval for this variable

Parameters **variable** – PyNN name of the variable

Returns Sampling interval in micro seconds

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters

- **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex
- **partition** – the partition that leaves this vertex

Returns A list of constraints

Return type `list(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 (*vertex_slice*, *graph*, *machine_time_step*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type `ResourceContainer`

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

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_spikes_sampling_interval ()

Return the current sampling interval for spikes

Returns Sampling interval in micro seconds

get_synapse_id_by_target (*target*)

Get the ID of a synapse given the name.

Parameters **target** (*str*) – The name of the synapse

Return type `int`

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.

initialize_parameters

List the parameters that are initializable.

If “foo” is initializable there should be a setter `initialize_foo` and a getter property `foo_init`

Returns list of property names

is_recording (*variable*)

Determines if variable is being recorded

Returns True if variable are being recorded, False otherwise

Return type bool

is_recording_spikes ()

Determine 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, so that new changes can be detected before the next check.

mark_regions_reloaded ()

Indicate that the regions have been reloaded

n_atoms

The number of atoms in the vertex

Return type int

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 (*spec, placement, machine_time_step, time_scale_factor, graph_mapper, routing_info*)

Regenerate the data specification, only generating regions that have changed and need to be reloaded

Parameters

- **spec** (*DataSpecificationGenerator*) – Where to write the regenerated spec
- **placement** (*Placement*) – Where are we regenerating for?

requires_data_generation

True if changes that have been made require that data generation be performed. By default this returns False but can be overridden to indicate changes that require data regeneration.

Return type bool

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool

requires_memory_regions_to_be_reloaded()

Return true if any data region needs to be reloaded

Return type bool

reset_to_first_timestep()

Reset the object to first time step.

ring_buffer_sigma**set_initial_value** (*variable, value, selector=None*)

Sets the value for any variable whose in initialize_parameters.keys

Must support the variable as listed in initialize_parameters.keys, ideally also with *_init* removed or added

Parameters

- **variable** (*str*) – variable name with our without *_init*
- **value** – New value for the variable
- **selector** – a description of the subrange to accept. Or None for all. See: *_selector_to_ids* in *SpiNNUtils.spinn_utilities.ranged.abstract_sized.py*

Returns A list or an Object which act like a list

set_recording (*variable, new_state=True, sampling_interval=None, indexes=None*)

Sets variable to being recorded

set_recording_spikes (*new_state=True, sampling_interval=None, indexes=None*)

Set spikes to being recorded. If new_state is false all other parameters are ignored.

Parameters

- **new_state** (*bool*) – Set if the spikes are recording or not
- **sampling_interval** – The interval at which spikes are recorded. Must be a whole multiple of the timestep None will be taken as the timestep
- **indexes** – The indexes of the neurons that will record spikes. If None the assumption is all neurons are recording

set_synapse_dynamics (*synapse_dynamics*)

Set the synapse dynamics of this vertex.

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

weight_scale

```
class spynnaker.pyNN.models.neuron.ConnectionHolder (data_items_to_return,  
                                                    as_list,           n_pre_atoms,  
                                                    n_post_atoms,      connec-  
                                                    tions=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 (n_synapse_types,  
                                                    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*)

changes_during_run

clear_connection_cache ()

gen_on_machine (*vertex_slice*)

True if the synapses should be generated on the machine

```
get_connection_holders()

get_connections_from_machine(transceiver, placement, machine_edge, graph_mapper,
                             routing_infos, synapse_info, machine_time_step, using_extra_monitor_cores,
                             placements=None, monitor_api=None, monitor_placement=None, monitor_cores=None,
                             handle_time_out_configuration=True, fixed_routes=None)

get_dtcmm_usage_in_bytes()

get_incoming_partition_constraints()

get_maximum_delay_supported_in_ms(machine_time_step)

get_n_cpu_cycles()

get_sdram_usage_in_bytes(vertex_slice, machine_time_step, application_graph, app_vertex)

ring_buffer_sigma

spikes_per_second

synapse_dynamics

vertex_executable_suffix

write_data_spec(spec, application_vertex, post_vertex_slice, machine_vertex, placement, machine_graph,
               application_graph, routing_info, graph_mapper, weight_scale, machine_time_step)

class spynnaker.pyNN.models.neuron.PopulationMachineVertex(resources_required,
                                                           recorded_region_ids,
                                                           label, constraints)

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

Parameters
    • resources_required –
    • recorded_region_ids –
    • label –
    • constraints –

class EXTRA_PROVENANCE_DATA_ENTRIES
    Bases: enum.Enum

    Entries for the provenance data generated by standard neuron models.

    BUFFER_OVERFLOW_COUNT = 2
    CURRENT_TIMER_TIC = 3
    PLASTIC_SYNAPTIC_WEIGHT_SATURATION_COUNT = 4
    PRE_SYNAPTIC_EVENT_COUNT = 0
```

```

SATURATION_COUNT = 1
N_ADDITIONAL_PROVENANCE_DATA_ITEMS = 5
PROFILE_TAG_LABELS = {0: 'TIMER', 1: 'DMA_READ', 2: 'INCOMING_SPIKE', 3: 'PROCESS_}

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)
    Retrieve the provenance data.

    Parameters
    • transceiver (Transceiver) – How to talk to the machine
    • placement (Placement) – Which vertex are we retrieving from, and where was it

    Return type list(ProvenanceDataItem)

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(int)

get_recording_region_base_address(txrx, placement)
    Get the recording region base address

    Parameters
    • txrx (Transceiver) – the SpiNNMan instance
    • placement (Placement) – the placement object of the core to find the address of

    Returns the base address of the recording region

    Return type int

is_recording()
    Deduce if the recorder is actually recording

    Return type bool

resources_required
    The resources required by the vertex

    Return type ResourceContainer

class spynnaker.pyNN.models.neuron.AbstractPyNNNeuronModel(model)
    Bases: spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel

    create_vertex(n_neurons, label, constraints, spikes_per_second, ring_buffer_sigma, incoming_spike_buffer_size)
        Create a vertex for a population of the model

        Parameters
        • n_neurons (int) – The number of neurons in the population
        • label (str) – The label to give to the vertex
        • constraints (list or None) – A list of constraints to give to the vertex, or None

        Returns An application vertex for the population

```

Return type `pacman.model.graphs.application.ApplicationVertex`

default_population_parameters = {'incoming_spike_buffer_size': None, 'ring_buffer_size': None}

classmethod `get_max_atoms_per_core()`

Get the maximum number of atoms per core for this model

Return type `int`

classmethod `set_model_max_atoms_per_core(n_atoms=255)`

Set the maximum number of atoms per core for this model

Parameters `n_atoms` (*int* or *None*) – The new maximum, or None for the largest possible

class `spynnaker.pyNN.models.neuron.AbstractPyNNNeuronModelStandard` (*model_name*,
binary,
neuron_model,
input_type,
synapse_type,
threshold_type,
additional_input_type=None)

Bases: `spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model.AbstractPyNNNeuronModel`

`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` (*spike_times=None*)

Bases: `spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel`

create_vertex (*n_neurons*, *label*, *constraints*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list* or *None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

default_population_parameters = {}

spynnaker.pyNN.models.spike_source.spike_source_array_vertex module

class spynnaker.pyNN.models.spike_source.spike_source_array_vertex.**SpikeSourceArrayVertex** (*...*)

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

SPIKE_RECORDING_REGION_ID = 0

clear_spike_recording (*buffer_manager, placements, graph_mapper*)

Clear 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

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_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_spikes_sampling_interval ()

Return the current sampling interval for spikes

Returns Sampling interval in micro seconds

is_recording_spikes()

Determine 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, so that new changes can be detected before the next check.

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool

set_recording_spikes (*new_state=True, sampling_interval=None, indexes=None*)

Set spikes to being recorded. If new_state is false all other parameters are ignored.

Parameters

- **new_state** (*bool*) – Set if the spikes are recording or not
- **sampling_interval** – The interval at which spikes are recorded. Must be a whole multiple of the timestep None will be taken as the timestep
- **indexes** – The indexes of the neurons that will record spikes. If None the assumption is all neurons are recording

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** (*spike_times*, *min_atoms*, *max_atoms*, *min_time*, *max_time*, *split_value*)

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** (*rate=1.0, start=0, duration=None*)

Bases: *spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel*

create_vertex (*n_neurons, label, constraints, seed, max_rate*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

default_population_parameters = {'max_rate': None, 'seed': None}

classmethod `get_max_atoms_per_core()`

Get the maximum number of atoms per core for this model

Return type `int`

classmethod `set_model_max_atoms_per_core(n_atoms=500)`

Set the maximum number of atoms per core for this model

Parameters **n_atoms** (*int or None*) – The new maximum, or None for the largest possible

`spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex` module

class `spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePo`

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.abstract_models.abstract_supports_database_injection`.
`AbstractSupportsDatabaseInjection`, `spinn_front_end_common.interface`.
`profiling.abstract_has_profile_data.AbstractHasProfileData`

class **POISSON_SPIKE_SOURCE_REGIONS**

Bases: `enum.Enum`

An enumeration.

POISSON_PARAMS_REGION = 1

PROFILER_REGION = 5

PROVENANCE_REGION = 4

RATES_REGION = 2

SPIKE_HISTORY_REGION = 3

SYSTEM_REGION = 0

PROFILE_TAG_LABELS = {0: 'TIMER', 1: 'PROB_FUNC'}

get_profile_data (*transceiver, placement*)

Get the profile data recorded during simulation

Return type `spinn_front_end_common.interface.profiling.
profile_data.ProfileData`

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(int)`

get_recording_region_base_address(txrx, placement)

Get the recording region base address

Parameters

- **txrx** (*Transceiver*) – the SpiNNMan instance
- **placement** (*Placement*) – the placement object of the core to find the address of

Returns the base address of the recording region

Return type `int`

is_in_injection_mode

Whether this vertex is actually in injection mode.

Return type `bool`

is_recording()

Deduce if the recorder is actually recording

Return type `bool`

resources_required

The resources required by the vertex

Return type `ResourceContainer`

spynnaker.pyNN.models.spike_source.spike_source_poisson_variable module

class `spynnaker.pyNN.models.spike_source.spike_source_poisson_variable.SpikeSourcePoissonVariable`

Bases: `spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel`

create_vertex(n_neurons, label, constraints, seed)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

default_population_parameters = {'seed': None}

classmethod `get_max_atoms_per_core()`

Get the maximum number of atoms per core for this model

Return type `int`

classmethod `set_model_max_atoms_per_core(n_atoms=500)`

Set the maximum number of atoms per core for this model

Parameters `n_atoms` (*int* or *None*) – The new maximum, or None for the largest possible

spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex module

class `spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVert`

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

SPIKE_RECORDING_REGION_ID = 0

clear_spike_recording (*buffer_manager*, *placements*, *graph_mapper*)

Clear 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 (*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str* or *None*) – human readable label for the machine vertex
- **constraints** (*iterable* (*AbstractConstraint*)) – Constraints to be passed on to the machine vertex

describe ()

Return 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

durations

generate_data_specification (*spec*, *placement*, *machine_time_step*, *time_scale_factor*, *graph_mapper*, *routing_info*, *data_n_time_steps*, *graph*, *first_machine_time_step*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – the placement the vertex is located at

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 ExecutableType

static get_cpu_usage_for_atoms ()

static get_dtcm_usage_for_atoms ()

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type `list(AbstractConstraint)`

get_rates_bytes (*vertex_slice*)

Gets the size of the Poisson rates in bytes

Parameters **vertex_slice** –

get_recording_sdr_usage (*vertex_slice, machine_time_step*)

get_resources_used_by_atoms (*vertex_slice, machine_time_step*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type `ResourceContainer`

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

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_spikes_sampling_interval ()

Return the current sampling interval for spikes

Returns Sampling interval in micro seconds

is_recording_spikes ()

Determine 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, so that new changes can be detected before the next check.

mark_regions_reloaded ()

Indicate that the regions have been reloaded

max_rate

n_atoms

The number of atoms in the vertex

Return type `int`

rate

rates

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 (*spec*, *placement*, *machine_time_step*,
time_scale_factor, *graph_mapper*, *routing_info*, *graph*,
first_machine_time_step)

Regenerate the data specification, only generating regions that have changed and need to be reloaded

Parameters

- **spec** (*DataSpecificationGenerator*) – Where to write the regenerated spec
- **placement** (*Placement*) – Where are we regenerating for?

requires_mapping

True if changes that have been made require that mapping be performed. By default this returns False but can be overridden to indicate changes that require mapping.

Return type bool

requires_memory_regions_to_be_reloaded (*first_machine_time_step*)

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

set_recording_spikes (*new_state=True*, *sampling_interval=None*, *indexes=None*)

Set spikes to being recorded. If new_state is false all other parameters are ignored.

Parameters

- **new_state** (*bool*) – Set if the spikes are recording or not
- **sampling_interval** – The interval at which spikes are recorded. Must be a whole multiple of the timestep None will be taken as the timestep
- **indexes** – The indexes of the neurons that will record spikes. If None the assumption is all neurons are recording

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

starts

Module contents

```
class spynnaker.pyNN.models.spike_source.SpikeSourceArray (spike_times=None)
    Bases: spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel

    create_vertex (n_neurons, label, constraints)
        Create a vertex for a population of the model

        Parameters
        • n_neurons (int) – The number of neurons in the population
        • label (str) – The label to give to the vertex
        • constraints (list or None) – A list of constraints to give to the vertex, or None

        Returns An application vertex for the population

        Return type pacman.model.graphs.application.ApplicationVertex

    default_population_parameters = {}

class spynnaker.pyNN.models.spike_source.SpikeSourceFromFile (spike_time_file,
                                                                min_atom=None,
                                                                max_atom=None,
                                                                min_time=None,
                                                                max_time=None,
                                                                split_value='t')

    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 (rate=1.0, start=0,
                                                                duration=None)
    Bases: spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel

    create_vertex (n_neurons, label, constraints, seed, max_rate)
        Create a vertex for a population of the model

        Parameters
        • n_neurons (int) – The number of neurons in the population
        • label (str) – The label to give to the vertex
        • constraints (list or None) – A list of constraints to give to the vertex, or None

        Returns An application vertex for the population

        Return type pacman.model.graphs.application.ApplicationVertex

    default_population_parameters = {'max_rate': None, 'seed': None}

    classmethod get_max_atoms_per_core ()
        Get the maximum number of atoms per core for this model

        Return type int

    classmethod set_model_max_atoms_per_core (n_atoms=500)
        Set the maximum number of atoms per core for this model
```

Parameters **n_atoms** (*int* or *None*) – The new maximum, or *None* for the largest possible

```
class spynnaker.pyNN.models.spike_source.SpikeSourcePoissonVariable (rates,
                                                                    starts,
                                                                    dura-
                                                                    tions=None)
```

Bases: `spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel`

create_vertex (*n_neurons*, *label*, *constraints*, *seed*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list* or *None*) – A list of constraints to give to the vertex, or *None*

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

```
default_population_parameters = {'seed': None}
```

```
classmethod get_max_atoms_per_core()
```

Get the maximum number of atoms per core for this model

Return type *int*

```
classmethod set_model_max_atoms_per_core (n_atoms=500)
```

Set the maximum number of atoms per core for this model

Parameters **n_atoms** (*int* or *None*) – The new maximum, or *None* for the largest possible

spynnaker.pyNN.models.utility_models package

Subpackages

spynnaker.pyNN.models.utility_models.delays package

Submodules

spynnaker.pyNN.models.utility_models.delays.delay_block module

```
class spynnaker.pyNN.models.utility_models.delays.delay_block.DelayBlock (n_delay_stages,
                                                                              de-
                                                                              lay_per_stage,
                                                                              ver-
                                                                              tex_slice)
```

Bases: *object*

A block of delays for a vertex.

add_delay (*source_id*, *stage*)

delay_block

spynnaker.pyNN.models.utility_models.delays.delay_extension_machine_vertex module

class spynnaker.pyNN.models.utility_models.delays.delay_extension_machine_vertex.**DelayExtensionMachineVertex**

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`

An enumeration.

N_BUFFER_OVERFLOWS = 4

N_DELAYS = 5

N_PACKETS_ADDED = 2

N_PACKETS_PROCESSED = 1

N_PACKETS_RECEIVED = 0

N_PACKETS_SENT = 3

N_EXTRA_PROVENANCE_DATA_ENTRIES = 6

get_provenance_data_from_machine (*transceiver*, *placement*)

Retrieve the provenance data.

Parameters

- **transceiver** (*Transceiver*) – How to talk to the machine
- **placement** (*Placement*) – Which vertex are we retrieving from, and where was it

Return type `list(ProvenanceDataItem)`

resources_required

The resources required by the vertex

Return type `ResourceContainer`

spynnaker.pyNN.models.utility_models.delays.delay_extension_vertex module

class spynnaker.pyNN.models.utility_models.delays.delay_extension_vertex.**DelayExtensionVertex**

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)

Parameters

- **n_neurons** – the number of neurons
- **delay_per_stage** – the delay per stage
- **source_vertex** – where messages are coming from
- **machine_time_step** – how long is the machine time step
- **timescale_factor** – what slowdown factor has been applied
- **constraints** – the vertex constraints
- **label** – the vertex label

add_delays (*vertex_slice, source_ids, stages*)

Add delayed connections for a given vertex slice

add_generator_data (*max_row_n_synapses, max_delayed_row_n_synapses, pre_slices,
pre_slice_index, post_slices, post_slice_index, pre_vertex_slice,
post_vertex_slice, synapse_information, max_stage, machine_time_step*)

Add delays for a connection to be generated

create_machine_vertex (*vertex_slice, resources_required, label=None, constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str or None*) – human readable label for the machine vertex
- **constraints** (*iterable(AbstractConstraint)*) – Constraints to be passed on to the machine vertex

gen_on_machine (*vertex_slice*)

Determine if the given slice needs to be generated on the machine

generate_data_specification (*spec, placement, machine_graph, graph_mapper, routing_infos*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – the placement the vertex is located at

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 ExecutableType

get_cpu_usage_for_atoms (*vertex_slice*)

get_dtcmm_usage_for_atoms (*vertex_slice*)

get_n_keys_for_partition (*partition*, *graph_mapper*)

Get the number of keys required by the given partition of edges.

Parameters

- **partition** (*AbstractOutgoingEdgePartition*) – An partition that comes out of this vertex
- **graph_mapper** (*GraphMapper*) – A mapper between the graphs

Returns A list of constraints

Return type list(*AbstractConstraint*)

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

get_resources_used_by_atoms (*vertex_slice*, *graph*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMMResource and SDRAMResource

Return type *ResourceContainer*

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

get_sdram_usage_for_atoms (*out_edges*)

n_atoms

The number of atoms in the vertex

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*,
total_n_vertices, *machine_time_step*, *time_scale_factor*,
n_outgoing_edges)

Generate Delay Parameter data

write_setup_info (*spec*, *machine_time_step*, *time_scale_factor*)

spynnaker.pyNN.models.utility_models.delays.delay_generator_data module**class** spynnaker.pyNN.models.utility_models.delays.delay_generator_data.**DelayGeneratorData** (*n*

Bases: object

Data for each connection of the delay generator

BASE_SIZE = 32**gen_data**

Get the data to be written for this connection

Return type numpy array of uint32**size**

The size of the generated data in bytes

Return type int**Module contents****class** spynnaker.pyNN.models.utility_models.delays.**DelayBlock** (*n_delay_stages, de-
lay_per_stage, ver-
tex_slice*)

Bases: object

A block of delays for a vertex.

add_delay (*source_id, stage*)**delay_block****class** spynnaker.pyNN.models.utility_models.delays.**DelayExtensionMachineVertex** (*resources_required,
la-
bel,
con-
straints=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`

An enumeration.

N_BUFFER_OVERFLOWS = 4**N_DELAYS** = 5

```

N_PACKETS_ADDED = 2
N_PACKETS_PROCESSED = 1
N_PACKETS_RECEIVED = 0
N_PACKETS_SENT = 3
N_EXTRA_PROVENANCE_DATA_ENTRIES = 6
get_provenance_data_from_machine(transceiver, placement)
    Retrieve the provenance data.

```

Parameters

- **transceiver** (*Transceiver*) – How to talk to the machine
- **placement** (*Placement*) – Which vertex are we retrieving from, and where was it

Return type list(*ProvenanceDataItem*)

resources_required

The resources required by the vertex

Return type *ResourceContainer*

```

class spynnaker.pyNN.models.utility_models.delays.DelayExtensionVertex(n_neurons,
                                                                    de-
                                                                    lay_per_stage,
                                                                    source_vertex,
                                                                    ma-
                                                                    chine_time_step,
                                                                    timescale_factor,
                                                                    con-
                                                                    straints=None,
                                                                    la-
                                                                    bel='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.AbstractProvidesOutgoingPartitionConstraints*, *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)

Parameters

- **n_neurons** – the number of neurons
- **delay_per_stage** – the delay per stage
- **source_vertex** – where messages are coming from
- **machine_time_step** – how long is the machine time step
- **timescale_factor** – what slowdown factor has been applied
- **constraints** – the vertex constraints
- **label** – the vertex label

add_delays (*vertex_slice*, *source_ids*, *stages*)

Add delayed connections for a given vertex slice

add_generator_data (*max_row_n_synapses*, *max_delayed_row_n_synapses*, *pre_slices*,
pre_slice_index, *post_slices*, *post_slice_index*, *pre_vertex_slice*,
post_vertex_slice, *synapse_information*, *max_stage*, *machine_time_step*)

Add delays for a connection to be generated

create_machine_vertex (*vertex_slice*, *resources_required*, *label=None*, *constraints=None*)

Create a machine vertex from this application vertex

Parameters

- **vertex_slice** (*Slice*) – The slice of atoms that the machine vertex will cover
- **resources_required** (*ResourceContainer*) – the resources used by the machine vertex
- **label** (*str* or *None*) – human readable label for the machine vertex
- **constraints** (*iterable* (*AbstractConstraint*)) – Constraints to be passed on to the machine vertex

gen_on_machine (*vertex_slice*)

Determine if the given slice needs to be generated on the machine

generate_data_specification (*spec*, *placement*, *machine_graph*, *graph_mapper*, *routing_infos*)

Generate a data specification.

Parameters

- **spec** (*DataSpecificationGenerator*) – The data specification to write to
- **placement** (*Placement*) – the placement the vertex is located at

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 *ExecutableType*

get_cpu_usage_for_atoms (*vertex_slice*)

get_dtcn_usage_for_atoms (*vertex_slice*)

get_n_keys_for_partition (*partition*, *graph_mapper*)

Get the number of keys required by the given partition of edges.

Parameters

- **partition** (*AbstractOutgoingEdgePartition*) – An partition that comes out of this vertex
- **graph_mapper** (*GraphMapper*) – A mapper between the graphs

Returns A list of constraints

Return type *list*(*AbstractConstraint*)

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*AbstractConstraint*)

get_resources_used_by_atoms (*vertex_slice, graph*)

Get the separate resource requirements for a range of atoms

Parameters **vertex_slice** (*Slice*) – the low value of atoms to calculate resources from

Returns a Resource container that contains a CPUCyclesPerTickResource, DTCMResource and SDRAMResource

Return type *ResourceContainer*

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

get_sdram_usage_for_atoms (*out_edges*)

n_atoms

The number of atoms in the vertex

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, total_n_vertices, machine_time_step, time_scale_factor, n_outgoing_edges*)

Generate Delay Parameter data

write_setup_info (*spec, machine_time_step, time_scale_factor*)

spynnaker.pyNN.models.utility_models.spike_injector package

Submodules

spynnaker.pyNN.models.utility_models.spike_injector.spike_injector module

class spynnaker.pyNN.models.utility_models.spike_injector.spike_injector.**SpikeInjector**

Bases: *spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel*

create_vertex (*n_neurons, label, constraints, port, virtual_key, reserve_reverse_ip_tag*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

default_population_parameters = {'port': None, 'reserve_reverse_ip_tag': False, 'vir

`spynnaker.pyNN.models.utility_models.spike_injector.spike_injector_vertex` module

class `spynnaker.pyNN.models.utility_models.spike_injector.spike_injector_vertex.SpikeInject`

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*)

Clear 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 = {'label': 'spikeInjector', 'port': None, 'virtual_key': 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.

get_outgoing_partition_constraints (*partition*)

Get constraints to be added to the given edge that comes out of this vertex.

Parameters **partition** (*AbstractOutgoingEdgePartition*) – An edge that comes out of this vertex

Returns A list of constraints

Return type list(*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

get_spikes_sampling_interval ()

Return the current sampling interval for spikes

Returns Sampling interval in micro seconds

is_recording_spikes ()

Determine if spikes are being recorded

Returns True if spikes are being recorded, False otherwise

Return type bool

port

set_recording_spikes (*new_state=True, sampling_interval=None, indexes=None*)

Set spikes to being recorded. If new_state is false all other parameters are ignored.

Parameters

- **new_state** (*bool*) – Set if the spikes are recording or not
- **sampling_interval** – The interval at which spikes are recorded. Must be a whole multiple of the timestep None will be taken as the timestep
- **indexes** – The indexes of the neurons that will record spikes. If None the assumption is all neurons are recording

virtual_key

Module contents

class spynnaker.pyNN.models.utility_models.spike_injector.**SpikeInjector**

Bases: *spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel*

create_vertex (*n_neurons, label, constraints, port, virtual_key, reserve_reverse_ip_tag*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type *pacman.model.graphs.application.ApplicationVertex*

default_population_parameters = {'port': None, 'reserve_reverse_ip_tag': False, 'vir

spynnaker.pyNN.models.utility_models.synapse_expander package

Submodules

spynnaker.pyNN.models.utility_models.synapse_expander.synapse_expander module

`spynnaker.pyNN.models.utility_models.synapse_expander.synapse_expander.synapse_expander` (*application vertex*)

Run the synapse expander - needs to be done after data has been loaded

Module contents

Module contents

Submodules

spynnaker.pyNN.models.abstract_pynn_model module

class `spynnaker.pyNN.models.abstract_pynn_model.AbstractPyNNModel`

Bases: `object`

A Model that can be passed in to a Population object in PyNN

create_vertex (*n_neurons, label, constraints*)

Create a vertex for a population of the model

Parameters

- **n_neurons** (*int*) – The number of neurons in the population
- **label** (*str*) – The label to give to the vertex
- **constraints** (*list or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type `pacman.model.graphs.application.ApplicationVertex`

default_initial_values = {}

default_parameters = {}

default_population_parameters

Get the default values for the parameters at the population level These are parameters that can be passed in to the Population constructor in addition to the standard PyNN options

Return type `dict(str, object)`

classmethod `get_max_atoms_per_core()`

Get the maximum number of atoms per core for this model

Return type int

classmethod `get_parameter_names()`

Get the names of the parameters of the model

Return type list(str)

classmethod `has_parameter(name)`

Determine if the model has a parameter with the given name

Parameters `name` (str) – The name of the parameter to check for

Return type bool

classmethod `set_model_max_atoms_per_core(n_atoms=9223372036854775807)`

Set the maximum number of atoms per core for this model

Parameters `n_atoms` (int or None) – The new maximum, or None for the largest possible

spynnaker.pyNN.models.defaults module

`spynnaker.pyNN.models.defaults.default_initial_values(state_variables)`

Specifies arguments which are state variables. Only works on the `__init__` method of a class that is additionally decorated with `defaults`` ()

Parameters `state_variables` (set of str) – The names of the arguments that are state variables

`spynnaker.pyNN.models.defaults.default_parameters(parameters)`

Specifies arguments which are parameters. Only works on the `__init__` method of a class that is additionally decorated with `defaults`` ()

Parameters `parameters` (set of str) – The names of the arguments that are parameters

`spynnaker.pyNN.models.defaults.defaults(cls)`

Get the default parameters and state variables from the arguments to the `__init__` method. This uses the decorators `default_parameters()` and `default_initial_values()` to determine the parameters and state variables respectively. If only one is specified, the other is assumed to be the remaining arguments. If neither are specified, it is assumed that all default arguments are parameters.

`spynnaker.pyNN.models.defaults.get_dict_from_init(init, skip=None, include=None)`

spynnaker.pyNN.models.pyNN_population_common module

```
class spynnaker.pyNN.models.pyNN_population_common.PyNNPopulationCommon (spinnaker_control,  
size,  
label,  
constraints,  
model,  
structure,  
initial_values,  
additional_parameters=None)
```

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)

conductance_based

True if the population uses conductance inputs

first_id

get (*parameter_names, gather=True, simplify=True*)

Get the values of a parameter for every local cell in the population.

Parameters

- **parameter_names** – Name of parameter. This is either a single string or a list of strings
- **gather** – pointless on sPyNNaker

Returns A single list of values (or possibly a single value) if *parameter_names* is a string, or a dict of these if *parameter_names* is a list.

Return type `str` or `list(str)` or `dict(str,str)` or `dict(str,list(str))`

get_by_selector (*selector, parameter_names*)

Get the values of a parameter for the selected cell in the population.

Parameters

- **parameter_names** – Name of parameter. This is either a single string or a list of strings
- **selector** – a description of the subrange to accept. Or `None` for all. See: `_selector_to_ids` in `SpiNNUtils.spinn_utilities.ranged.abstract_sized.py`

Returns A single list of values (or possibly a single value) if *parameter_names* is a string or a dict of these if *parameter_names* is a list.

Return type `str` or `list(str)` or `dict(str,str)` or `dict(str,list(str))`

get_spike_counts (*spikes, gather=True*)

Return the number of spikes for each neuron.

Defined by <http://neuralensemble.org/docs/PyNN/reference/populations.html>

Parameters **gather** – pointless on sPyNNaker

id_to_index (*id*)

Given the ID(s) of cell(s) in the Population, return its (their) index (order in the Population).

Defined by <http://neuralensemble.org/docs/PyNN/reference/populations.html>

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. Defined by <http://neuralensemble.org/docs/PyNN/reference/populations.html>

index_to_id (*index*)

Given the index (order in the Population) of cell(s) in the Population, return their ID(s)

inject (*current_source*)

Connect a current source to all cells in the Population.

Defined by <http://neuralensemble.org/docs/PyNN/reference/populations.html>

label

The label of the population

last_id

local_size

The number of local cells

Defined by <http://neuralensemble.org/docs/PyNN/reference/populations.html>

mark_no_changes ()

positions

Return the position array for structured populations.

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** (*str or dict*) – the parameter to set
- **value** – the value of the parameter to set.

set_by_selector (*selector, parameter, value=None*)

Set one or more parameters for selected 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

- **selector** – See `RangedList.set_value_by_selector` as this is just a pass through method
- **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(str, int)*) – A dictionary containing “x”, “y” and optionally “p” as keys, and ints as values

set_max_atoms_per_core (*max_atoms_per_core*)

Supports the setting of this population’s max atoms per core

Parameters **max_atoms_per_core** – 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,
con-
nec-
tor,
synapse_dynamics_stdp,
tar-
get,
pre_synaptic_population,
post_synaptic_population,
pre-
pop_is_view,
post-
pop_is_view,
rng,
ma-
chine_time_step,
user_max_delay,
la-
bel,
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.

Parameters **gather** – If False, only get the number of connections locally. Which means nothing on SpiNNaker..

spynnaker.pyNN.models.recording_common module

class spynnaker.pyNN.models.recording_common.**RecordingCommon** (*population*)

Bases: object

Object to hold recording behaviour.

Parameters **population** – the population to record for

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.**GraphEdgeWeightt**

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.**Spynna**

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.MunichIoSpiNNakerLinkProt
```

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

    types of modes supported by this protocol

    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, 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 `spinn_front_end_common.utility_models.
multi_cast_command.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,
                                                                              pix-
                                                                              els,
                                                                              bits_per_coordinate)
```

Bases: `enum.Enum`

An enumeration.

```
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`

An enumeration.

```
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_1
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,            in-
                                                                stance_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`

types of modes supported by this protocol

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, 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 `spinn_front_end_common.utility_models.
multi_cast_command.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

```

An enumeration.

```

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

```

An enumeration.

```

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

spynnaker.pyNN.utilities.ranged package

Submodules

spynnaker.pyNN.utilities.ranged.spynnaker_ranged_dict module

class spynnaker.pyNN.utilities.ranged.spynnaker_ranged_dict.**SpynnakerRangeDictionary** (*size, de-faults=1*)

Bases: spinn_utilities.ranged.range_dictionary.RangeDictionary

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

Parameters

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

list_factory (*size, value, key*)

Defines which class or subclass of RangedList to use

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

Parameters

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

Returns AbstractList in this case a RangedList

spynnaker.pyNN.utilities.ranged.spynnaker_ranged_list module

class spynnaker.pyNN.utilities.ranged.spynnaker_ranged_list.**SpynnakerRangedList** (*size=None, value=None, key=None, use_list_as_val*)

Bases: spinn_utilities.ranged.ranged_list.RangedList

Parameters

- **size** – Fixed length of the list
- **value** – value to given to all elements in the list
- **key** – The dict key this list covers. This is used only for better Exception messages

- **use_list_as_value** – True if the value *is* a list

static as_list (*value, size, ids=None*)

Converts (if required) the value into a list of a given size. An exception is raised if value cannot be given size elements.

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

Parameters value –

Returns value as a list

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

static is_list (*value, size*)

Determines if the value should be treated as a list.

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

Module contents

class spynnaker.pyNN.utilities.ranged.**SpynnakerRangeDictionary** (*size, defaults=None*)

Bases: spinn_utilities.ranged.range_dictionary.RangeDictionary

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

Parameters

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

list_factory (*size, value, key*)

Defines which class or subclass of RangedList to use

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

Parameters

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

Returns AbstractList in this case a RangedList

class spynnaker.pyNN.utilities.ranged.**SpynnakerRangedList** (*size=None, value=None, key=None, use_list_as_value=False*)

Bases: spinn_utilities.ranged.ranged_list.RangedList

Parameters

- **size** – Fixed length of the list
- **value** – value to given to all elements in the list
- **key** – The dict key this list covers. This is used only for better Exception messages
- **use_list_as_value** – True if the value *is* a list

static as_list (*value, size, ids=None*)

Converts (if required) the value into a list of a given size. An exception is raised if value cannot be given size elements.

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

Parameters value –

Returns value as a list

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

static is_list (*value, size*)

Determines if the value should be treated as a list.

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

Submodules**spynnaker.pyNN.utilities.constants module**

class spynnaker.pyNN.utilities.constants.**POPULATION_BASED_REGIONS**

Bases: `enum.Enum`

Regions for populations.

CONNECTOR_BUILDER = 9

DIRECT_MATRIX = 10

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.extracted_data module

class spynnaker.pyNN.utilities.extracted_data.**ExtractedData**

Bases: object

Data holder for all synaptic data being extracted in parallel. @Chimp: play here to hearts content.

get (*projection, attribute*)

Allow getting data from a given projection and attribute

Parameters

- **projection** – the projection data was extracted from
- **attribute** – the attribute to retrieve

Returns the attribute data in a connection holder

set (*projection, attribute, data*)

Allow the addition of data from a projection and attribute.

Parameters

- **projection** – the projection data was extracted from
- **attribute** – the attribute to store
- **data** – attribute data in a connection holder

Return type None

spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider module

class spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.**FakeHBPPortalMachineProvide**

Bases: object

create ()

destroy ()

get_machine_info ()

wait_till_not_ready ()

wait_until_ready ()

spynnaker.pyNN.utilities.reports module

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.**SpYNNakerConnectionHolderGenerations**
 Bases: object
 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()`
`get_random_distribution()`
`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.**SpYNNakerNeuronNetworkSpecificationReport**
 Bases: object

spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report module

class spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report.**SpYNNakerSynapticMatrixReport**
 Bases: object
 Generate 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.check_sampling_interval(sampling_interval)`

`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.convert_to(value, data_type)`

Convert a value to a given data type

Parameters

- **value** – The value to convert
- **data_type** – The data type to convert to

Returns The converted data as a numpy data type

`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_n_bits(n_values)`

Determine how many bits are required for the given number of values

`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_trials,
n_trials,
selection_prob,
chance=0.01)`

Get the likely maximum number of items that will be selected from a set of n_trials from a total set of n_total_trials 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 boundary value for this distribution

Could return None

`spynnaker.pyNN.utilities.utility_calls.low(dist)`

Gets the high or min boundary 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, ex-
 tra=False)`

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>

Parameters

- **file_path** (*str*) – absolute path to a file containing spike values
- **min_atom** (*int*) – min neuron ID to which neurons to read in
- **max_atom** (*int*) – max neuron ID to which neurons to read in
- **min_time** (*int*) – min time slot to read neurons values of.
- **max_time** (*int*) – max time slot to read neurons values of.
- **split_value** (*str*) – the pattern to split by

Returns a numpy array with max_atom elements each of which is a list of spike times.

Return type `numpy.array(int, int)`

`spynnaker.pyNN.utilities.utility_calls.validate_mars_kiss_64_seed(seed)`
 Update the seed to make it compatible with the rng algorithm

Module contents

1.1.1.2 Submodules

1.1.1.3 spynnaker.pyNN.abstract_spinnaker_common module

```
class spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon (graph_label,  
                                                                    database_socket_addresses,  
                                                                    n_chips_required,  
                                                                    n_boards_required,  
                                                                    timestep,  
                                                                    max_delay,  
                                                                    min_delay,  
                                                                    host-  
                                                                    name,  
                                                                    user_extra_algorithm_xml_p  
                                                                    user_extra_mapping_inputs=  
                                                                    user_extra_algorithms_pre_r  
                                                                    time_scale_factor=None,  
                                                                    ex-  
                                                                    tra_post_run_algorithms=No  
                                                                    ex-  
                                                                    tra_mapping_algorithms=No  
                                                                    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*)

Parameters **vertex** – the vertex to add to the graph

Type ApplicationVertex

Return type None

Raises

- **ConfigurationException** – when both graphs contain vertices
- **PacmanConfigurationException** – If there is an attempt to add the same vertex more than once

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.

get_projections_data (*projection_to_attribute_map*)

Common data extractor for projection data. Allows fully exploitation of the ????

Parameters **projection_to_attribute_map** (*dict of projection with set of attributes*) – the projection to attributes mapping

Returns a extracted data object with get method for getting the data

Return type `spynnaker.pyNN.utilities.extracted_data.ExtractedData`

id_counter

Getter for id_counter, currently used by the populations.

Note: Maybe it could live in the pop class???

Returns

Return type int

max_delay

The maximum supported delay, in milliseconds.

min_delay

The minimum supported delay, in milliseconds.

static register_binary_search_path (*search_path*)

Register an additional binary search path for executables.

Parameters *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 milliseconds) 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

1.1.1.4 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: `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.PyNN

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.

1.1.1.5 spynnaker.pyNN.spynnaker_external_device_plugin_manager module

class `spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager`

Bases: `object`

User-level interface 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_common.PyNNPopulationCommon*) – 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 an 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 configuration 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 configuration 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
- **label** (*str*) – The label of the gatherer vertex
- **partition_ids** (*list (str)*) – The names of the partitions to create edges for

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** (*spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon*) – The pyNN population object from which spikes will be sent.
- **device** (*spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon* or *pacman.model.graphs.application.ApplicationVertex*) – The pyNN population or 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*)

Add an 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_poisson_live_rate_control (*poisson_population*, *control_label_extension*='control', *receive_port*=None, *database_notify_host*=None, *database_notify_port_num*=None, *database_ack_port_num*=None, *notify*=True, *reserve_reverse_ip_tag*=False)

Add a live rate controller to a Poisson population.

Parameters

- **poisson_population** (*spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon*) – The population to control
- **control_label_extension** (*str*) – An extension to add to the label of the Poisson source. Must match up with the equivalent in the SpynnakerPoissonControlConnection
- **receive_port** (*int*) – The port that the SpiNNaker board should listen on
- **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
- **reserve_reverse_ip_tag** (*bool*) – True if a reverse ip tag is to be used, False if SDP is to be used (default)

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, lpg_label,
                                         port=None, hostname=None,
                                         board_address=None, tag=None,
                                         strip_sdp=True, use_prefix=False,
                                         key_prefix=None, prefix_type=None, mes-
                                         sage_type=<EIEIOType.KEY_32_BIT:
                                         2>, right_shift=0, pay-
                                         load_as_time_stamps=True,
                                         use_payload_prefix=True, pay-
                                         load_prefix=None, payload_right_shift=0,
                                         number_of_packets_sent_per_time_step=0,
                                         partition_ids=None)

```

Add an edge from a vertex to the live packet gatherer, builds as needed and has all the parameters for the creation of the live packet gatherer if needed.

1.1.1.6 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()
    get_random_distribution()
    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)

```

1.1.1.7 Module contents

1.2 Submodules

1.3 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)

```

1.4 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

1.5 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)
```

1.6 Module contents

CHAPTER 2

Indices and tables

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

S

297

spynnaker.pyNN.models.neural_properties, spynnaker.pyNN.models.neuron.implementations.abstract	
108	118
spynnaker.pyNN.models.neural_properties.spynnaker.pyNN.models.neuron.implementations.neuron	
108	120
spynnaker.pyNN.models.neuron, 235	spynnaker.pyNN.models.neuron.implementations.range
spynnaker.pyNN.models.neuron.abstract_population, 223	spynnaker.pyNN.models.neuron.implementations.structure,
spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model, 229	spynnaker.pyNN.models.neuron.input_types,
spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard, 230	spynnaker.pyNN.models.neuron.input_types.abstract_
spynnaker.pyNN.models.neuron.additional_inputs, 110	spynnaker.pyNN.models.neuron.input_types.input_type
spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input, 109	spynnaker.pyNN.models.neuron.input_types.input_type
spynnaker.pyNN.models.neuron.additional_inputs.additional_input_ca2_adaptive, 109	spynnaker.pyNN.models.neuron.input_types.input_type
spynnaker.pyNN.models.neuron.builds, 114	133
spynnaker.pyNN.models.neuron.builds.eif_spynnaker.pyNN.models.neuron.master_pop_table_gener	
111	140
spynnaker.pyNN.models.neuron.builds.hh_spynnaker.pyNN.models.neuron.master_pop_table_gener	
112	138
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.master_pop_table_gener	
112	139
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.neuron_models,	
112	146
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.neuron_models.abstract	
112	142
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.neuron_models.neuron_r	
112	143
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.neuron_models.neuron_r	
113	145
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.plasticity,	
113	164
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.plasticity.stdp,	
113	164
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.plasticity.stdp.common	
113	150
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.plasticity.stdp.common	
113	150
spynnaker.pyNN.models.neuron.builds.if_spynnaker.pyNN.models.neuron.plasticity.stdp.synaps	
113	151
spynnaker.pyNN.models.neuron.builds.izk_spynnaker.pyNN.models.neuron.plasticity.stdp.synaps	
114	150
spynnaker.pyNN.models.neuron.builds.izk_spynnaker.pyNN.models.neuron.plasticity.stdp.synaps	
114	151
spynnaker.pyNN.models.neuron.connection_hpydarker.pyNN.models.neuron.plasticity.stdp.synaps	
231	151
spynnaker.pyNN.models.neuron.generator_dapynaker.pyNN.models.neuron.plasticity.stdp.timing	
232	156
spynnaker.pyNN.models.neuron.implementat	spynnaker.pyNN.models.neuron.plasticity.stdp.timing
123	152
spynnaker.pyNN.models.neuron.implementat	spynnaker.pyNN.models.neuron.plasticity.stdp.timing
116	152

spynnaker.pyNN.models.spike_source, 253
 spynnaker.pyNN.models.spike_source.spikes, 244
 spynnaker.pyNN.models.spike_source.spike_source, 245
 spynnaker.pyNN.models.spike_source.spike_source.array_vertex, 245
 spynnaker.pyNN.models.spike_source.spike_source.from_file, 246
 spynnaker.pyNN.models.spike_source.spike_source.poisson, 246
 spynnaker.pyNN.models.spike_source.spike_source.poisson_machine_vertex, 247
 spynnaker.pyNN.models.spike_source.spikes, 248
 spynnaker.pyNN.models.spike_source.spikes.pyNN_utilities, 249
 spynnaker.pyNN.models.utility_models, 264
 spynnaker.pyNN.models.utility_models.delays, 258
 spynnaker.pyNN.models.utility_models.delays.del_block, 254
 spynnaker.pyNN.models.utility_models.delays.del_extension_machine_vertex, 255
 spynnaker.pyNN.models.utility_models.delays.del_extension_vertex, 255
 spynnaker.pyNN.models.utility_models.delays.del_generator_data, 258
 spynnaker.pyNN.models.utility_models.spike_injector, 263
 spynnaker.pyNN.models.utility_models.spike_injector.spike_injector, 261
 spynnaker.pyNN.models.utility_models.spike_injector.spike_injector_vertex, 262
 spynnaker.pyNN.models.utility_models.synapse_expander, 264
 spynnaker.pyNN.models.utility_models.synapse_expander.synapse_expander, 264
 spynnaker.pyNN.overridden_pacman_functions, 269
 spynnaker.pyNN.overridden_pacman_functions.graph_edge_filter, 269
 spynnaker.pyNN.overridden_pacman_functions.graph_edge_weight_updater, 269
 spynnaker.pyNN.overridden_pacman_functions.spynnaker_data_specification_writer, 269
 spynnaker.pyNN.protocols, 275
 spynnaker.pyNN.protocols.munich_io_ethernet_protocol, 270
 spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol, 270
 spynnaker.pyNN.spynnaker_external_device_plugin_manager, 290
 spynnaker.pyNN.spynnaker_simulator_interface, 293
 spynnaker.pyNN.utilities, 288
 spynnaker.pyNN.utilities.constants, 283
 spynnaker.pyNN.utilities.extracted_data, 284
 spynnaker.pyNN.utilities.fake_HBP_Portal_machine_parameters, 284
 spynnaker.pyNN.utilities.random_stats, 280
 spynnaker.pyNN.utilities.random_stats.abstract_random_stats, 280
 spynnaker.pyNN.utilities.ranged, 282
 spynnaker.pyNN.utilities.ranged.spynnaker_ranged_data_specification_writer, 281
 spynnaker.pyNN.utilities.ranged.spynnaker_ranged_data_specification_writer, 281
 spynnaker.pyNN.utilities.reports, 284
 spynnaker.pyNN.utilities.running_stats, 284
 spynnaker.pyNN.utilities.spynnaker_connection_holder, 285
 spynnaker.pyNN.utilities.spynnaker_failed_state, 285
 spynnaker.pyNN.utilities.spynnaker_neuron_network_configuration, 285
 spynnaker.pyNN.utilities.spynnaker_synaptic_matrix, 285
 spynnaker.pyNN.utilities.utility_calls, 285
 spynnaker.spike_checker, 294

A

- AbstractAcceptsIncomingSynapses (class in `spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh_neuron_model_izh`), 58
- `a` (`spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh_neuron_model_izh` attribute), 143
- `a` (`spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh` attribute), 147
- `A3_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceAdditiveTriplet` attribute), 161
- `A3_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet` attribute), 164
- `A3_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceAdditiveTriplet` attribute), 161
- `A3_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet` attribute), 164
- `A_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus` attribute), 159
- `A_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractHasAPlusAMinus` attribute), 162
- `A_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus` attribute), 159
- `A_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractHasAPlusAMinus` attribute), 162
- `ABSOLUTE_2_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload` attribute), 274
- `ABSOLUTE_2_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.RetinaPayload` attribute), 279
- `ABSOLUTE_3_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload` attribute), 274
- `ABSOLUTE_3_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.RetinaPayload` attribute), 279
- `ABSOLUTE_4_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload` attribute), 274
- `ABSOLUTE_4_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.RetinaPayload` attribute), 279
- `AbstractAcceptsIncomingSynapses` (class in `spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh_neuron_model_izh`), 58
- `AbstractAdditionalInput` (class in `spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input`), 110
- `AbstractAdditionalInput` (class in `spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input`), 109
- `AbstractConnector` (class in `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector`), 87
- `AbstractConnector` (class in `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector`), 71
- `AbstractContainsUnits` (class in `spynnaker.pyNN.models.abstract_models.abstract_contains_units`), 59
- `AbstractContainsUnits` (class in `spynnaker.pyNN.models.abstract_models.abstract_contains_units`), 55
- `AbstractElimination` (class in `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination`), 166
- `AbstractElimination` (class in `spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination`), 165
- `AbstractEthernetController` (class in `spynnaker.pyNN.external_devices_models`), 47
- `AbstractEthernetController` (class in `spynnaker.pyNN.external_devices_models.abstract_ethernet_controller`), 37
- `AbstractEthernetSensor` (class in `spynnaker.pyNN.external_devices_models`), 47
- `AbstractEthernetSensor` (class in `spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor`), 37
- `AbstractEthernetTranslator` (class in `spynnaker.pyNN.external_devices_models`), 47

<code>naker.pyNN.external_devices_models</code>), 48	128
<code>AbstractEthernetTranslator</code> (class in <code>slynaker.pyNN.external_devices_models.abstract_ethernet_translator</code>), 38	<code>AbstractNeuronImpl</code> (class in <code>slynaker.pyNN.models.neuron.implementations.abstract_neuron_impl</code>), 116
<code>AbstractFilterableEdge</code> (class in <code>slynaker.pyNN.models.abstract_models</code>), 59	<code>AbstractNeuronModel</code> (class in <code>slynaker.pyNN.models.neuron.neuron_models</code>), 146
<code>AbstractFilterableEdge</code> (class in <code>slynaker.pyNN.models.abstract_models.abstract_filterable_edge</code>), 56	<code>AbstractNeuronModel</code> (class in <code>slynaker.pyNN.models.neuron.neuron_models.abstract_neuron_model</code>), 142
<code>AbstractFormation</code> (class in <code>slynaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formations</code>), 168	<code>AbstractPartnerSelection</code> (class in <code>slynaker.pyNN.models.common</code>), 66
<code>AbstractFormation</code> (class in <code>slynaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formations.abstract_neuron_formation</code>), 167	<code>AbstractNeuronRecordable</code> (class in <code>slynaker.pyNN.models.neuron.implementations.abstract_neuron_recordable</code>), 61
<code>AbstractGenerateConnectorOnMachine</code> (class in <code>slynaker.pyNN.models.neural_projections.connectors</code>), 87	<code>AbstractPartnerSelection</code> (class in <code>slynaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection</code>), 171
<code>AbstractGenerateConnectorOnMachine</code> (class in <code>slynaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine</code>), 72	<code>AbstractPartnerSelection</code> (class in <code>slynaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.abstract_partner_selection</code>), 170
<code>AbstractGenerateOnMachine</code> (class in <code>slynaker.pyNN.models.neuron.synapse_dynamics</code>), 187	<code>AbstractGenerateOnMachine</code> (class in <code>slynaker.pyNN.models.neuron.synapse_dynamics</code>), 188
<code>AbstractGenerateOnMachine</code> (class in <code>slynaker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine</code>), 172	<code>AbstractPlasticSynapseDynamics</code> (class in <code>slynaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics</code>), 173
<code>AbstractHasAPlusAMinus</code> (class in <code>slynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus</code>), 162	<code>AbstractPopulationInitializable</code> (class in <code>slynaker.pyNN.models.abstract_models</code>), 59
<code>AbstractHasAPlusAMinus</code> (class in <code>slynaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus</code>), 159	<code>AbstractPopulationInitializable</code> (class in <code>slynaker.pyNN.models.abstract_models.abstract_population_initializable</code>), 56
<code>AbstractInputType</code> (class in <code>slynaker.pyNN.models.neuron.input_types</code>), 134	<code>AbstractPopulationSettable</code> (class in <code>slynaker.pyNN.models.abstract_models</code>), 60
<code>AbstractInputType</code> (class in <code>slynaker.pyNN.models.neuron.input_types.abstract_input_type</code>), 130	<code>AbstractPopulationSettable</code> (class in <code>slynaker.pyNN.models.abstract_models.abstract_population_settable</code>), 57
<code>AbstractMasterPopTableFactory</code> (class in <code>slynaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory</code>), 138	<code>AbstractPopulationVertex</code> (class in <code>slynaker.pyNN.models.neuron</code>), 235
<code>AbstractMulticastControllableDevice</code> (class in <code>slynaker.pyNN.external_devices_models</code>), 48	<code>AbstractPopulationVertex</code> (class in <code>slynaker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_factory</code>), 223
<code>AbstractMulticastControllableDevice</code> (class in <code>slynaker.pyNN.external_devices_models.abstract_multicast_controllable_device</code>), 38	<code>AbstractPushBotOutputDevice</code> (class in <code>slynaker.pyNN.external_devices_models.push_bot</code>), 36
<code>AbstractNeuronImpl</code> (class in <code>slynaker.pyNN.models.neuron.implementations</code>),	<code>AbstractPushBotOutputDevice</code> (class in <code>slynaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device</code>), 36
	<code>AbstractPushBotRetinaDevice</code> (class in <code>slynaker.pyNN.external_devices_models.push_bot</code>), 36

AbstractPushBotRetinaDevice (class in spyn- naker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device), 36	(class in spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics), 173
AbstractPyNNModel (class in spyn- naker.pyNN.models.abstract_pynn_model), 264	AbstractSynapseDynamics (class in spyn- naker.pyNN.models.neuron.synapse_dynamics), 186
AbstractPyNNNeuronModel (class in spyn- naker.pyNN.models.neuron), 243	AbstractSynapseDynamics (class in spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics), 174
AbstractPyNNNeuronModel (class in spyn- naker.pyNN.models.neuron.abstract_pynn_neuron_model), 229	AbstractSynapseDynamicsStructural (class in spyn- naker.pyNN.models.neuron.synapse_dynamics), 193
AbstractPyNNNeuronModelStandard (class in spynnaker.pyNN.models.neuron), 244	AbstractSynapseDynamicsStructural (class in spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics), 175
AbstractPyNNNeuronModelStandard (class in spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model_standard), 230	AbstractSynapseIO (class in spyn- naker.pyNN.models.neuron.synapse_io), 201
AbstractRandomStats (class in spyn- naker.pyNN.utilities.random_stats), 280	AbstractSynapseIO (class in spyn- naker.pyNN.models.neuron.synapse_io.abstract_synapse_io), 200
AbstractRandomStats (class in spyn- naker.pyNN.utilities.random_stats.abstract_random_stats), 280	AbstractSynapseStructure (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.synapse_structure), 151
AbstractReadParametersBeforeSet (class in spynnaker.pyNN.models.abstract_models), 60	AbstractSynapseStructure (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure), 150
AbstractReadParametersBeforeSet (class in spynnaker.pyNN.models.abstract_models.abstract_read_parameters_before_set), 57	AbstractSynapseType (class in spyn- naker.pyNN.models.neuron.synapse_types), 211
AbstractSettable (class in spyn- naker.pyNN.models.abstract_models), 60	AbstractSynapseType (class in spyn- naker.pyNN.models.neuron.synapse_types.abstract_synapse_type), 202
AbstractSettable (class in spyn- naker.pyNN.models.abstract_models.abstract_settable), 58	AbstractThresholdType (class in spyn- naker.pyNN.models.neuron.threshold_types), 221
AbstractSpikeRecordable (class in spyn- naker.pyNN.models.common), 67	AbstractThresholdType (class in spyn- naker.pyNN.models.neuron.threshold_types.abstract_threshold_type), 218
AbstractSpikeRecordable (class in spyn- naker.pyNN.models.common.abstract_spike_recordable), 62	AbstractTimingDependence (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 156
AbstractSpiNNakerCommon (class in spyn- naker.pyNN.abstract_spinnaker_common), 288	AbstractTimingDependence (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence), 152
AbstractStandardNeuronComponent (class in spynnaker.pyNN.models.neuron.implementations), 123	AbstractWeightDependence (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 162
AbstractStandardNeuronComponent (class in spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component), 118	AbstractWeightDependence (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence), 159
AbstractStaticSynapseDynamics (class in spynnaker.pyNN.models.neuron.synapse_dynamics), 188	AbstractWeightUpdatable (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.weight_updatable), 162
AbstractStaticSynapseDynamics	

<code>naker.pyNN.models.abstract_models), 61</code>	<code>add_generator_data()</code>	<code>(spyn-</code>
<code>AbstractWeightUpdatable (class in spyn-</code>	<code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code>	
<code>naker.pyNN.models.abstract_models.abstract_weight_updatable), 260</code>		
<code>58</code>	<code>add_init_callback()</code>	<code>(spyn-</code>
<code>activate_live_output_for()</code>	<code>(spyn-</code>	<code>naker.pyNN.connections.spynnaker_poisson_control_connection.</code>
<code>naker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager</code>		
<code>static method), 290</code>	<code>add_init_callback()</code>	<code>(spyn-</code>
<code>activate_live_output_to()</code>	<code>(spyn-</code>	<code>naker.pyNN.connections.SpynnakerPoissonControlConnection</code>
<code>naker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager</code>		
<code>static method), 291</code>	<code>add_item()</code>	<code>(spynnaker.pyNN.utilities.running_stats.RunningStats</code>
<code>actual_sdram_usage</code>	<code>(spyn-</code>	<code>method), 284</code>
<code>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon</code>		
<code>attribute), 181</code>	<code>naker.pyNN.utilities.running_stats.RunningStats</code>	
<code>actual_sdram_usage</code>	<code>(spyn-</code>	<code>method), 284</code>
<code>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon</code>		
<code>attribute), 194</code>	<code>naker.pyNN.external_devices_models.threshold_type_multicast_d</code>	
<code>add_application_vertex()</code>	<code>(spyn-</code>	<code>method), 46</code>
<code>naker.pyNN.abstract_spinnaker_common.AbstractSpinnakerCommon</code>		
<code>method), 288</code>	<code>naker.pyNN.external_devices_models.ThresholdTypeMulticastDev</code>	
<code>add_application_vertex()</code>	<code>(spyn-</code>	<code>method), 53</code>
<code>naker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager</code>		
<code>static method), 292</code>	<code>naker.pyNN.models.neuron.additional_inputs.additional_input_co</code>	
<code>add_command_container()</code>	<code>(spyn-</code>	<code>method), 109</code>
<code>naker.pyNN.connections.ethernet_command_connection.EthernetCommandConnection</code>		
<code>method), 4</code>	<code>naker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2</code>	
<code>add_command_container()</code>	<code>(spyn-</code>	<code>method), 110</code>
<code>naker.pyNN.connections.EthernetCommandConnection</code>		
<code>method), 7</code>	<code>add_parameters()</code>	<code>(spyn-</code>
<code>add_connections()</code>	<code>(spyn-</code>	<code>method), 116</code>
<code>naker.pyNN.models.neuron.connection_holder.ConnectionHolder</code>		
<code>method), 231</code>	<code>add_parameters()</code>	<code>(spyn-</code>
<code>add_connections()</code>	<code>(spyn-</code>	<code>method), 118</code>
<code>naker.pyNN.models.neuron.ConnectionHolder</code>		
<code>method), 241</code>	<code>add_parameters()</code>	<code>(spyn-</code>
<code>add_database_socket_address()</code>	<code>(spyn-</code>	<code>method), 128</code>
<code>naker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager</code>		
<code>static method), 292</code>	<code>naker.pyNN.models.neuron.implementations.AbstractStandardNeu</code>	
<code>add_delay()</code>	<code>(spyn-</code>	<code>method), 123</code>
<code>naker.pyNN.models.utility_models.delays.delay_block.DelayBlock</code>		
<code>method), 254</code>	<code>add_parameters()</code>	<code>(spyn-</code>
<code>add_delay()</code>	<code>(spyn-</code>	<code>method), 120</code>
<code>naker.pyNN.models.utility_models.delays.DelayBlock</code>		
<code>method), 258</code>	<code>add_parameters()</code>	<code>(spyn-</code>
<code>add_delays()</code>	<code>(spyn-</code>	<code>method), 126</code>
<code>naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex</code>		
<code>method), 256</code>	<code>naker.pyNN.models.neuron.input_types.input_type_conductance.I</code>	
<code>add_delays()</code>	<code>(spyn-</code>	<code>method), 130</code>
<code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code>		
<code>method), 259</code>	<code>add_parameters()</code>	<code>(spyn-</code>
<code>add_edge()</code>	<code>(spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager</code>	
<code>static method), 292</code>	<code>add_parameters()</code>	<code>(spyn-</code>
<code>add_generator_data()</code>	<code>(spyn-</code>	<code>naker.pyNN.models.neuron.input_types.input_type_current_semd.</code>
<code>naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex</code>		
<code>method), 256</code>	<code>add_parameters()</code>	<code>(spyn-</code>

`naker.pyNN.models.neuron.input_types.InputTypeConductance`
`method), 134`

`naker.pyNN.models.neuron.threshold_types.threshold_type_static`
`method), 220`

`add_parameters()` (spyn- `add_parameters()` (spyn-
`naker.pyNN.models.neuron.input_types.InputTypeCurrent` `naker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass`
`method), 135` `method), 222`

`add_parameters()` (spyn- `add_parameters()` (spyn-
`naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMDr` `naker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic`
`method), 136` `method), 221`

`add_parameters()` (spyn- `add_pause_stop_callback()` (spyn-
`naker.pyNN.models.neuron.neuron_models.neuron_model_integrated_and_fire` `naker.pyNN.models.neuron_models.neuron_model_integrated_and_fire`
`method), 143` `method), 6`

`add_parameters()` (spyn- `add_pause_stop_callback()` (spyn-
`naker.pyNN.models.neuron.neuron_models.neuron_model_integrated_and_fire` `naker.pyNN.models.neuron_models.neuron_model_integrated_and_fire`
`method), 145` `method), 9`

`add_parameters()` (spyn- `add_payload_logic_to_current_output()` (spyn-
`naker.pyNN.models.neuron.neuron_models.NeuronModelLzh` `naker.pyNN.models.neuron_models.NeuronModelLzh`
`method), 147` `method), 271`

`add_parameters()` (spyn- `add_payload_logic_to_current_output()` (spyn-
`naker.pyNN.models.neuron.neuron_models.NeuronModelLzh` `naker.pyNN.models.neuron_models.NeuronModelLzh`
`method), 148` `method), 276`

`add_parameters()` (spyn- `add_payload_logic_to_current_output_key` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_alpha` `naker.pyNN.models.neuron.synapse_types.synapse_type_alpha`
`method), 203` `attribute), 271`

`add_parameters()` (spyn- `add_payload_logic_to_current_output_key` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_delta` `naker.pyNN.models.neuron.synapse_types.synapse_type_delta`
`method), 204` `attribute), 276`

`add_parameters()` (spyn- `add_placement_constraint()` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential` `naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential`
`method), 206` `method), 266`

`add_parameters()` (spyn- `add_poisson_label()` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_exponential` `naker.pyNN.models.neuron.synapse_types.synapse_type_exponential`
`method), 207` `method), 6`

`add_parameters()` (spyn- `add_poisson_label()` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_semdr` `naker.pyNN.models.neuron.synapse_types.synapse_type_semdr`
`method), 209` `method), 9`

`add_parameters()` (spyn- `add_poisson_live_rate_control()` (spyn-
`naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha` `naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha`
`method), 215` `static method), 292`

`add_parameters()` (spyn- `add_population()` (spyn-
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta` `naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta`
`method), 214` `method), 288`

`add_parameters()` (spyn- `add_pre_run_connection_holder()` (spyn-
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential` `naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential`
`method), 211` `method), 55`

`add_parameters()` (spyn- `add_pre_run_connection_holder()` (spyn-
`naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential` `naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential`
`method), 213` `method), 58`

`add_parameters()` (spyn- `add_pre_run_connection_holder()` (spyn-
`naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMDr` `naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMDr`
`method), 217` `method), 224`

`add_parameters()` (spyn- `add_pre_run_connection_holder()` (spyn-
`naker.pyNN.models.neuron.threshold_types.threshold_type_maass` `naker.pyNN.models.neuron.threshold_types.threshold_type_maass`
`method), 219` `method), 235`

`add_parameters()` (spyn- `add_pre_run_connection_holder()` (spyn-

`naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta` (class in `naker.pyNN.models.neuron.synapse_types`), 214
`naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generator` (class in `naker.pyNN.models.neuron.master_pop_table_generators`), 139
`add_state_variables()` (method in `naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta`), 214
`add_state_variables()` (method in `naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential`), 211
`add_state_variables()` (method in `naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential`), 213
`add_state_variables()` (method in `naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMI`), 217
`add_state_variables()` (method in `naker.pyNN.models.neuron.threshold_types.threshold_type`), 219
`add_state_variables()` (method in `naker.pyNN.models.neuron.threshold_types.threshold_type`), 220
`add_state_variables()` (method in `naker.pyNN.models.neuron.threshold_types.ThresholdTypeMakesSynapseStable`), 222
`add_state_variables()` (method in `naker.pyNN.models.neuron.threshold_types.ThresholdTypeStake`), 221
`add_synapse_information()` (method in `naker.pyNN.models.neural_projections.delayed_application_edges.DelayedApplicationEdges`), 102
`add_synapse_information()` (method in `naker.pyNN.models.neural_projections.DelayedApplicationEdges`), 105
`add_synapse_information()` (method in `naker.pyNN.models.neural_projections.projection_application_edges.ProjectionApplicationEdges`), 103
`add_synapse_information()` (method in `naker.pyNN.models.neural_projections.ProjectionApplicationEdges`), 106
`add_translator()` (method in `naker.pyNN.connections.ethernet_control_connection.EthernetControlConnection`), 4
`add_translator()` (method in `naker.pyNN.connections.EthernetControlConnection`), 7
`AdditionalInputCa2Adaptive` (class in `naker.pyNN.models.neuron.additional_inputs`), 110
`AdditionalInputCa2Adaptive` (class in `naker.pyNN.models.neuron.additional_inputs.additional_input_models`), 109
`ADDRESS_LIST_DTYPE` (attribute in `naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generator`), 139
`ADDRESS_LIST_DTYPE` (attribute in `naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerator`), 140
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.abstract_connector`), 73
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.all_to_all_connector`), 74
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.AllToAllConnector`), 89
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.distance_dependent_connector`), 76
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.DistanceDependentConnector`), 91
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.fixed_number_connector`), 77
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.FixedNumberConnector`), 92
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.index_based_projection`), 81
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.index_based_projection`), 88
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.index_based_projection`), 88
`ADDRESS_MASK` (attribute in `naker.pyNN.models.neural_projections.connectors.index_based_projection`), 88

[naker.pyNN.models.neural_projections.connectors.ArrayConnector](#), (spyn-
[73](#) [naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES](#)
[alpha \(spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011.TimingDependenceVogels](#)
[attribute\), 155](#) [BASE_SIZE \(spynnaker.pyNN.models.neuron.generator_data.GeneratorD](#)
[alpha \(spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011.TimingDependenceVogels2011](#)
[attribute\), 159](#) [BASE_SIZE \(spynnaker.pyNN.models.utility_models.delays.delay_generat](#)
[ArbitraryFPGADevice \(class in spyn-](#) [attribute\), 258](#)
[naker.pyNN.external_devices_models\), 48](#) [BASIC_MALLOC_USAGE \(spyn-](#)
[ArbitraryFPGADevice \(class in spyn-](#) [naker.pyNN.models.neuron.abstract_population_vertex.AbstractP](#)
[naker.pyNN.external_devices_models.arbitrary_fpga_device\), 224](#) [attribute\), 224](#)
[39](#) [BASIC_MALLOC_USAGE \(spyn-](#)
[are_weights_signed\(\) \(spyn-](#) [naker.pyNN.models.neuron.AbstractPopulationVertex](#)
[naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics](#)
[method\), 174](#) [bias_values\(\) \(spyn-](#)
[are_weights_signed\(\) \(spyn-](#) [naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich](#)
[naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics](#)
[method\), 186](#) [bias_values\(\) \(spyn-](#)
[are_weights_signed\(\) \(spyn-](#) [naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic](#)
[method\), 176](#) [bias_values_key \(spyn-](#)
[are_weights_signed\(\) \(spyn-](#) [naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich](#)
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP](#)
[method\), 178](#) [bias_values_key \(spyn-](#)
[are_weights_signed\(\) \(spyn-](#) [naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
[naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic](#)
[method\), 189](#) [binary_name \(spyn-](#)
[are_weights_signed\(\) \(spyn-](#) [naker.pyNN.models.neuron.implementations.abstract_neuron_impl](#)
[naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP](#)
[method\), 191](#) [binary_name \(spyn-](#)
[ArrayConnector \(class in spyn-](#) [naker.pyNN.models.neuron.implementations.AbstractNeuronImpl](#)
[naker.pyNN.models.neural_projections.connectors\), attribute\), 128](#)
[89](#) [binary_name \(spyn-](#)
[ArrayConnector \(class in spyn-](#) [naker.pyNN.models.neuron.implementations.neuron_impl_standar](#)
[naker.pyNN.models.neural_projections.connectors.array_connector\), 120](#) [attribute\), 120](#)
[74](#) [binary_name \(spyn-](#)
[as_list\(\) \(spynnaker.pyNN.utilities.ranged.spynnaker_ranged_list.SpyNNakerRangedList](#) [naker.pyNN.models.neuron.implementations.NeuronImplStandard](#)
[static method\), 282](#) [attribute\), 126](#)
[as_list\(\) \(spynnaker.pyNN.utilities.ranged.SpyNNakerRangedList](#) [naker.pyNN.models.neuron.implementations.NeuronImplStandard](#)
[static method\), 283](#) [coordinate \(spyn-](#)
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Retina](#)
[attribute\), 274](#)
B [bits_per_coordinate \(spyn-](#)
[b \(spynnaker.pyNN.models.neuron.neuron_models.neuron_model_ich.NeuronModelIch](#) [naker.pyNN.protocols.RetinaKey attribute\),](#)
[attribute\), 143](#) [279](#)
[b \(spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIch](#) [BUFFER_OVERFLOW_COUNT \(spyn-](#)
[attribute\), 147](#) [naker.pyNN.models.neuron.population_machine_vertex.Population](#)
[backprop_delay \(spyn-](#) [attribute\), 233](#)
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP \(spyn-](#)
[attribute\), 178](#) [naker.pyNN.models.neuron.PopulationMachineVertex.EXTRA_PR](#)
[backprop_delay \(spyn-](#) [attribute\), 242](#)
[naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP](#)
[attribute\), 191](#) [\(spynnaker.pyNN.models.neuron.abstract_population_vertex.Abst](#)
[BALL_BALANCER \(spyn-](#) [attribute\), 224](#)
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES](#)
[attribute\), 271](#) [\(spynnaker.pyNN.models.neuron.AbstractPopulationVertex](#)

<i>attribute</i>), 235	<i>check_sampling_interval()</i> (in module <i>spynaker.pyNN.utilities.utility_calls</i>), 286
C	<i>check_sister_gsyn()</i> (in module <i>spynaker.pyNN.utilities.utility_calls</i>), 293
<i>c</i> (<i>spynaker.pyNN.models.neuron.neuron_models.neuron_model_izh_neuron_model_izh</i> <i>attribute</i>), 143	<i>clear_connection_cache()</i> (<i>spynaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses</i> <i>method</i>), 55
<i>c</i> (<i>spynaker.pyNN.models.neuron.neuron_models.NeuronModelIzh_neuron_model_izh</i> <i>attribute</i>), 147	<i>clear_connection_cache()</i> (<i>spynaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses</i> <i>method</i>), 58
<i>cdf()</i> (<i>spynaker.pyNN.utilities.random_stats.abstract_random_stats_abstract_random_stats</i> <i>method</i>), 280	<i>clear_connection_cache()</i> (<i>spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</i> <i>method</i>), 224
<i>cdf()</i> (<i>spynaker.pyNN.utilities.random_stats.AbstractRandomStats_abstract_random_stats</i> <i>method</i>), 280	<i>clear_connection_cache()</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_abstract_synapse_dynamics</i> <i>method</i>), 234
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_abstract_synapse_dynamics</i> <i>attribute</i>), 174	<i>clear_connection_cache()</i> (<i>spynaker.pyNN.models.neuron.AbstractPopulationVertex_abstract_population_vertex</i> <i>method</i>), 235
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics_abstract_synapse_dynamics</i> <i>attribute</i>), 186	<i>clear_connection_cache()</i> (<i>spynaker.pyNN.models.neuron.synaptic_manager.SynapticManager_synaptic_manager</i> <i>method</i>), 234
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic_synapse_dynamics_static</i> <i>attribute</i>), 176	<i>clear_connection_cache()</i> (<i>spynaker.pyNN.models.neuron.SynapticManager_synaptic_manager</i> <i>method</i>), 244
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP_synapse_dynamics_stdp</i> <i>attribute</i>), 178	<i>clear_recording()</i> (<i>spynaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable_abstract_neuron_recordable</i> <i>method</i>), 61
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStatic_synapse_dynamics_structural_static</i> <i>attribute</i>), 183	<i>clear_recording()</i> (<i>spynaker.pyNN.models.common.AbstractNeuronRecordable_abstract_neuron_recordable</i> <i>method</i>), 66
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic_synapse_dynamics_static</i> <i>attribute</i>), 189	<i>clear_recording()</i> (<i>spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex_abstract_population_vertex</i> <i>method</i>), 224
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP_synapse_dynamics_stdp</i> <i>attribute</i>), 191	<i>clear_recording()</i> (<i>spynaker.pyNN.models.neuron.AbstractPopulationVertex_abstract_population_vertex</i> <i>method</i>), 235
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic_synapse_dynamics_structural_static</i> <i>attribute</i>), 196	<i>clear_spike_recording()</i> (<i>spynaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable_abstract_spike_recordable</i> <i>method</i>), 62
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.synaptic_manager.SynapticManager_synaptic_manager</i> <i>attribute</i>), 234	<i>clear_spike_recording()</i> (<i>spynaker.pyNN.models.common.AbstractSpikeRecordable_abstract_spike_recordable</i> <i>method</i>), 67
<i>changes_during_run</i> (<i>spynaker.pyNN.models.neuron.SynapticManager_synaptic_manager</i> <i>attribute</i>), 241	<i>clear_spike_recording()</i> (<i>spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex_abstract_population_vertex</i> <i>method</i>), 224
<i>check_directory_exists_and_create_if_not()</i> (in module <i>spynaker.pyNN.utilities.utility_calls</i>), 285	<i>clear_spike_recording()</i> (<i>spynaker.pyNN.models.neuron.AbstractPopulationVertex_abstract_population_vertex</i> <i>method</i>), 236
<i>check_gsyn()</i> (in module <i>spynaker.gsyn_tools</i>), 293	<i>clear_spike_recording()</i> (<i>spynaker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex_spike_source_array_vertex</i> <i>method</i>), 245
<i>check_indexes()</i> (<i>spynaker.pyNN.models.common.neuron_recorder.NeuronRecorder_neuron_recorder</i> <i>method</i>), 64	<i>clear_spike_recording()</i> (<i>spynaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex_spike_source_poisson_vertex</i> <i>method</i>), 249
<i>check_indexes()</i> (<i>spynaker.pyNN.models.common.NeuronRecorder_neuron_recorder</i> <i>method</i>), 68	<i>clear_spike_recording()</i> (<i>spynaker.pyNN.models.utility_models.spike_injector.spike_injector_v</i> <i>method</i>), 249
<i>check_path_gsyn()</i> (in module <i>spynaker.gsyn_tools</i>), 293	

`method`), 162
`close()` (`spynnaker.pyNN.external_devices_models.push_bot.push_bot.PushBotWiFiConnection` (spyn-
`method`), 18
`close()` (`spynnaker.pyNN.external_devices_models.push_bot.push_bot.PushBotWiFiConnection` (spyn-
`method`), 25
`cm` (`spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAndFire`
`attribute`), 145
`cm` (`spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire`
`attribute`), 149
`column_names` (spyn-
`naker.pyNN.models.neural_projections.connectors.ColumnListConnectorFromListConnector` (spyn-
`attribute`), 80
`column_names` (spyn-
`naker.pyNN.models.neural_projections.connectors.ColumnListConnector` (class in spyn-
`attribute`), 95
`compute_statistics()` (spyn-
`naker.pyNN.models.neural_projections.connectors.kernel_connector.KernelConnector` (in module spyn-
`method`), 83
`compute_statistics()` (spyn-
`naker.pyNN.models.neural_projections.connectors.KernelConnector` (in module spyn-
`method`), 100
`conductance_based` (spyn-
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex` (in module spyn-
`attribute`), 225
`conductance_based` (spyn-
`naker.pyNN.models.neuron.AbstractPopulationVertex` (in module spyn-
`attribute`), 236
`conductance_based` (spyn-
`naker.pyNN.models.pynn_population_common.PyNNPopulationCommon` (class in spyn-
`attribute`), 266
`CONFIG_FILE_NAME` (spyn-
`naker.pyNN.abstract_spinnaker_common.AbstractSpinnakerCommon` (in module spyn-
`attribute`), 288
`configure_master_key()` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (spyn-
`method`), 271
`configure_master_key()` (spyn-
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (in module spyn-
`method`), 276
`configure_master_key_key` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (spyn-
`attribute`), 271
`configure_master_key_key` (spyn-
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (in module spyn-
`attribute`), 276
`conn_list` (`spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector`
`attribute`), 80
`conn_list` (`spynnaker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector`
`attribute`), 95
`ConnectionHolder` (class in spyn-
`naker.pyNN.models.neuron`), 241
`ConnectionHolder` (class in spyn-
`naker.pyNN.models.neuron.connection_holder`), 231

<i>method</i>), 51	<i>method</i>), 79	
<code>create_machine_vertex()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neural_projections.connectors.FixedNumberBlock</code>	<code>naker.pyNN.models.neural_projections.connectors.FixedNumberBlock</code>
<i>method</i>), 225	<i>method</i>), 92	
<code>create_machine_vertex()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neuron.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neural_projections.connectors.FixedNumberBlock</code>	<code>naker.pyNN.models.neural_projections.connectors.FixedNumberBlock</code>
<i>method</i>), 236	<i>method</i>), 93	
<code>create_machine_vertex()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</code>	<code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityBlock</code>	<code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityBlock</code>
<i>method</i>), 250	<i>method</i>), 94	
<code>create_machine_vertex()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex</code>	<code>naker.pyNN.models.neural_projections.connectors.FromListConnections</code>	<code>naker.pyNN.models.neural_projections.connectors.FromListConnections</code>
<i>method</i>), 256	<i>method</i>), 80	
<code>create_machine_vertex()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code>	<code>naker.pyNN.models.neural_projections.connectors.FromListConnections</code>	<code>naker.pyNN.models.neural_projections.connectors.FromListConnections</code>
<i>method</i>), 260	<i>method</i>), 95	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.abstract_neural_projection_connectors.AbstractNeuralProjectionConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.index_based_projection_connectors.IndexBasedProjectionConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.index_based_projection_connectors.IndexBasedProjectionConnectors</code>
<i>method</i>), 71	<i>method</i>), 81	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.AbstractNeuralProjectionConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.IndexBasedProjectionConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.IndexBasedProjectionConnectors</code>
<i>method</i>), 87	<i>method</i>), 96	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.all_to_all_connectors.AllToAllConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.kernel_connectors.KernelConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.kernel_connectors.KernelConnectors</code>
<i>method</i>), 74	<i>method</i>), 83	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.AllToAllConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.KernelConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.KernelConnectors</code>
<i>method</i>), 89	<i>method</i>), 100	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.array_connectors.ArrayConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.multipse_connectors.MultipseConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.multipse_connectors.MultipseConnectors</code>
<i>method</i>), 74	<i>method</i>), 84	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.ArrayConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.MultipseConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.MultipseConnectors</code>
<i>method</i>), 90	<i>method</i>), 97	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.csa_connectors.CSAConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.one_to_one_connectors.OneToOneConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.one_to_one_connectors.OneToOneConnectors</code>
<i>method</i>), 75	<i>method</i>), 85	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.CSAConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.OneToOneConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.OneToOneConnectors</code>
<i>method</i>), 90	<i>method</i>), 98	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connectors.DistanceDependentProbabilityConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.small_world_connectors.SmallWorldConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.small_world_connectors.SmallWorldConnectors</code>
<i>method</i>), 76	<i>method</i>), 86	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_synaptic_block()</code>
<code>naker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.SmallWorldConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.SmallWorldConnectors</code>
<i>method</i>), 91	<i>method</i>), 99	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_vertex()</code>	(<i>spyn-</i> <code>create_vertex()</code>
<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>
<i>method</i>), 77	<i>method</i>), 39	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_vertex()</code>	(<i>spyn-</i> <code>create_vertex()</code>
<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>
<i>method</i>), 78	<i>method</i>), 49	
<code>create_synaptic_block()</code>	(<i>spyn-</i> <code>create_vertex()</code>	(<i>spyn-</i> <code>create_vertex()</code>
<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_connectors.FixedNumberConnectors</code>

method), 264
create_vertex() (spyn- DEFAULT_F_REW (spyn-
naker.pyNN.models.neuron.abstract_pynn_neuron_model.AbstractPyNNNeuronModel.synapse_dynamics.synapse_dynamics
method), 229 attribute), 181
create_vertex() (spyn- DEFAULT_F_REW (spyn-
naker.pyNN.models.neuron.AbstractPyNNNeuronModel naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics
method), 243 attribute), 194
create_vertex() (spyn- DEFAULT_INITIAL_DELAY (spyn-
naker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics
method), 244 attribute), 181
create_vertex() (spyn- DEFAULT_INITIAL_DELAY (spyn-
naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics
method), 246 attribute), 194
create_vertex() (spyn- default_initial_values (spyn-
naker.pyNN.models.spike_source.spike_source_poisson_variable.SpikeSourcePoissonVariable naker.pyNN.models.munich_spinnaker_link_mo
method), 248 attribute), 44
create_vertex() (spyn- default_initial_values (spyn-
naker.pyNN.models.spike_source.SpikeSourceArray naker.pyNN.external_devices_models.MunichMotorDevice
method), 253 attribute), 51
create_vertex() (spyn- default_initial_values (spyn-
naker.pyNN.models.spike_source.SpikeSourcePoisson naker.pyNN.models.abstract_pynn_model.AbstractPyNNModel
method), 253 attribute), 264
create_vertex() (spyn- default_initial_values (spyn-
naker.pyNN.models.spike_source.SpikeSourcePoissonVariable naker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_ista.EIFC
method), 254 attribute), 111
create_vertex() (spyn- default_initial_values (spyn-
naker.pyNN.models.utility_models.spike_injector.spike_injector.SpikeInjector naker.pyNN.models.neuron.builds.EIFConductanceAlphaPopulati
method), 261 attribute), 114
create_vertex() (spyn- default_initial_values (spyn-
naker.pyNN.models.utility_models.spike_injector.SpikeInjector naker.pyNN.models.neuron.builds.hh_cond_exp.HHCondExp
method), 263 attribute), 112
CSAConnector (class in spyn- default_initial_values (spyn-
naker.pyNN.models.neural_projections.connectors), naker.pyNN.models.neuron.builds.HHCondExp
90 attribute), 114
CSAConnector (class in spyn- default_initial_values (spyn-
naker.pyNN.models.neural_projections.connectors.csa_connector.SpikeInjector naker.pyNN.models.neuron.builds.if_cond_alpha.IFCondAlpha
75 attribute), 112
CURRENT_TIMER_TIC (spyn- default_initial_values (spyn-
naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex.EXTRA_PROPERTIES_DATA_ENTRIES
attribute), 233 attribute), 113
CURRENT_TIMER_TIC (spyn- default_initial_values (spyn-
naker.pyNN.models.neuron.PopulationMachineVertex.EXTRA_PROPERTIES_DATA_ENTRIES
attribute), 242 attribute), 114
D default_initial_values (spyn-
naker.pyNN.models.neuron.builds.IFFacetsConductancePopulatio
d (spynnaker.pyNN.models.neuron.neuron_models.neuron_model.IzhikevichModel default_initial_values() (in module spyn-
attribute), 144 attribute), 265
d (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh naker.pyNN.models.defaults), 265
attribute), 147 DEFAULT_INITIAL_WEIGHT (spyn-
d_expression (spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics
naker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbability
attribute), 76 attribute), 194
d_expression (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics
naker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector attribute), 194

default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.munich_spinnaker_link</code>	<code>naker.pyNN.models.munich_motor_device.HHCondExp</code>	<code>attribute</code>), 44	<code>attribute</code>), 114
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.munich_spinnaker_link</code>	<code>naker.pyNN.models.munich_retina_device.IFCondAlpha</code>	<code>attribute</code>), 45	<code>attribute</code>), 112
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.MunichMotorDevice</code>	<code>naker.pyNN.models.neuron.builds.if_facets_hardware1.IFFacetsC</code>	<code>attribute</code>), 51	<code>attribute</code>), 113
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.MunichRetinaDevice</code>	<code>naker.pyNN.models.neuron.builds.IFCondAlpha</code>	<code>attribute</code>), 53	<code>attribute</code>), 114
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 30	<code>attribute</code>), 115
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 30	<code>attribute</code>), 153
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 31	<code>attribute</code>), 154
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 31	<code>attribute</code>), 155
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 32	<code>attribute</code>), 157
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 33	<code>attribute</code>), 158
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 34	<code>attribute</code>), 159
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 34	<code>attribute</code>), 161
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 35	<code>attribute</code>), 164
default_parameters	(spyn-	default_parameters	(spyn-
<code>naker.pyNN.external_devices_models.push_bot.push_bot_spi</code>	<code>naker.pyNN.models.neuron.builds.IFFacetsCondAlpha</code>	<code>attribute</code>), 35	<code>attribute</code>), 262
default_parameters	(spyn-	default_parameters()	(in module spyn-
<code>naker.pyNN.models.abstract_pynn_model.AbstractPyNNModel</code>	<code>naker.pyNN.models.defaults</code>	<code>attribute</code>), 264	<code>attribute</code>), 265
default_parameters	(spyn-	default_population_parameters	(spyn-
<code>naker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_ista</code>	<code>naker.pyNN.models.neuron.abstract_pynn_neuron_model.Abstract</code>	<code>attribute</code>), 111	<code>attribute</code>), 230
default_parameters	(spyn-	default_population_parameters	(spyn-
<code>naker.pyNN.models.neuron.builds.EIFConductanceAlphaPopu</code>	<code>naker.pyNN.models.neuron.AbstractPyNNNeuronModel</code>	<code>attribute</code>), 114	<code>attribute</code>), 244
default_parameters	(spyn-	default_population_parameters	(spyn-
<code>naker.pyNN.models.neuron.builds.hh_cond_exp.HHCondExp</code>	<code>attribute</code>), 112		

[naker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray](#) (class in [spynnaker.pyNN.models.neural_projections](#)), 105
[attribute](#)), 244
[DelayAfferentApplicationEdge](#) (class in [spynnaker.pyNN.models.neural_projections.delay_afferent_application](#))
[default_population_parameters](#) (spynnaker.pyNN.models.neural_projections.delay_afferent_application.
[naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson](#)
[attribute](#)), 247
[DelayAfferentMachineEdge](#) (class in [spynnaker.pyNN.models.neural_projections](#)), 105
[default_population_parameters](#) (spynnaker.pyNN.models.neural_projections.
[naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson](#) (class in [spynnaker.pyNN.models.neural_projections.delay_afferent_machine_e](#)
[attribute](#)), 248
[101](#)
[default_population_parameters](#) (spynnaker.pyNN.models.neural_projections.
[naker.pyNN.models.spike_source.SpikeSourceArray](#) (class in [spynnaker.pyNN.models.utility_models.delays](#)),
[attribute](#)), 253
[default_population_parameters](#) (spynnaker.pyNN.models.utility_models.delays.
[naker.pyNN.models.spike_source.SpikeSourcePoisson](#) (class in [spynnaker.pyNN.models.utility_models.delays.delay_block](#)),
[attribute](#)), 253
[258](#)
[default_population_parameters](#) (spynnaker.pyNN.models.utility_models.delays.
[naker.pyNN.models.spike_source.SpikeSourcePoisson](#) (class in [spynnaker.pyNN.models.utility_models.delays.delay_block](#)),
[attribute](#)), 254
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#) (class in [spynnaker.pyNN.models.neuron.synapse_io](#)), 200
[default_population_parameters](#) (spynnaker.pyNN.models.neuron.synapse_io.
[naker.pyNN.models.utility_models.spike_injector.SpikeInjector](#) (class in [spynnaker.pyNN.models.utility_models](#)), 262
[attribute](#)), 262
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#) (class in [spynnaker.pyNN.models.neuron.synapse_io](#)), 200
[default_population_parameters](#) (spynnaker.pyNN.models.neuron.synapse_io.
[naker.pyNN.models.utility_models.spike_injector.SpikeInjector](#) (class in [spynnaker.pyNN.models.utility_models](#)), 263
[attribute](#)), 263
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#) (class in [spynnaker.pyNN.models.neuron.synapse_io](#)), 200
[DEFAULT_S_MAX](#) (spynnaker.pyNN.models.neuron.synapse_io.
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon](#) (class in [spynnaker.pyNN.models.neuron.synapse_dynamics](#)), 181
[attribute](#)), 181
[naker.pyNN.models.neural_projections](#)), 105
[DEFAULT_S_MAX](#) (spynnaker.pyNN.models.neural_projections.
[naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon](#) (class in [spynnaker.pyNN.models.neural_projections.delayed_application_edge](#)
[attribute](#)), 194
[naker.pyNN.models.neural_projections.delayed_application_edge](#)
[defaults\(\)](#) (in module [spynnaker.pyNN.models.defaults](#)), 265
[DelayedMachineEdge](#) (class in [spynnaker.pyNN.models.neural_projections](#)), 102
[delay](#) ([spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics](#) (class in [spynnaker.pyNN.models.neuron.synapse_dynamics](#)), 174
[attribute](#)), 174
[106](#)
[delay](#) ([spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics](#) (class in [spynnaker.pyNN.models.neuron.synapse_dynamics](#)), 186
[attribute](#)), 186
[naker.pyNN.models.neural_projections.delayed_machine_edge](#)), 106
[delay](#) ([spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic](#) (class in [spynnaker.pyNN.models.neuron.synapse_dynamics](#)), 176
[attribute](#)), 176
[DelayExtensionException](#), 289
[delay](#) ([spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic](#) (class in [spynnaker.pyNN.models.neuron.synapse_dynamics](#)), 178
[attribute](#)), 178
[naker.pyNN.models.utility_models.delays](#)), 258
[delay](#) ([spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic](#) (class in [spynnaker.pyNN.models.neuron.synapse_dynamics](#)), 189
[attribute](#)), 189
[naker.pyNN.models.utility_models.delays.delay_extension_machine](#)
[delay](#) ([spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP](#) (class in [spynnaker.pyNN.models.neuron.synapse_dynamics](#)), 191
[attribute](#)), 191
[DelayExtensionMachineVertex.EXTRA_PROVENANCE_DATA_I](#)
[delay_block](#) (spynnaker.pyNN.models.utility_models.delays.
[naker.pyNN.models.utility_models.delays.delay_block.DelayBlock](#) (class in [spynnaker.pyNN.models.utility_models.delays](#)),
[attribute](#)), 254
[258](#)
[delay_block](#) (spynnaker.pyNN.models.utility_models.delays.
[naker.pyNN.models.utility_models.delays.DelayBlock](#) (class in [spynnaker.pyNN.models.utility_models.delays](#)),
[attribute](#)), 258
[naker.pyNN.models.utility_models.delays.delay_extension_machi](#)
[delay_edge](#) ([spynnaker.pyNN.models.neural_projections.projection_application_edge.ProjectionApplicationEdge](#) (class in [spynnaker.pyNN.models.neural_projections](#)), 103
[attribute](#)), 103
[DelayExtensionVertex](#) (class in [spynnaker.pyNN.models.utility_models](#)), 259
[delay_edge](#) ([spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge](#) (class in [spynnaker.pyNN.models.utility_models](#)),
[attribute](#)), 106
[259](#)
[DelayAfferentApplicationEdge](#) (class in [spynnaker.pyNN.models.neural_projections](#)), 105
[DelayExtensionVertex](#) (class in [spynnaker.pyNN.models.utility_models](#)), 259

`naker.pyNN.models.utility_models.delays.delay_extension_vertex` (spyn-
 255 `naker.pyNN.external_devices_models.abstract_multicast_controll`
 DelayGeneratorData (class in spyn- attribute), 38
`naker.pyNN.models.utility_models.delays.delay_generator_data` (spyn-
 258 `naker.pyNN.external_devices_models.AbstractMulticastControlla`
`delays (spynnaker.pyNN.models.neural_projections.synapse_information` (spyn-
 attribute), 104 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`delays (spynnaker.pyNN.models.neural_projections.SynapseInformation` (spyn-
 attribute), 107 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
 DELTA_TIMESTAMPS (spyn- device_control_max_value (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload` (spyn-
 attribute), 274 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
 DELTA_TIMESTAMPS (spyn- device_control_min_value (spyn-
`naker.pyNN.protocols.RetinaPayload` attribute), `naker.pyNN.external_devices_models.abstract_multicast_controll`
 279 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`dendritic_delay_fraction` (spyn- device_control_min_value (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics` (spyn-
 attribute), 178 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`dendritic_delay_fraction` (spyn- device_control_min_value (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` (spyn-
 attribute), 191 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`dependent_vertices()` (spyn- device_control_min_value (spyn-
`naker.pyNN.external_devices_models.external_device_lif_controller` (spyn-
 method), 40 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`dependent_vertices()` (spyn- device_control_partition_id (spyn-
`naker.pyNN.external_devices_models.munich_spinnaker_link_controller` (spyn-
 method), 44 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`dependent_vertices()` (spyn- device_control_partition_id (spyn-
`naker.pyNN.external_devices_models.MunichMotorDevice` (spyn-
 method), 51 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`describe()` (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex (spyn-
 method), 225 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`describe()` (spynnaker.pyNN.models.neuron.AbstractPopulationVertex (spyn-
 method), 236 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`describe()` (spynnaker.pyNN.models.spike_source.spike_source_analyser.pyNN.SpikeSourceAnalyserVertex (spyn-
 method), 245 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`describe()` (spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex (spyn-
 method), 250 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`describe()` (spynnaker.pyNN.models.utility_models.spike_injector.pyNN.SpikeInjectorVertex (spyn-
 method), 262 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`destroy()` (spynnaker.pyNN.utilities.fake_HBP_Portal_machine.pyNN.HBPPortalMachineDevice (spyn-
 method), 284 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`device_control_key` (spyn- device_control_send_type (spyn-
`naker.pyNN.external_devices_models.abstract_multicast_controller` (spyn-
 attribute), 38 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`device_control_key` (spyn- device_control_send_type (spyn-
`naker.pyNN.external_devices_models.AbstractMulticastController` (spyn-
 attribute), 48 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`device_control_key` (spyn- device_control_send_type (spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_etherne` (spyn-
 attribute), 11 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`device_control_key` (spyn- device_control_send_type (spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_etherne` (spyn-
 attribute), 19 `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`

[DOWNSAMPLE_64_X_64](#) (spyn- [elimination](#) (spyn-
[naker.pyNN.external_devices_models.push_bot.push_bot_payload_payload](#) [naker.pyNN.external_devices_models.push_bot.push_bot_payload_payload](#)
[attribute](#)), 29 [attribute](#)), 181
[DOWNSAMPLE_64_X_64](#) (spyn- [elimination](#) (spyn-
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey](#) [naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics](#)
[attribute](#)), 274 [attribute](#)), 183
[DOWNSAMPLE_64_X_64](#) (spyn- [elimination](#) (spyn-
[naker.pyNN.protocols.RetinaKey](#) [naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics](#)
[attribute](#)), 279 [attribute](#)), 185
[du_th](#) (spynaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic.ThresholdTypeMaassStochastic
[attribute](#)), 219 [naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics](#)
[du_th](#) (spynaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic
[attribute](#)), 222 [elimination](#) (spyn-
[duration](#) (spynaker.pyNN.models.spike_source.spike_source_poisson_spike_source_poisson_vertex.SpikeSourcePoissonVertex [naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics](#)
[attribute](#)), 250 [attribute](#)), 196
[durations](#) (spynaker.pyNN.models.spike_source.spike_source_poisson_spike_source_poisson_vertex.SpikeSourcePoissonVertex (spyn-
[attribute](#)), 250 [naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics](#)
[attribute](#)), 199
E [enable_disable_motor_key](#) (spyn-
[e_rev_E](#) (spynaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance [naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol](#)
[attribute](#)), 131 [attribute](#)), 271
[e_rev_E](#) (spynaker.pyNN.models.neuron.input_types.InputTypeConductance [enable_disable_motor_key](#) (spyn-
[attribute](#)), 134 [naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
[e_rev_I](#) (spynaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance [attribute](#)), 276
[attribute](#)), 131 [enable_motor\(\)](#) (spyn-
[e_rev_I](#) (spynaker.pyNN.models.neuron.input_types.InputTypeConductance [naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol](#)
[attribute](#)), 134 [static method](#)), 270
[edge_partition_identifiers_for_dependent_vertices](#) [enable_motor\(\)](#) (spyn-
(a spynaker.pyNN.external_devices_models.external_device_munich_io_ethernet_protocol.MunichIoEthernetProtocol
[method](#)), 40 [static method](#)), 275
[edge_partition_identifiers_for_dependent_vertices](#) [enable_motor\(\)](#) (spyn-
(a spynaker.pyNN.external_devices_models.munich_spinnaker_link_protocol.MunichIoEthernetProtocol
[method](#)), 44 [static method](#)), 270
[edge_partition_identifiers_for_dependent_vertices](#) [enable_motor\(\)](#) (spyn-
(a spynaker.pyNN.external_devices_models.MunichMotorDevice [naker.pyNN.protocols.MunichIoEthernetProtocol](#)
[method](#)), 51 [static method](#)), 275
[EIEIOSpikeRecorder](#) (class in spyn- [EthernetCommandConnection](#) (class in spyn-
[naker.pyNN.models.common](#)), 68 [naker.pyNN.connections](#)), 7
[EIEIOSpikeRecorder](#) (class in spyn- [EthernetCommandConnection](#) (class in spyn-
[naker.pyNN.models.common.eieio_spike_recorder](#)), [naker.pyNN.connections.ethernet_command_connection](#)),
63 3
[EIFConductanceAlphaPopulation](#) (class in [EthernetControlConnection](#) (class in spyn-
spynaker.pyNN.models.neuron.builds), 114 [naker.pyNN.connections](#)), 7
[EIFConductanceAlphaPopulation](#) [EthernetControlConnection](#) (class in spyn-
(class in spyn- [naker.pyNN.connections.ethernet_control_connection](#)),
[naker.pyNN.models.neuron.builds.eif_cond_alpha_isfa_isfa](#)), 4
111 [EVENTS_IN_PAYLOAD](#) (spyn-
[elimination](#) (spyn- [naker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey](#)
[naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural](#)
[attribute](#)), 175 [EVENTS_IN_PAYLOAD](#) (spyn-
[elimination](#) (spyn- [naker.pyNN.protocols.RetinaPayload](#) [attribute](#)),
[naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamicsStructural](#)
[attribute](#)), 193 [exc2_old](#) (spynaker.pyNN.models.neuron.synapse_types.synapse_type_s
[attribute](#)), 209

exc2_old (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha, attribute), 217

exc_response (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha, attribute), 203

exc_response (spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha, attribute), 216

ExternalCochleaDevice (class in spynnaker.pyNN.external_devices_models), 49

ExternalCochleaDevice (class in spynnaker.pyNN.external_devices_models.external_spinnaker_link_cochlea_device), 41

ExternalDeviceLifControl (class in spynnaker.pyNN.external_devices_models), 49

ExternalDeviceLifControl (class in spynnaker.pyNN.external_devices_models.external_device_lif_control), 39

ExternalDeviceLifControlVertex (class in spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex), 40

ExternalFPGA retina Device (class in spynnaker.pyNN.external_devices_models), 49

ExternalFPGA retina Device (class in spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device), 42

extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_factory.AbstractMasterPopTableFactory, method), 138

extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch, method), 139

extract_synaptic_matrix_data_location() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch, method), 141

ExtractedData (class in spynnaker.pyNN.utilities.extracted_data), 284

F

f Rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic, attribute), 175

f Rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic, attribute), 193

f Rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic, attribute), 181

f Rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic, attribute), 183

f Rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic, attribute), 185

f Rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic, attribute), 195

f Rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic, attribute), 196

finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_factory.AbstractMasterPopTableFactory, method), 138

finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch, method), 139

finish_master_pop_table() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch, method), 141

first_id (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon, attribute), 266

FIXED_KEY (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey, attribute), 274

tribute), 279
 FIXED_NUMBER_POST_CONNECTOR (spyn-
 naker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.ConnectorIDs
 attribute), 73
 FIXED_NUMBER_PRE_CONNECTOR (spyn-
 naker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.ConnectorIDs
 attribute), 73
 FIXED_PROBABILITY_CONNECTOR (spyn-
 naker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.ConnectorIDs
 attribute), 73
 FIXED_TOTAL_NUMBER_CONNECTOR (spyn-
 naker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine.ConnectorIDs
 attribute), 73
 FixedNumberPostConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors), 91
 FixedNumberPostConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_number_post_connector), 77
 FixedNumberPreConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors), 93
 FixedNumberPreConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector), 78
 FixedProbabilityConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors), 94
 FixedProbabilityConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_probability_connector), 79
 float_to_fixed() (in module spyn-
 naker.pyNN.models.neuron.plasticity.stdp.common_plasticity_helpers), 150
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStru-
 ture), 175
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamicsStructuralCommon), 193
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralC-
 ommon), 181
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStat-
 ic), 183
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStat-
 ic), 185
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon), 195
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic), 197
 formation (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic), 199
 FREE (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES), 271
 FREE (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES), 276

`gen_delay_params()` (spyn- `gen_matrix_params` (spyn-
`naker.pyNN.models.neural_projections.connectors.kernel_connection` (spyn-
`method`), 83 `attribute`), 178
`gen_delay_params()` (spyn- `gen_matrix_params` (spyn-
`naker.pyNN.models.neural_projections.connectors.KernelConnection` (spyn-
`method`), 100 `attribute`), 191
`gen_delay_params_size_in_bytes()` (spyn- `gen_matrix_params_size_in_bytes` (spyn-
`naker.pyNN.models.neural_projections.connectors.abstract_generator` (spyn-
`method`), 73 `attribute`), 172
`gen_delay_params_size_in_bytes()` (spyn- `gen_matrix_params_size_in_bytes` (spyn-
`naker.pyNN.models.neural_projections.connectors.AbstractGenerator` (spyn-
`method`), 88 `attribute`), 187
`gen_delay_params_size_in_bytes()` (spyn- `gen_matrix_params_size_in_bytes` (spyn-
`naker.pyNN.models.neural_projections.connectors.kernel_connection` (spyn-
`method`), 83 `attribute`), 178
`gen_delay_params_size_in_bytes()` (spyn- `gen_matrix_params_size_in_bytes` (spyn-
`naker.pyNN.models.neural_projections.connectors.KernelConnection` (spyn-
`method`), 100 `attribute`), 191
`gen_delays_id()` (spyn- `gen_on_machine()` (spyn-
`naker.pyNN.models.neural_projections.connectors.abstract_generator` (spyn-
`method`), 73 `method`), 225
`gen_delays_id()` (spyn- `gen_on_machine()` (spyn-
`naker.pyNN.models.neural_projections.connectors.AbstractGenerator` (spyn-
`method`), 88 `method`), 236
`gen_delays_id()` (spyn- `gen_on_machine()` (spyn-
`naker.pyNN.models.neural_projections.connectors.kernel_connection` (spyn-
`method`), 83 `method`), 234
`gen_delays_id()` (spyn- `gen_on_machine()` (spyn-
`naker.pyNN.models.neural_projections.connectors.KernelConnection` (spyn-
`method`), 100 `method`), 241
`gen_matrix_id` (spyn- `gen_on_machine()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.abstract_generator` (spyn-
`attribute`), 172 `method`), 256
`gen_matrix_id` (spyn- `gen_on_machine()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.AbstractGenerator` (spyn-
`attribute`), 187 `method`), 260
`gen_matrix_id` (spyn- `gen_weight_params_size_in_bytes()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics` (spyn-
`attribute`), 176 `method`), 73
`gen_matrix_id` (spyn- `gen_weight_params_size_in_bytes()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics` (spyn-
`attribute`), 178 `method`), 88
`gen_matrix_id` (spyn- `gen_weight_params_size_in_bytes()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` (spyn-
`attribute`), 189 `method`), 83
`gen_matrix_id` (spyn- `gen_weight_params_size_in_bytes()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` (spyn-
`attribute`), 191 `method`), 100
`gen_matrix_params` (spyn- `gen_weights_id()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.abstract_generator` (spyn-
`attribute`), 172 `method`), 73
`gen_matrix_params` (spyn- `gen_weights_id()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.AbstractGenerator` (spyn-
`attribute`), 187 `method`), 88

generic_motor_disable()	(spyn-	method), 260	
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spyn-	naker.pyNN.external_devices_models.munich_spinnaker_link_mo	
method), 271			
generic_motor_disable()	(spyn-	method), 44	
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	get_block_n_bytes()	(spyn-	
method), 276		naker.pyNN.external_devices_models.MunichMotorDevice	
generic_motor_enable()	(spyn-	method), 52	
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spyn-	naker.pyNN.models.neuron.abstract_population_vertex.AbstractP	
method), 271		naker.pyNN.models.neuron.abstract_population_vertex.AbstractP	
generic_motor_enable()	(spyn-	method), 225	
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	get_block_n_bytes()	(spyn-	
method), 276		naker.pyNN.models.neuron.AbstractPopulationVertex	
generic_motor_total_period()	(spyn-	method), 236	
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spyn-	naker.pyNN.models.spike_source.spike_source_poisson_vertex.Sp	
method), 271		naker.pyNN.models.spike_source.spike_source_poisson_vertex.Sp	
generic_motor_total_period()	(spyn-	method), 250	
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	get_block_n_bytes()	(spyn-	
method), 276		naker.pyNN.models.utility_models.delays.delay_extension_vertex.	
generic_motor_total_period_key	(spyn-	method), 256	
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spyn-	naker.pyNN.models.utility_models.delays.DelayExtensionVertex	
attribute), 271		naker.pyNN.models.utility_models.delays.DelayExtensionVertex	
generic_motor_total_period_key	(spyn-	method), 260	
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	get_block_n_bytes()	(spyn-	
attribute), 276		naker.pyNN.models.neuron.synapse_io.abstract_synapse_io.Abstr	
get()	(spyn-	naker.pyNN.models.pynn_population_common.PyNNPopulationCommon	
method), 266		get_block_n_bytes()	(spyn-
get()	(spyn-	naker.pyNN.models.neuron.synapse_io.AbstractSynapseIO	
method), 284		method), 201	
get_allowed_row_length()	(spyn-	get_block_n_bytes()	(spyn-
naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators	naker.pyNN.models.neuron.synapse_io.SynapseIORowBased		
method), 139		method), 201	
get_allowed_row_length()	(spyn-	get_block_n_bytes()	(spyn-
naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAbstractBinarySource	naker.pyNN.models.neuron.synapse_io.SynapseIORowBased		
method), 141		method), 202	
get_binary_file_name()	(spyn-	get_buffer_sizes()	(in module spyn-
naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	naker.pyNN.external_devices_models.MunichMotorDevice		
method), 44		get_buffer_sizes()	(in module spyn-
get_binary_file_name()	(spyn-	naker.pyNN.models.common.recording_utils),	
naker.pyNN.external_devices_models.MunichMotorDevice	65		
method), 52		get_buffered_sdram()	(spyn-
get_binary_file_name()	(spyn-	naker.pyNN.models.common.neuron_recorder.NeuronRecorder	
naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex	abstract_population_vertex.AbstractPopulationVertex		
method), 225		get_buffered_sdram()	(spyn-
get_binary_file_name()	(spyn-	naker.pyNN.models.common.NeuronRecorder	
naker.pyNN.models.neuron.AbstractPopulationVertex	method), 68		
method), 236		get_buffered_sdram_per_record()	(spyn-
get_binary_file_name()	(spyn-	naker.pyNN.models.common.neuron_recorder.NeuronRecorder	
naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex	method), 250	get_buffered_sdram_per_record()	(spyn-
method), 250		get_buffered_sdram_per_timestep()	(spyn-
get_binary_file_name()	(spyn-	naker.pyNN.models.common.NeuronRecorder	
naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex	method), 256	get_buffered_sdram_per_timestep()	(spyn-
method), 256		naker.pyNN.models.common.neuron_recorder.NeuronRecorder	
get_binary_file_name()	(spyn-	naker.pyNN.models.utility_models.delays.DelayExtensionVertex	
naker.pyNN.models.utility_models.delays.DelayExtensionVertex	method), 64		

[get_buffered_sdram_per_timestep\(\)](#) (*spynaker.pyNN.models.common.NeuronRecorder* method), 69
[get_by_selector\(\)](#) (*spynaker.pyNN.models.pynn_population_common.PyNNPopulationCommon* (in module *spynaker.pyNN.models.common*), 70
[get_colour\(\)](#) (in module *spynaker.plot_utils*), 294
[get_connection_holders\(\)](#) (*spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex* method), 225
[get_connection_holders\(\)](#) (*spynaker.pyNN.models.neuron.AbstractPopulationVertex* method), 237
[get_connection_holders\(\)](#) (*spynaker.pyNN.models.neuron.synaptic_manager.SynapticManager* method), 234
[get_connection_holders\(\)](#) (*spynaker.pyNN.models.neuron.SynapticManager* method), 241
[get_connections_from_machine\(\)](#) (*spynaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses* method), 55
[get_connections_from_machine\(\)](#) (*spynaker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses* method), 58
[get_connections_from_machine\(\)](#) (*spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex* method), 225
[get_connections_from_machine\(\)](#) (*spynaker.pyNN.models.neuron.AbstractPopulationVertex* method), 237
[get_connections_from_machine\(\)](#) (*spynaker.pyNN.models.neuron.synaptic_manager.SynapticManager* method), 234
[get_connections_from_machine\(\)](#) (*spynaker.pyNN.models.neuron.SynapticManager* method), 242
[get_cpu_usage_for_atoms\(\)](#) (*spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex* method), 225
[get_cpu_usage_for_atoms\(\)](#) (*spynaker.pyNN.models.neuron.AbstractPopulationVertex* method), 237
[get_cpu_usage_for_atoms\(\)](#) (*spynaker.pyNN.models.spike_source.spike_source_poisson_data.SpikeSourcePoissonVertex* (static method), 250
[get_cpu_usage_for_atoms\(\)](#) (*spynaker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex* method), 257
[get_cpu_usage_for_atoms\(\)](#) (*spynaker.pyNN.models.utility_models.delays.DelayExtensionVertex* method), 260
[get_current_time\(\)](#) (*spynaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface* method), 23
[get_current_time\(\)](#) (*spynaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState* method), 285
[get_data\(\)](#) (in module *spynaker.pyNN.models.common.recording_utils*),
[get_data\(\)](#) (*spynaker.pyNN.models.common.abstract_neuron_recorder.AbstractNeuronRecorder* method), 61
[get_data\(\)](#) (*spynaker.pyNN.models.common.AbstractNeuronRecorder* method), 66
[get_data\(\)](#) (*spynaker.pyNN.models.common.neuron_recorder.NeuronRecorder* method), 64
[get_data\(\)](#) (*spynaker.pyNN.models.common.NeuronRecorder* method), 69
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex* method), 225
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.AbstractPopulationVertex* method), 235
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.implementations.abstract_neuron_model.AbstractNeuronModel* method), 116
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.implementations.abstract_neuron_model.AbstractNeuronModel* method), 118
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.implementations.AbstractNeuronModel* method), 123
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.implementations.neuron_implementations.NeuronImplementations* method), 126
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.implementations.StructuredNeuronModel* method), 125
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.implementations.struct.StructuredNeuronModel* method), 122
[get_data\(\)](#) (*spynaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel* method), 146
[get_database_connection\(\)](#) (*spynaker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor* method), 37
[get_data\(\)](#) (*spynaker.pyNN.external_devices_models.AbstractEthernetSensor* method), 47
[get_data\(\)](#) (*spynaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot.PushBotEthernetPushBot* method), 15
[get_data\(\)](#) (*spynaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot.PushBotEthernetPushBot* method), 23
[get_data_type\(\)](#) (*spynaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface* method), 23

<code>naker.pyNN.models.neural_properties.neural_parameter.NeuralParameter</code> (method), 108	<code>naker.pyNN.models.neural_projections.connectors.from_list_connector</code> (method), 80
<code>get_dataspec_datatype()</code> <code>naker.pyNN.models.neural_properties.NeuralParameter</code> (method), 108	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.FromListConnector</code> (method), 95
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.abstract_connector</code> (method), 71	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.index_based_projection</code> (method), 82
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.AbstractConnector</code> (method), 87	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.IndexBasedProjection</code> (method), 96
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.all_to_all_connector</code> (method), 74	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.kernel_connector</code> (method), 83
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.AllToAllConnector</code> (method), 89	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.KernelConnector</code> (method), 100
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.array_connector</code> (method), 74	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.multipse_connector</code> (method), 85
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.ArrayConnector</code> (method), 90	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.MultipseConnector</code> (method), 97
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.csa_connector</code> (method), 75	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.one_to_one_connector</code> (method), 86
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.CSAConnector</code> (method), 90	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.OneToOneConnector</code> (method), 98
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector</code> (method), 76	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.SmallWorldConnector</code> (method), 99
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilityConnector</code> (method), 91	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.SmallWorldConnector</code> (method), 99
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.fixed_number_of_synapses_connector</code> (method), 77	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.fixed_number_of_synapses_connector</code> (method), 174
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.fixed_number_of_synapses_connector</code> (method), 78	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.fixed_number_of_synapses_connector</code> (method), 186
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.fixed_probability_connector</code> (method), 80	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.fixed_probability_connector</code> (method), 71
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector</code> (method), 92	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector</code> (method), 87
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector</code> (method), 93	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector</code> (method), 80
<code>get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector</code> (method), 94	<code>(spyn- get_delay_maximum()</code> <code>naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector</code> (method), 95
<code>get_delay_maximum()</code>	<code>(spyn- get_delay_maximum()</code>

<code>naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics</code> (method), 174	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.implementations.AbstractStandardNeuron</code> (method), 186
<code>get_delay_variance()</code> (spyn- <code>naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics</code> (method), 186	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.implementations.neuron_impl_standard</code> (method), 120
<code>get_dict_from_init()</code> (in module <code>spyn- naker.pyNN.models.defaults</code>), 265	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.implementations.NeuronImplStandard</code> (method), 126
<code>get_distribution_to_stats()</code> (spyn- <code>naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface</code> (method), 293	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.neuron_models.abstract_neuron_model</code> (method), 142
<code>get_distribution_to_stats()</code> (spyn- <code>naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState</code> (method), 285	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.neuron_models.abstract_neuron_model</code> (method), 146
<code>get_dtcn_usage_for_atoms()</code> (spyn- <code>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</code> (method), 237	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.synaptic_manager.SynapticManager</code> (method), 234
<code>get_dtcn_usage_for_atoms()</code> (spyn- <code>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</code> (static method), 250	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.synaptic_manager.SynapticManager</code> (method), 242
<code>get_dtcn_usage_for_atoms()</code> (spyn- <code>naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex</code> (method), 257	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code> (method), 138
<code>get_dtcn_usage_for_atoms()</code> (spyn- <code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code> (method), 260	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators.abstract</code> (method), 139
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.common.eieio_spike_recorder.EIEIOSpikeRecorder</code> (method), 63	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators</code> (method), 141
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.common.EIEIOSpikeRecorder</code> (method), 68	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators.master_pop_table_generators</code> (method), 140
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder</code> (method), 63	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.master_pop_table_generators.M</code> (method), 141
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.common.MultiSpikeRecorder</code> (method), 70	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.plasticity.stdp.common.plasticity_help</code> (method), 150
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.common.neuron_recorder.NeuronRecorder</code> (method), 64	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.plasticity.stdp.common.plasticity_help</code> (method), 150
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.common.NeuronRecorder</code> (method), 69	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.external_devices_models.abstract_ethernet_controller</code> (method), 37
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.implementations.abstract_neuron_model</code> (method), 116	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.external_devices_models.AbstractEthernetController</code> (method), 47
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.implementations.abstract_standard_neuron_model</code> (method), 118	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.external_devices_models.AbstractStandardNeuronModel</code> (method), 41
<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neuron.implementations.AbstractNeuronModel</code> (method), 118	<code>get_dtcn_usage_in_bytes()</code> (spyn- <code>naker.pyNN.models.neural_projections.connectors.from_list_con</code> (method), 118

329

`method)`, 237
`get_maximum_delay_supported_in_ms()`
`(spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io_row_based.SynapseIORowBased`
`method)`, 200
`get_maximum_delay_supported_in_ms()`
`(spynnaker.pyNN.models.neuron.synapse_io.AbstractSynapseIORowBased``method)`, 74
`method)`, 201
`get_maximum_delay_supported_in_ms()`
`(spynnaker.pyNN.models.neuron.synapse_io.synapse_io_row_based.SynapseIORowBased`
`method)`, 201
`get_maximum_delay_supported_in_ms()`
`(spynnaker.pyNN.models.neuron.synapse_io.SynapseIORowBased``method)`, 75
`method)`, 202
`get_maximum_delay_supported_in_ms()`
`(spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager``method)`, 234
`method)`, 242
`get_maximum_probable_value()` (in module
`spynnaker.pyNN.utilities.utility_calls)`, 286
`get_mean()` (in module `spyn-`
`naker.pyNN.utilities.utility_calls)`, 286
`get_message_translator()` (`spyn-`
`naker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController`
`method)`, 37
`get_message_translator()` (`spyn-`
`naker.pyNN.external_devices_models.AbstractEthernetController`
`method)`, 47
`get_message_translator()` (`spyn-`
`naker.pyNN.external_devices_models.external_device_life_controller.ExternalDeviceLifeController`
`method)`, 41
`get_minimum_probable_value()` (in module
`spynnaker.pyNN.utilities.utility_calls)`, 286
`get_munich_d()` (in module `spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol)`, 274
`method)`, 80
`get_munich_f()` (in module `spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol)`, 275
`method)`, 92
`get_munich_i()` (in module `spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol)`, 275
`method)`, 93
`get_n_bits()` (in module `spyn-`
`naker.pyNN.utilities.utility_calls)`, 286
`get_n_connections()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.from_list.FromList`
`method)`, 81
`get_n_connections()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.FromList`
`method)`, 95
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.abstract_synapse_io_row_based.SynapseIORowBased`
`method)`, 71
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.AbstractSynapseIO`
`method)`, 77
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.all_to_all.AllToAll`
`method)`, 74
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.AllToAll`
`method)`, 79
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.array_based.ArrayBased`
`method)`, 75
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.ArrayCon`
`method)`, 90
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.csa_conn`
`method)`, 75
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.CSAConn`
`method)`, 90
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.distance`
`method)`, 76
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.Distance`
`method)`, 91
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.fixed_num`
`method)`, 77
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.fixed_num`
`method)`, 79
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.fixed_pro`
`method)`, 80
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.FixedNum`
`method)`, 92
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.FixedNum`
`method)`, 93
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.FixedPro`
`method)`, 94
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.FromList`
`method)`, 81
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.FromList`
`method)`, 95
`get_n_connections_from_pre_vertex_maximum()`
`(spynnaker.pyNN.models.neural_projections.connectors.index_ba`
`method)`, 82

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.IndexBasedPrePyNNinCortex.neural_projections.connectors.Distance
 method), 96

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.kernel_based_pre_pyNNinCortex.neural_projections.connectors.fixed_nu
 method), 84

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.KernelBasedPrePyNNinCortex.neural_projections.connectors.fixed_nu
 method), 100

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.multipse_pre_pyNNinCortex.neural_projections.connectors.fixed_pro
 method), 85

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.MultipsePrePyNNinCortex.neural_projections.connectors.FixedNum
 method), 97

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.one_to_one_pre_pyNNinCortex.neural_projections.connectors.FixedNum
 method), 86

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.OneToOnePrePyNNinCortex.neural_projections.connectors.FixedPro
 method), 98

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.small_world_pre_pyNNinCortex.neural_projections.connectors.from_list
 method), 86

`get_n_connections_from_pre_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.SmallWorldPrePyNNinCortex.neural_projections.connectors.FromList
 method), 99

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.abstract_pre_pyNNinCortex.neural_projections.connectors.index_ba
 method), 71

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.AbstPrePyNNinCortex.neural_projections.connectors.IndexBas
 method), 87

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.all_pre_pyNNinCortex.neural_projections.connectors.kernel_co
 method), 74

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.AllToAllPrePyNNinCortex.neural_projections.connectors.KernelCo
 method), 89

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.array_pre_pyNNinCortex.neural_projections.connectors.multipse
 method), 75

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.ArrayPrePyNNinCortex.neural_projections.connectors.Multipse
 method), 90

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.csa_pre_pyNNinCortex.neural_projections.connectors.one_to_o
 method), 75

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.CSACommPrePyNNinCortex.neural_projections.connectors.OneToOn
 method), 90

`get_n_connections_to_post_vertex_maximum()`
 (spynnaker.pyNN.models.neural_projections.connectors.distributable_pre_pyNNinCortex.neural_projections.connectors.Distrib
 method), 76

333

<code>get_n_cpu_cycles()</code>	(<code>spyn-</code>	<code>get_n_items()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMIPlasticStaticSynapseDynamicsAbstractSynapseDynamics</code>		<code>naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics</code>	
<code>method), 217</code>		<code>method), 186</code>	
<code>get_n_cpu_cycles()</code>	(<code>spyn-</code>	<code>get_n_keys_for_partition()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.synaptic_manager.SynapticManager</code>		<code>naker.pyNN.models.utility_models.delays.delay_extension_vertex</code>	
<code>method), 234</code>		<code>method), 257</code>	
<code>get_n_cpu_cycles()</code>	(<code>spyn-</code>	<code>get_n_keys_for_partition()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.SynapticManager</code>		<code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code>	
<code>method), 242</code>		<code>method), 260</code>	
<code>get_n_cpu_cycles()</code>	(<code>spyn-</code>	<code>get_n_neurons()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.threshold_types.threshold_type_neuroplasticity_threshold_type_neuroplasticity</code>		<code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code>	
<code>method), 219</code>		<code>method), 38</code>	
<code>get_n_cpu_cycles()</code>	(<code>spyn-</code>	<code>get_n_neurons()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.threshold_types.threshold_type_neuroplasticity_threshold_type_neuroplasticity</code>		<code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code>	
<code>method), 220</code>		<code>method), 48</code>	
<code>get_n_cpu_cycles()</code>	(<code>spyn-</code>	<code>get_n_neurons()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.threshold_types.ThresholdTypeNeuroplasticityThresholdTypeNeuroplasticity</code>		<code>naker.pyNN.external_devices_models.external_spinnaker_link_fp</code>	
<code>method), 222</code>		<code>static method), 42</code>	
<code>get_n_cpu_cycles()</code>	(<code>spyn-</code>	<code>get_n_neurons()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.threshold_types.ThresholdTypeNeuroplasticityThresholdTypeNeuroplasticity</code>		<code>naker.pyNN.external_devices_models.ExternalFPGA retina device</code>	
<code>method), 221</code>		<code>static method), 50</code>	
<code>get_n_fixed_plastic_words_per_row()</code>	<code>get_n_neurons()</code>		(<code>spyn-</code>
(<code>spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics</code>)			
<code>method), 173</code>			<code>method), 15</code>
<code>get_n_fixed_plastic_words_per_row()</code>	<code>get_n_neurons()</code>		(<code>spyn-</code>
(<code>spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseDynamics</code>)			
<code>method), 188</code>			<code>method), 23</code>
<code>get_n_fixed_plastic_words_per_row()</code>	<code>get_n_plastic_plastic_words_per_row()</code>		
(<code>spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics</code>)			
<code>method), 178</code>			<code>method), 173</code>
<code>get_n_fixed_plastic_words_per_row()</code>	<code>get_n_plastic_plastic_words_per_row()</code>		
(<code>spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</code>)			
<code>method), 191</code>			<code>method), 188</code>
<code>get_n_half_words_per_connection()</code>	(<code>spyn-</code>	<code>get_n_plastic_plastic_words_per_row()</code>	
<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure</code>		<code>abstract_synapse_dynamics</code>	
<code>method), 150</code>		<code>method), 178</code>	
<code>get_n_half_words_per_connection()</code>	(<code>spyn-</code>	<code>get_n_plastic_plastic_words_per_row()</code>	
<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure</code>		<code>abstract_synapse_dynamics</code>	
<code>method), 151</code>		<code>method), 191</code>	
<code>get_n_half_words_per_connection()</code>	(<code>spyn-</code>	<code>get_n_static_words_per_row()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure</code>		<code>synapse_dynamics</code>	
<code>method), 151</code>		<code>method), 173</code>	
<code>get_n_half_words_per_connection()</code>	(<code>spyn-</code>	<code>get_n_static_words_per_row()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure</code>		<code>synapse_dynamics</code>	
<code>method), 151</code>		<code>method), 188</code>	
<code>get_n_half_words_per_connection()</code>	(<code>spyn-</code>	<code>get_n_static_words_per_row()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure</code>		<code>synapse_dynamics</code>	
<code>method), 151</code>		<code>method), 176</code>	
<code>get_n_half_words_per_connection()</code>	(<code>spyn-</code>	<code>get_n_static_words_per_row()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure</code>		<code>synapse_dynamics</code>	
<code>method), 151</code>		<code>method), 189</code>	
<code>get_n_items()</code>	(<code>spyn-</code>	<code>get_n_synapse_types()</code>	(<code>spyn-</code>
<code>naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics</code>		<code>naker.pyNN.models.neuron.synapse_dynamics.abstract_neuron_imp</code>	
<code>method), 174</code>		<code>method), 116</code>	

<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.implementations.AbstractNeuronImpl</i> <i>method), 129</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.implementations.AbstractNeuronImpl</i> <i>method), 188</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamics</i> <i>method), 188</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.implementations.neuron_impl_standard</i> <i>method), 121</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.implementations.neuron_impl_standard</i> <i>method), 176</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_standard</i> <i>method), 176</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.implementations.NeuronImplStandard</i> <i>method), 127</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.implementations.NeuronImplStandard</i> <i>method), 179</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_standard</i> <i>method), 179</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.abstract_synapse_type</i> <i>method), 202</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.abstract_synapse_type</i> <i>method), 189</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsAbstract</i> <i>method), 189</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.AbstractSynapseType</i> <i>method), 211</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.AbstractSynapseType</i> <i>method), 191</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsAbstract</i> <i>method), 191</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_alpha</i> <i>method), 203</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_alpha</i> <i>method), 173</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_connections_alpha</i> <i>method), 173</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_delta</i> <i>method), 205</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_delta</i> <i>method), 188</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticConnectionsDelta</i> <i>method), 188</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential</i> <i>method), 206</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential</i> <i>method), 179</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_dual_exponential</i> <i>method), 179</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_exponential</i> <i>method), 208</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_exponential</i> <i>method), 185</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_exponential</i> <i>method), 185</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_semi_exponential</i> <i>method), 209</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.synapse_type_semi_exponential</i> <i>method), 192</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSemiExponential</i> <i>method), 192</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha</i> <i>method), 216</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha</i> <i>method), 199</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsAlpha</i> <i>method), 199</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta</i> <i>method), 214</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta</i> <i>method), 173</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.abstract_static_connections_delta</i> <i>method), 173</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential</i> <i>method), 211</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential</i> <i>method), 188</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamicsDualExponential</i> <i>method), 188</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential</i> <i>method), 213</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential</i> <i>method), 176</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_exponential</i> <i>method), 176</i>
<code>get_n_synapse_types()</code> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMI</i> <i>method), 217</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMI</i> <i>method), 183</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_semi_exponential</i> <i>method), 183</i>
<code>get_n_synapses_in_rows()</code> <i>naker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_connections_delta</i> <i>method), 173</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_connections_delta</i> <i>method), 189</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsAbstractPlasticConnectionsDelta</i> <i>method), 189</i>
<code>get_n_synapses_in_rows()</code> <i>naker.pyNN.models.neuron.synapse_dynamics.abstract_static_connections_delta</i> <i>method), 173</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.abstract_static_connections_delta</i> <i>method), 197</i>	<i>(spyn-</i> <i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsAbstractStaticConnectionsDelta</i> <i>method), 197</i>
<code>get_n_synapses_in_rows()</code> <i>naker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticConnectionsDelta</i> <i>method), 188</i>	<i>(spyn-</i> <i>naker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable</i> <i>method), 61</i>	<i>(spyn-</i> <i>naker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable</i> <i>method), 61</i>

`get_neuron_sampling_interval()` (spyn- `get_outgoing_partition_constraints()`
`naker.pyNN.models.common.AbstractNeuronRecordable` (spynnaker.pyNN.models.utility_models.delays.DelayExtensionVe
method), 67 method), 260
`get_neuron_sampling_interval()` (spyn- `get_outgoing_partition_constraints()`
`naker.pyNN.models.common.neuron_recorder.NeuronRecorder` (spynnaker.pyNN.models.utility_models.spike_injector.spike_injec
method), 65 method), 262
`get_neuron_sampling_interval()` (spyn- `get_outgoing_partition_ids()` (spyn-
`naker.pyNN.models.common.NeuronRecorder` `naker.pyNN.external_devices_models.abstract_ethernet_controller`
method), 69 method), 37
`get_neuron_sampling_interval()` (spyn- `get_outgoing_partition_ids()` (spyn-
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex` `naker.pyNN.external_devices_models.AbstractEthernetController`
method), 226 method), 47
`get_neuron_sampling_interval()` (spyn- `get_outgoing_partition_ids()` (spyn-
`naker.pyNN.models.neuron.AbstractPopulationVertex` `naker.pyNN.external_devices_models.external_device_lif_control`
method), 238 method), 41
`get_next_allowed_address()` (spyn- `get_parameter_names()` (spyn-
`naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators` `naker.pyNN.models.utility_models.utility_models`
method), 140 class method), 265
`get_next_allowed_address()` (spyn- `get_parameter_names()` (spyn-
`naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators` `naker.pyNN.models.utility_models.utility_models`
method), 141 method), 152
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.external_devices_models.external_device_lif_control)` `naker.pyNN.models.utility_models.utility_models`
method), 41 method), 156
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.external_devices_models.external_spinnaker)` `naker.pyNN.models.utility_models.utility_models`
method), 42 method), 152
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.external_devices_models.ExternalFPGA)` `naker.pyNN.models.utility_models.utility_models`
method), 50 method), 153
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.external_devices_models.munich_spinnaker)` `naker.pyNN.models.utility_models.utility_models`
method), 44 method), 154
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.external_devices_models.munich_spinnaker)` `naker.pyNN.models.utility_models.utility_models`
method), 45 method), 154
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.external_devices_models.MunichMotorDevice)` `naker.pyNN.models.utility_models.utility_models`
method), 52 method), 155
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.external_devices_models.MunichRetinaDevice)` `naker.pyNN.models.utility_models.utility_models`
method), 53 method), 157
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex)` `naker.pyNN.models.utility_models.utility_models`
method), 226 method), 157
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.models.neuron.AbstractPopulationVertex)` `naker.pyNN.models.utility_models.utility_models`
method), 238 method), 158
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.models.spike_source.spike_source_poisson)` `naker.pyNN.models.utility_models.utility_models`
method), 250 method), 156
`get_outgoing_partition_constraints()` `get_parameter_names()` (spyn-
`(spynnaker.pyNN.models.utility_models.delays.delay_extension)` `naker.pyNN.models.utility_models.utility_models`
method), 257 method), 159

[illegible]

[illegible]

`method`), 257
`get_resources_used_by_atoms()` (`spyn-` `get_sdram_usage_in_bytes()` (`spyn-`
`naker.pyNN.models.utility_models.delays.DelayExtensionVertex`
`method`), 261
`get_retina_i()` (`in` `module` `spyn-` `get_sdram_usage_in_bytes()` (`spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol`), `naker.pyNN.models.neuron.implementations.NeuronImplStandard`
275
`method`), 127
`GET_RETINA_KEY_VALUE()` (`in` `module` `spyn-` `get_sdram_usage_in_bytes()` (`spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol`), `naker.pyNN.models.neuron.neuron_models.abstract_neuron_model`
270
`method`), 143
`GET_RETINA_PAYLOAD_VALUE()` (`in` `module` `spyn-` `get_sdram_usage_in_bytes()` (`spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol`), `naker.pyNN.models.neuron.neuron_models.AbstractNeuronModel`
270
`method`), 146
`get_rng_next()` (`spyn-` `get_sdram_usage_in_bytes()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.multiparser`
`method`), 85
`method`), 234
`get_rng_next()` (`spyn-` `get_sdram_usage_in_bytes()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.Multiparser`
`method`), 97
`method`), 242
`get_sampling_overflow_sdram()` (`spyn-` `get_size_in_whole_words()` (`spyn-`
`naker.pyNN.models.common.neuron_recorder.NeuronRecorder`
`method`), 65
`method`), 125
`get_sampling_overflow_sdram()` (`spyn-` `get_size_in_whole_words()` (`spyn-`
`naker.pyNN.models.common.NeuronRecorder`
`method`), 70
`method`), 122
`get_sdram_usage_for_atoms()` (`spyn-` `get_spike_counts()` (`spyn-`
`naker.pyNN.models.utility_models.delays.delay_extension_vertex`
`method`), 257
`method`), 266
`get_sdram_usage_for_atoms()` (`spyn-` `get_spike_value_from_fpga_retina()`
`naker.pyNN.models.utility_models.delays.DelayExtensionVertex`
`method`), 261
`method`), 43
`get_sdram_usage_in_bytes()` (`spyn-` `get_spike_value_from_fpga_retina()`
`naker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder`
`method`), 63
`method`), 46
`get_sdram_usage_in_bytes()` (`spyn-` `get_spike_value_from_robot_retina()`
`naker.pyNN.models.common.MultiSpikeRecorder`
`method`), 70
`method`), 46
`get_sdram_usage_in_bytes()` (`spyn-` `get_spikes()` (`spyn-`
`naker.pyNN.models.common.neuron_recorder.NeuronRecorder`
`method`), 65
`method`), 62
`get_sdram_usage_in_bytes()` (`spyn-` `get_spikes()` (`spyn-`
`naker.pyNN.models.common.NeuronRecorder`
`method`), 70
`method`), 67
`get_sdram_usage_in_bytes()` (`spyn-` `get_spikes()` (`spyn-`
`naker.pyNN.models.neuron.implementations.abstract_neuron_impl`
`method`), 117
`method`), 63
`get_sdram_usage_in_bytes()` (`spyn-` `get_spikes()` (`spyn-`
`naker.pyNN.models.neuron.implementations.abstract_standard_neuron_impl`
`method`), 118
`method`), 68
`get_sdram_usage_in_bytes()` (`spyn-` `get_spikes()` (`spyn-`
`naker.pyNN.models.neuron.implementations.AbstractNeuronImpl`
`method`), 129
`method`), 63
`get_sdram_usage_in_bytes()` (`spyn-` `get_spikes()` (`spyn-`
`naker.pyNN.models.neuron.implementations.AbstractStandardNeuronComponent`
`method`), 129
`method`), 63

<code>get_spikes()</code>	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder method), 65	naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics method), 189
<code>get_spikes()</code>	(spyn- naker.pyNN.models.common.NeuronRecorder method), 70	get_structural_parameters_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics. method), 175
<code>get_spikes()</code>	(spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 227	get_structural_parameters_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics. method), 175
<code>get_spikes()</code>	(spyn- naker.pyNN.models.neuron.AbstractPopulationVertex method), 238	get_structural_parameters_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics. method), 183
<code>get_spikes()</code>	(spyn- naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex method), 245	get_structural_parameters_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 185
<code>get_spikes()</code>	(spyn- naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex method), 251	get_structural_parameters_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 185
<code>get_spikes()</code>	(spyn- naker.pyNN.models.utility_models.spike_injector.spike_injector_vertex.SpikeInjectorVertex method), 262	get_structural_parameters_sdram_usage_in_bytes() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 185
<code>get_spikes_sampling_interval()</code>	(spyn- naker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable method), 62	get_synapse_id_by_target() (spyn- naker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses. method), 58
<code>get_spikes_sampling_interval()</code>	(spyn- naker.pyNN.models.common.AbstractSpikeRecordable method), 67	get_synapse_id_by_target() (spyn- naker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses. method), 58
<code>get_spikes_sampling_interval()</code>	(spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 227	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 238
<code>get_spikes_sampling_interval()</code>	(spyn- naker.pyNN.models.neuron.AbstractPopulationVertex method), 238	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.AbstractPopulationVertex method), 238
<code>get_spikes_sampling_interval()</code>	(spyn- naker.pyNN.models.neuron.implementations.abstract_neuron_impl.AbstractNeuronImpl method), 245	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.implementations.abstract_neuron_impl.AbstractNeuronImpl method), 245
<code>get_spikes_sampling_interval()</code>	(spyn- naker.pyNN.models.neuron.implementations.AbstractNeuronImpl method), 251	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.implementations.AbstractNeuronImpl method), 245
<code>get_spikes_sampling_interval()</code>	(spyn- naker.pyNN.models.utility_models.spike_injector.spike_injector_vertex.SpikeInjectorVertex method), 263	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.implementations.neuron_impl_standard.AbstractNeuronImplStandard. method), 127
<code>get_standard_deviation()</code> (in module spyn- naker.pyNN.utilities.utility_calls), 286		get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.implementations.NeuronImplStandard. method), 127
<code>get_static_synaptic_data()</code>	(spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics. method), 173	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics. method), 203
<code>get_static_synaptic_data()</code>	(spyn- naker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamics. method), 188	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.synapse_types.AbstractSynapseType. method), 211
<code>get_static_synaptic_data()</code>	(spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics. method), 176	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.SynapseTypeAlpha. method), 203
<code>get_static_synaptic_data()</code>	(spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics. method), 176	get_synapse_id_by_target() (spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.SynapseTypeAlpha. method), 203

<code>naker.pyNN.models.abstract_models.AbstractContainsUnits</code>	<code>naker.pyNN.models.neuron.neuron_models.neuron_model_leaky</code>
<code>method</code>), 59	<code>method</code>), 145
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neuron.neuron_models.NeuronModelIzh</code>
<code>method</code>), 227	<code>method</code>), 147
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIn</code>
<code>method</code>), 238	<code>method</code>), 149
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.additional_inputs.additional_input_type_additional_input_type_alpha</code>	<code>naker.pyNN.models.neuron.additional_inputs.additional_input_type_additional_input_type_alpha</code>
<code>method</code>), 109	<code>method</code>), 204
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.additional_inputs.AdditionalInputTypeDelta</code>	<code>naker.pyNN.models.neuron.synapse_types.synapse_type_delta</code>
<code>method</code>), 111	<code>method</code>), 205
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.implementations.abstract_neuron_impl_abstract_neuron_impl</code>	<code>naker.pyNN.models.neuron.implementations.abstract_neuron_impl_abstract_neuron_impl</code>
<code>method</code>), 117	<code>method</code>), 206
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.implementations.abstract_standard_neuron_impl_abstract_standard_neuron_impl</code>	<code>naker.pyNN.models.neuron.implementations.abstract_standard_neuron_impl_abstract_standard_neuron_impl</code>
<code>method</code>), 118	<code>method</code>), 208
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.implementations.AbstractNeuronImpl</code>	<code>naker.pyNN.models.neuron.synapse_types.synapse_type_semd</code>
<code>method</code>), 129	<code>method</code>), 210
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.implementations.AbstractStandardNeuronImpl</code>	<code>naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha</code>
<code>method</code>), 124	<code>method</code>), 216
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.implementations.neuron_impl_standard_neuron_impl_standard_neuron_impl</code>	<code>naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta</code>
<code>method</code>), 121	<code>method</code>), 215
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.implementations.NeuronImplStandardNeuronImpl</code>	<code>naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExpo</code>
<code>method</code>), 127	<code>method</code>), 212
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.input_types.input_type_conductance</code>	<code>naker.pyNN.models.neuron.synapse_types.SynapseTypeExponenti</code>
<code>method</code>), 131	<code>method</code>), 213
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.input_types.input_type_current</code>	<code>naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMD</code>
<code>method</code>), 132	<code>method</code>), 217
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.input_types.input_type_current</code>	<code>naker.pyNN.models.neuron.threshold_types.threshold_type_maass</code>
<code>method</code>), 133	<code>method</code>), 219
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.input_types.InputTypeConductance</code>	<code>naker.pyNN.models.neuron.threshold_types.threshold_type_static</code>
<code>method</code>), 135	<code>method</code>), 220
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code>	<code>naker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass</code>
<code>method</code>), 136	<code>method</code>), 222
<code>get_units()</code>	<code>(spyn- get_units()</code>
<code>naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD</code>	<code>naker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic</code>
<code>method</code>), 137	<code>method</code>), 221
<code>get_units()</code>	<code>(spyn- get_value()</code>
<code>naker.pyNN.models.neuron.neuron_models.neuron_model_leaky</code>	<code>naker.pyNN.models.neuron_models.neuron_model_leaky</code>
<code>method</code>), 144	<code>method</code>), 58
<code>get_units()</code>	<code>(spyn- get_value()</code>

<code>naker.pyNN.models.abstract_models.AbstractSettable</code> <code>method</code>), 60	<code>naker.pyNN.models.neuron.implementations.AbstractStandardNeuron</code> <code>method</code>), 124
<code>get_value()</code> <code>naker.pyNN.models.common.simple_population_settable.SimplePopulationSettable</code> <code>method</code>), 66	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance</code> <code>method</code>), 131
<code>get_value()</code> <code>naker.pyNN.models.common.SimplePopulationSettable</code> <code>method</code>), 70	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent</code> <code>method</code>), 132
<code>get_value()</code> <code>naker.pyNN.models.neural_properties.neural_parameter.NeuralParameter</code> <code>method</code>), 108	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.input_types.input_type_current_semd.InputTypeCurrentSEMD</code> <code>method</code>), 133
<code>get_value()</code> <code>naker.pyNN.models.neural_properties.NeuralParameter</code> <code>method</code>), 108	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.input_types.InputTypeConductance</code> <code>method</code>), 135
<code>get_value()</code> <code>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</code> <code>method</code>), 227	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code> <code>method</code>), 136
<code>get_value()</code> <code>naker.pyNN.models.neuron.AbstractPopulationVertex</code> <code>method</code>), 239	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD</code> <code>method</code>), 137
<code>get_value()</code> <code>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.StaticSynapseDynamics</code> <code>method</code>), 177	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron_model_izh.NeuronModelIzh</code> <code>method</code>), 144
<code>get_value()</code> <code>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.STDP</code> <code>method</code>), 179	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron_model_leaky.NeuronModelLeakyIntegrator</code> <code>method</code>), 145
<code>get_value()</code> <code>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</code> <code>method</code>), 190	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.neuron_models.NeuronModelIzh</code> <code>method</code>), 148
<code>get_value()</code> <code>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</code> <code>method</code>), 192	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrator</code> <code>method</code>), 149
<code>get_value_by_selector()</code> <code>naker.pyNN.models.abstract_models.abstract_population_settable.AbstractPopulationSettable</code> <code>method</code>), 57	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.SynapseTypeAlpha</code> <code>method</code>), 204
<code>get_value_by_selector()</code> <code>naker.pyNN.models.abstract_models.AbstractPopulationSettable</code> <code>method</code>), 60	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta</code> <code>method</code>), 205
<code>get_values()</code> <code>naker.pyNN.external_devices_models.threshold_type_multicausal_device.MulticausalDevice</code> <code>method</code>), 46	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron_model_type_multicausal_device.MulticausalDeviceGeneralExp</code> <code>method</code>), 207
<code>get_values()</code> <code>naker.pyNN.external_devices_models.ThresholdTypeMulticausalDevice</code> <code>method</code>), 54	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential</code> <code>method</code>), 208
<code>get_values()</code> <code>naker.pyNN.models.neuron.additional_inputs.additional_inputs.AdditionalInputCa2Ap</code> <code>method</code>), 109	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.synapse_types.sem_d.SynapseTypeSEMD</code> <code>method</code>), 210
<code>get_values()</code> <code>naker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2Ap</code> <code>method</code>), 111	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha</code> <code>method</code>), 216
<code>get_values()</code> <code>naker.pyNN.models.neuron.implementations.abstract_standard_neuron.AbstractStandardNeuron</code> <code>method</code>), 119	<code>(spyn- get_values()</code> <code>naker.pyNN.models.neuron.implementations.AbstractStandardNeuron</code> <code>method</code>), 215
<code>get_values()</code>	<code>(spyn- get_values()</code>

`naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential` (method), 192

`get_values()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` method), 212

`get_values()` (spyn- `naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential` method), 195

`get_values()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` method), 213

`get_values()` (spyn- `naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMI` method), 197

`get_values()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` method), 218

`get_values()` (spyn- `naker.pyNN.models.neuron.threshold_types.threshold_type_mathematical_stochastic.ThresholdTypeMaassStochastic` method), 219

`get_values()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_dynamics.ThresholdTypeStatic` method), 220

`get_values()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.AbstractSynapseDynamics.ThresholdTypeMaassStochastic` method), 222

`get_values()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_dynamics.ThresholdTypeStatic` method), 221

`get_variable_sdram_usage()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.synapse_dynamics.ThresholdTypeStatic` method), 151

`get_variable_sdram_usage()` (spyn- `naker.pyNN.models.common.neuron_recorder.NeuronRecorder` method), 65

`get_variable_sdram_usage()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseDynamics` method), 151

`get_variance()` (in module `spyn- naker.pyNN.utilities.utility_calls`), 286

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.SynapseDynamics` method), 174

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics` method), 187

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 177

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 179

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 181

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 183

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 185

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` method), 190

`get_vertex_executable_suffix()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` method), 192

`get_weight_half_word()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_dynamics.ThresholdTypeStatic` method), 151

`get_weight_half_word()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseDynamics` method), 151

`get_weight_half_word()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.SynapseDynamics` method), 151

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.SynapseDynamics` method), 72

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics` method), 87

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 74

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 89

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 75

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 90

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics` method), 75

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` method), 90

`get_weight_maximum()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics` method), 92

Index 347

[method](#)), 185
[get_weight_mean\(\)](#) ([spyn- get_y_from_robot_retina\(\)](#) (in module [spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics](#)), 43
[method](#)), 192
[get_weight_mean\(\)](#) ([spyn- global_struct](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics](#)), 46
[method](#)), 197
[get_weight_mean\(\)](#) ([spyn- global_struct](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics](#)), 143
[method](#)), 199
[get_weight_variance\(\)](#) ([spyn- graph_mapper\(\)](#) ([spyn- naker.pyNN.models.neural_projections.connectors.abstract_neuron.neuron_models.push_bot.push_bot_spinnaker](#)), 31
[method](#)), 72
[get_weight_variance\(\)](#) ([spyn- graph_mapper\(\)](#) ([spyn- naker.pyNN.models.neural_projections.connectors.AbstractConnector](#)), 35
[method](#)), 87
[get_weight_variance\(\)](#) ([spyn- GraphEdgeFilter](#) (class in [spyn- naker.pyNN.models.neural_projections.connectors.from_list_of_edges_to_graph_mapper](#)), 269
[method](#)), 81
[get_weight_variance\(\)](#) ([spyn- GraphEdgeWeightUpdater](#) (class in [spyn- naker.pyNN.models.neural_projections.connectors.FromListConnector](#)), 269
[method](#)), 96
[get_weight_variance\(\)](#) ([spyn- grid\(\)](#) (in module [spynaker.plot_utils](#)), 294
[method](#)), 174
H
[get_weight_variance\(\)](#) ([spyn- has_parameter\(\)](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics](#)), 265
[method](#)), 187
[get_weight_variance\(\)](#) ([spyn- has_reset_last](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP](#)), 293
[method](#)), 179
[get_weight_variance\(\)](#) ([spyn- has_reset_last](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural.StaticSynapseDynamicsStructuralStatic](#)), 285
[method](#)), 183
[get_weight_variance\(\)](#) ([spyn- has_variable\(\)](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP](#)), 47
[method](#)), 192
[get_weight_variance\(\)](#) ([spyn- has_variable\(\)](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic](#)), 54
[method](#)), 197
[get_words\(\)](#) ([spyn- has_variable\(\)](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics](#)), 110
[method](#)), 175
[get_words\(\)](#) ([spyn- has_variable\(\)](#) ([spyn- naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics](#)), 111
[method](#)), 187
[get_x_from_fpga_retina\(\)](#) (in module [spyn- has_variable\(\)](#) ([spyn- naker.pyNN.external_devices_models.external_spinnaker_link.fpga_retina_device](#)), 119
[43](#)
[get_x_from_robot_retina\(\)](#) (in module [spyn- has_variable\(\)](#) ([spyn- naker.pyNN.external_devices_models.munich_spinnaker_link.fpga_retina_device](#)), 124
[46](#)
[get_y_from_fpga_retina\(\)](#) (in module [spyn- has_variable\(\)](#) ([spyn- naker.pyNN.external_devices_models.external_spinnaker_link.fpga_retina_device](#)), 119
[naker.pyNN.models.neuron.input_types.input_type_conductance.I](#)

`method`), 131
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent` `naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMD`
`method`), 132 `method`), 218
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.input_type_current.SynapseTypeSEMD` `naker.pyNN.models.neuron.threshold_types.threshold_type_maass`
`method`), 133 `method`), 219
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.InputTypeConductance` `naker.pyNN.models.neuron.threshold_types.threshold_type_static`
`method`), 135 `method`), 220
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.InputTypeCurrent` `naker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass`
`method`), 136 `method`), 223
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD` `naker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic`
`method`), 137 `method`), 222
`has_variable()` (spyn- `heat_plot()` (in module `spynaker.plot_utils`), 294
`naker.pyNN.models.neuron.neuron_models.neuron_model_exp.NeuronModelExp` `in` `spyn-`
`method`), 144 `naker.pyNN.models.neuron.builds`), 114
`has_variable()` (spyn- `HHCondExp` (class `in` `spyn-`
`naker.pyNN.models.neuron.neuron_models.neuron_model_leaky.NeuronModelLeaky` `naker.pyNN.models.neuron.neuron_models.neuron_model_integrate_and_fire.NeuronModelIntegrateAndFire`
`method`), 145 112
`has_variable()` (spyn- `high()` (in module `spyn-`
`naker.pyNN.models.neuron.neuron_models.NeuronModelExp` `naker.pyNN.utilities.utility_calls`), 286
`method`), 148 `high()` (`spynaker.pyNN.utilities.random_stats.abstract_random_stats.Ab-`
`has_variable()` (spyn- `method`), 280
`naker.pyNN.models.neuron.neuron_models.NeuronModelLeaky` `spynaker.pyNN.utilities.random_stats.AbstractRandomStats`
`method`), 149 `method`), 280
`has_variable()` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.SynapseTypeAlpha`
`method`), 204 `i_alpha` (`spynaker.pyNN.models.neuron.additional_inputs.additional_in-`
`has_variable()` (spyn- `attribute`), 110
`naker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta` `naker.pyNN.models.neuron.additional_inputs.AdditionalInp-`
`method`), 205 `attribute`), 111
`has_variable()` (spyn- `i_ca2` (`spynaker.pyNN.models.neuron.additional_inputs.additional_inpu-`
`naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential`
`method`), 207 `attribute`), 111
`has_variable()` (spyn- `attribute`), 111
`naker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential` `naker.pyNN.models.neuron.neuron_models.neuron_model-`
`method`), 208 `attribute`), 144
`has_variable()` (spyn- `i_offset` (`spynaker.pyNN.models.neuron.neuron_models.neuron_model-`
`naker.pyNN.models.neuron.synapse_types.synapse_type_semd.SynapseTypeSEMD` `attribute`), 148
`method`), 210 `i_offset` (`spynaker.pyNN.models.neuron.neuron_models.NeuronModel-`
`has_variable()` (spyn- `attribute`), 148
`naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha` `naker.pyNN.models.neuron.neuron_models.NeuronModel-`
`method`), 216 `attribute`), 149
`has_variable()` (spyn- `id_counter` (`spynaker.pyNN.abstract_spinnaker_common.AbstractSpiN-`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta` `attribute`), 289
`method`), 215 `id_to_index()` (spyn-
`has_variable()` (spyn- `naker.pyNN.models.pyNN_population_common.PyNNPopulationC-`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential` `method`), 212
`method`), 212 `id_to_local_index()` (spyn-
`has_variable()` (spyn- `naker.pyNN.models.pyNN_population_common.PyNNPopulationC-`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential` `method`), 267

IFCondAlpha	(class in spyn- naker.pyNN.models.neuron.builds), 114	index_expression (spyn- naker.pyNN.models.neural_projections.connectors.IndexBasedPr
IFCondAlpha	(class in spyn- naker.pyNN.models.neuron.builds.if_cond_alpha),index_to_id()	attribute), 96 (spyn- naker.pyNN.models.pynn_population_common.PyNNPopulationC
IFCondExpBase	(class in spyn- naker.pyNN.models.neuron.builds), 114	method), 267
IFCondExpBase	(class in spyn- naker.pyNN.models.neuron.builds.if_cond_exp_base),	IndexBasedProbabilityConnector (class in spyn- naker.pyNN.models.neural_projections.connectors),
IFCondExpStoc	(class in spyn- naker.pyNN.models.neuron.builds), 115	96
IFCondExpStoc	(class in spyn- naker.pyNN.models.neuron.builds.if_cond_exp_stoc),	IndexBasedProbabilityConnector (class in spyn- naker.pyNN.models.neural_projections.connectors.index_based_p
IFCurrAlpha	(class in spyn- naker.pyNN.models.neuron.builds), 114	81
IFCurrAlpha	(class in spyn- naker.pyNN.models.neuron.builds.if_curr_alpha),	inh_input_previous (spyn- naker.pyNN.models.neuron.input_types.input_type_current_semd
IFCurrDelta	(class in spyn- naker.pyNN.models.neuron.builds), 115	attribute), 134
IFCurrDelta	(class in spyn- naker.pyNN.models.neuron.builds.if_curr_delta),	inh_input_previous (spyn- naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD
IFCurrDualExpBase	(class in spyn- naker.pyNN.models.neuron.builds), 114	attribute), 137
IFCurrDualExpBase	(class in spyn- naker.pyNN.models.neuron.builds.if_curr_dual_exp_base),	inh_response (spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.Sy
IFCurrExpBase	(class in spyn- naker.pyNN.models.neuron.builds), 115	attribute), 204
IFCurrExpBase	(class in spyn- naker.pyNN.models.neuron.builds.if_curr_exp_base),	inh_response (spyn- naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha
IFCurrExpCa2Adaptive	(class in spyn- naker.pyNN.models.neuron.builds), 115	attribute), 216
IFCurrExpCa2Adaptive	(class in spyn- naker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive),	initial_delay (spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_
IFCurrExpSEMDBase	(class in spyn- naker.pyNN.models.neuron.builds), 115	attribute), 175
IFCurrExpSEMDBase	(class in spyn- naker.pyNN.models.neuron.builds.if_curr_exp_semd_base),	initial_delay (spyn- naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseD
IFFacetsConductancePopulation	(class in spynnaker.pyNN.models.neuron.builds), 115	attribute), 193
IFFacetsConductancePopulation	(class in spyn- naker.pyNN.models.neuron.builds.if_facets_hardwareI),	initial_delay (spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics
index_expression	(spyn- naker.pyNN.models.neural_projections.connectors.index_based_probabi	attribute), 181
		initial_values (spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics
		attribute), 185
		initial_delay (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsS
		initial_delay (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsS
		attribute), 195
		initial_delay (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsS
		attribute), 197
		initial_delay (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsS
		attribute), 199
		initial_values (spyn- naker.pyNN.models.abstract_models.abstract_population_initiali
		attribute), 82
		initial_values (spyn- naker.pyNN.models.neural_projections.connectors.IndexBasedProbabilityConnector

Index 353

`naker.pyNN.protocols.MunichIoEthernetProtocol` (static method), 275
`LED_TOTAL_PERIOD` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLED` attribute), 26
`LED_TOTAL_PERIOD` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotLED` attribute), 28
`led_total_period()` (spyn- `naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol` static method), 270
`led_total_period()` (spyn- `naker.pyNN.protocols.MunichIoEthernetProtocol` static method), 275
`LEFT_RETINA` (spyn- `naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice` attribute), 45
`LEFT_RETINA` (spyn- `naker.pyNN.external_devices_models.MunichRetinaDevice` attribute), 53
`LEFT_RETINA_DISABLE` (spyn- `naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice` attribute), 45
`LEFT_RETINA_DISABLE` (spyn- `naker.pyNN.external_devices_models.MunichRetinaDevice` attribute), 53
`LEFT_RETINA_ENABLE` (spyn- `naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice` attribute), 45
`LEFT_RETINA_ENABLE` (spyn- `naker.pyNN.external_devices_models.MunichRetinaDevice` attribute), 53
`LEFT_RETINA_KEY_SET` (spyn- `naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice` attribute), 45
`LEFT_RETINA_KEY_SET` (spyn- `naker.pyNN.external_devices_models.MunichRetinaDevice` attribute), 53
`line_plot()` (in module `spynaker.plot_utils`), 294
`list_factory()` (spyn- `naker.pyNN.utilities.ranged.spynaker_ranged_dictionary.SpynakerRangeDictionary` method), 281
`list_factory()` (spyn- `naker.pyNN.utilities.ranged.SpynakerRangeDictionary` method), 282
`local_host` (`spynaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_viewer.PushBotRetinaViewer` attribute), 27
`local_host` (`spynaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotRetinaViewer` attribute), 29
`local_ip_address` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot_wifi_connection.PushBotWiFiConnection` attribute), 18
`local_ip_address` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot_wifi_connection.PushBotWiFiConnection` attribute), 177

<i>method</i>), 179	<i>method</i>), 271	
mark_no_changes ()	(spyn- master_slave_set_slave ()	(spyn-
<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</i>	<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</i>	<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</i>
<i>method</i>), 190	<i>method</i>), 277	
mark_no_changes ()	(spyn- master_slave_use_internal_counter ()	
<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</i>	<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</i>	<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</i>
<i>method</i>), 192	<i>method</i>), 271	
mark_no_changes ()	(spyn- master_slave_use_internal_counter ()	
<i>naker.pyNN.models.pynn_population_common.PyNNPopulationCommon</i>	<i>naker.pyNN.models.pynn_population_common.PyNNPopulationCommon</i>	<i>naker.pyNN.models.pynn_population_common.PyNNPopulationCommon</i>
<i>method</i>), 267	<i>method</i>), 277	
mark_no_changes ()	(spyn- MasterPopTableAsBinarySearch (class in spyn-	
<i>naker.pyNN.models.pynn_projection_common.PyNNProjectionCommon</i>	<i>naker.pyNN.models.pynn_projection_common.PyNNProjectionCommon</i>	<i>naker.pyNN.models.pynn_projection_common.PyNNProjectionCommon</i>
<i>method</i>), 268	<i>method</i>), 140	
mark_no_changes ()	(spyn- MasterPopTableAsBinarySearch (class in spyn-	
<i>naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex</i>	<i>naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex</i>	<i>naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex</i>
<i>method</i>), 246	<i>method</i>), 139	
mark_no_changes ()	(spyn- MatrixGeneratorID (class in spyn-	
<i>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</i>	<i>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</i>	<i>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</i>
<i>method</i>), 251	<i>method</i>), 172	
mark_regions_reloaded ()	(spyn- max_delay (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon)	
<i>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</i>	<i>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</i>	<i>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</i>
<i>method</i>), 228	<i>method</i>), 280	
mark_regions_reloaded ()	(spyn- attribute), 293	
<i>naker.pyNN.models.neuron.AbstractPopulationVertex</i>	<i>naker.pyNN.models.neuron.AbstractPopulationVertex</i>	<i>naker.pyNN.models.neuron.AbstractPopulationVertex</i>
<i>method</i>), 239	<i>method</i>), 285	
mark_regions_reloaded ()	(spyn- MAX_RATE (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder)	
<i>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</i>	<i>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</i>	<i>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</i>
<i>method</i>), 251	<i>method</i>), 68	
MASTER_POP_ENTRY_DTYPE	(spyn- attribute), 68	
<i>naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators</i>
<i>attribute</i>), 139	<i>attribute</i>), 251	
MASTER_POP_ENTRY_DTYPE	(spyn- max_size (spynnaker.pyNN.exceptions.SynapseRowTooBigException)	
<i>naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators</i>
<i>attribute</i>), 140	<i>attribute</i>), 36	
master_slave_key	(spyn- attribute), 36	
<i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol</i>	<i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol</i>	<i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol</i>
<i>attribute</i>), 271	<i>attribute</i>), 36	
master_slave_key	(spyn- MaxRowInfo (class in spyn-	
<i>naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol</i>	<i>naker.pyNN.models.neuron.synapse_io.max_row_info</i>),	
<i>attribute</i>), 276	<i>attribute</i>), 200	
master_slave_set_master_clock_active ()	mean (spynnaker.pyNN.utilities.running_stats.RunningStats	
(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
<i>method</i>), 271	<i>method</i>), 280	
master_slave_set_master_clock_active ()	mean () (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats	
(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
<i>method</i>), 277	<i>method</i>), 280	
master_slave_set_master_clock_not_started	MeanReadException, 290	
(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
<i>method</i>), 271	<i>method</i>), 175	
master_slave_set_master_clock_not_started	merge () (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics	
(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol	(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
<i>method</i>), 277	<i>method</i>), 187	
master_slave_set_slave ()	merge () (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics	
(spyn- <i>method</i>), 177	(spyn- <i>method</i>), 177	
<i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol</i>	<i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics</i>	<i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics</i>

motor_1_permanent_velocity() (spyn- MunichMotorDevice (class in spyn-
naker.pyNN.protocols.MunichIoEthernetProtocol naker.pyNN.external_devices_models.munich_spinnaker_link_mo
static method), 275 43

MultipaseConnector (class in spyn- MunichRetinaDevice (class in spyn-
naker.pyNN.models.neural_projections.connectors), naker.pyNN.external_devices_models), 52
96

MultipaseConnector (class in spyn- MunichRetinaDevice (class in spyn-
naker.pyNN.models.neural_projections.connectors.multipase45connector),
84

multiplicator (spyn- MY_ORO_BOTICS (spyn-
naker.pyNN.models.neuron.input_types.input_type_current_attribute), 134 naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich
attribute), 134 MY_ORO_BOTICS (spyn-
naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD attribute), 276

multiplicator (spyn- N
naker.pyNN.models.neuron.synapse_types.synapse_type_send_synapse_type_SEMD
attribute), 210 naker.pyNN.models.neuron.synapse_types.synapse_type_send_synapse_type_SEMD
attribute), 218

multiplicator (spyn- (spynnaker.pyNN.models.neuron.population_machine_vertex.Pop
naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMD attribute), 233
attribute), 218 (spynnaker.pyNN.models.neuron.population_machine_vertex.Pop
attribute), 243

MultiSpikeRecorder (class in spyn- n_atoms (spynnaker.pyNN.external_devices_models.munich_spinnaker_li
naker.pyNN.models.common), 70 attribute), 45

MultiSpikeRecorder (class in spyn- n_atoms (spynnaker.pyNN.external_devices_models.MunichMotorDevice
naker.pyNN.models.common.multi_spike_recorder), 63 attribute), 52

munich_key() (in module spyn- n_atoms (spynnaker.pyNN.models.abstract_models.abstract_population_s
naker.pyNN.protocols.munich_io_spinnaker_link_protocol), attribute), 57
275 n_atoms (spynnaker.pyNN.models.abstract_models.AbstractPopulationSe
attribute), 60

munich_key_i() (in module spyn- n_atoms (spynnaker.pyNN.models.neuron.abstract_population_vertex.Abs
naker.pyNN.protocols.munich_io_spinnaker_link_protocol), attribute), 228
275 n_atoms (spynnaker.pyNN.models.neuron.AbstractPopulationVertex
attribute), 239

munich_key_i_d() (in module spyn- n_atoms (spynnaker.pyNN.models.spike_source.spike_source_poisson_ve
naker.pyNN.protocols.munich_io_spinnaker_link_protocol), attribute), 251
275 n_atoms (spynnaker.pyNN.models.utility_models.delays.delay_extension_
attribute), 257

MunichIoEthernetProtocol (class in spyn- n_atoms (spynnaker.pyNN.models.utility_models.delays.DelayExtensionV
naker.pyNN.protocols), 275 attribute), 261

MunichIoEthernetProtocol (class in spyn- n_atoms (spynnaker.pyNN.models.utility_models.delays.DelayExtensionV
naker.pyNN.protocols.munich_io_ethernet_protocol), attribute), 261
270

MunichIoSpiNNakerLinkProtocol (class in N_BUFFER_OVERFLOWS (spyn-
spynnaker.pyNN.protocols), 275 naker.pyNN.models.utility_models.delays.delay_extension_machi
attribute), 255

MunichIoSpiNNakerLinkProtocol (class in spyn- N_BUFFER_OVERFLOWS (spyn-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol), naker.pyNN.models.utility_models.delays.DelayExtensionMachi
270 attribute), 258

MunichIoSpiNNakerLinkProtocol.MODES N_BYTES_FOR_TIMESTAMP (spyn-
(class in spynnaker.pyNN.protocols), 276 naker.pyNN.models.common.neuron_recorder.NeuronRecorder
attribute), 63

MunichIoSpiNNakerLinkProtocol.MODES (class in spyn- N_BYTES_FOR_TIMESTAMP (spyn-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol), naker.pyNN.models.common.NeuronRecorder
271 attribute), 68

MunichMotorDevice (class in spyn- N_BYTES_PER_INDEX (spyn-
naker.pyNN.external_devices_models), 50 naker.pyNN.models.common.neuron_recorder.NeuronRecorder

attribute), 63		naker.pyNN.models.utility_models.delays.DelayExtensionMachine
N_BYTES_PER_INDEX	(spyn-	naker.pyNN.models.utility_models.delays.DelayExtensionMachine
naker.pyNN.models.common.NeuronRecorder		n_items (spynnaker.pyNN.utilities.running_stats.RunningStats
attribute), 68		attribute), 284
N_BYTES_PER_POINTER	(spyn-	n_neurons (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol
naker.pyNN.models.common.neuron_recorder.NeuronRecorder		attribute), 274
attribute), 63		n_neurons (spynnaker.pyNN.protocols.RetinaKey at-
N_BYTES_PER_POINTER	(spyn-	tribute), 279
naker.pyNN.models.common.NeuronRecorder		N_PACKETS_ADDED (spyn-
attribute), 68		naker.pyNN.models.utility_models.delays.delay_extension_machin
N_BYTES_PER_RATE	(spyn-	attribute), 255
naker.pyNN.models.common.neuron_recorder.NeuronRecorder		N_PACKETS_ADDED (spyn-
attribute), 63		naker.pyNN.models.utility_models.delays.DelayExtensionMachine
N_BYTES_PER_RATE	(spyn-	attribute), 258
naker.pyNN.models.common.NeuronRecorder		N_PACKETS_PROCESSED (spyn-
attribute), 68		naker.pyNN.models.utility_models.delays.delay_extension_machin
N_BYTES_PER_SIZE	(spyn-	attribute), 255
naker.pyNN.models.common.neuron_recorder.NeuronRecorder		N_PACKETS_PROCESSED (spyn-
attribute), 63		naker.pyNN.models.utility_models.delays.DelayExtensionMachine
N_BYTES_PER_SIZE	(spyn-	attribute), 259
naker.pyNN.models.common.NeuronRecorder		N_PACKETS_RECEIVED (spyn-
attribute), 68		naker.pyNN.models.utility_models.delays.delay_extension_machin
N_BYTES_PER_VALUE	(spyn-	attribute), 255
naker.pyNN.models.common.neuron_recorder.NeuronRecorder		N_PACKETS_RECEIVED (spyn-
attribute), 64		naker.pyNN.models.utility_models.delays.DelayExtensionMachine
N_BYTES_PER_VALUE	(spyn-	attribute), 259
naker.pyNN.models.common.NeuronRecorder		N_PACKETS_SENT (spyn-
attribute), 68		naker.pyNN.models.utility_models.delays.delay_extension_machin
N_CPU_CYCLES_PER_NEURON	(spyn-	attribute), 255
naker.pyNN.models.common.neuron_recorder.NeuronRecorder		N_PACKETS_SENT (spyn-
attribute), 64		naker.pyNN.models.utility_models.delays.DelayExtensionMachine
N_CPU_CYCLES_PER_NEURON	(spyn-	attribute), 259
naker.pyNN.models.common.NeuronRecorder		n_payload_bytes (spyn-
attribute), 68		naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Retina
n_delay_stages	(spyn-	attribute), 274
naker.pyNN.models.neural_projections.projection_application_model.ProjectionApplicationEdge		(spyn-
attribute), 103		naker.pyNN.protocols.RetinaPayload attribute),
n_delay_stages	(spyn-	280
naker.pyNN.models.neural_projections.ProjectionApplicationEdge		n_neurons (spyn-
attribute), 106		naker.pyNN.models.neural_projections.synapse_information.Synap
n_delay_stages	(spyn-	tribute), 104
naker.pyNN.models.utility_models.delays.delay_extension_machine_vertex.DelayExtensionVertex		(spyn-
attribute), 257		naker.pyNN.models.neural_projections.SynapseInformation
n_delay_stages	(spyn-	tribute), 107
naker.pyNN.models.utility_models.delays.DelayExtensionVertex		n_neurons (spyn-
attribute), 261		naker.pyNN.models.neural_projections.synapse_information.Synap
N_DELAYS (spynnaker.pyNN.models.utility_models.delays.delay_extension_machine_vertex.DelayExtensionMachineVertex.EXTRA_PL		tribute), 104
attribute), 255		n_pre_neurons (spyn-
N_DELAYS (spynnaker.pyNN.models.utility_models.delays.DelayExtensionMachineVertex.EXTRA_PL		naker.pyNN.models.utility_models.delays.DelayExtensionMachine
attribute), 258		tribute), 107
N_EXTRA_PROVENANCE_DATA_ENTRIES	(spyn-	n_weight_terms (spyn-
naker.pyNN.models.utility_models.delays.delay_extension_machine_vertex.DelayExtensionMachineVertex.EXTRA_PL		naker.pyNN.models.utility_models.delays.DelayExtensionMachine
attribute), 255		tribute), 152
N_EXTRA_PROVENANCE_DATA_ENTRIES	(spyn-	n_weight_terms (spyn-

`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011` (class in `spyn-
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`), 156

`n_weight_terms` (spyn- `needs_buffering()` (in module `spyn-
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`), 152

`n_weight_terms` (spyn- `naker.pyNN.models.common.recording_utils`),
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`, 153

`n_weight_terms` (spyn- `naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS`),
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`, 154

`n_weight_terms` (spyn- `naker.pyNN.models.neuron.implementations`),
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`, 155

`n_weight_terms` (spyn- `naker.pyNN.models.neuron.implementations.neuron_impl_standard`),
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`, 155

`n_weight_terms` (spyn- `naker.pyNN.models.neuron.neuron_models`),
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`, 157

`n_weight_terms` (spyn- `naker.pyNN.models.neuron.neuron_models.neuron_model_izh`),
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`, 158

`n_weight_terms` (spyn- (class in spyn-
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`), 158

`n_weight_terms` (spyn- `NeuronModelLeakyIntegrateAndFire` (class in spyn-
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`), 156

`n_weight_terms` (spyn- 145
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011`), 159

`n_words_for_plastic_connections()` (spyn- `NeuronParameter` (class in spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common`), 181

`n_words_for_plastic_connections()` (spyn- `NeuronRecorder` (class in spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common`), 195

`n_words_for_static_connections()` (spyn- `naker.pyNN.models.common.neuron_recorder`),
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common`, 181

`n_words_for_static_connections()` (spyn- `NO_PAYLOAD` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common`), 195

`NATIVE_128_X_128` (spyn- `NUMPY_CONNECTORS_DTYPE` (spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_payload`), 27

`NATIVE_128_X_128` (spyn- `NUMPY_CONNECTORS_DTYPE` (spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_payload`), 29

`NATIVE_128_X_128` (spyn- `numpy_dtype` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol`), 274

`NATIVE_128_X_128` (spyn- `numpy_dtype` (spyn-

`naker.pyNN.models.neuron.implementations.struct.Struct` `attribute`), 195
`attribute`), 123
`partner_selection` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon` `attribute`), 175
`NUMPY_SYNAPSES_DTYPE` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon` `attribute`), 193
`naker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector` `attribute`), 71
`partner_selection` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon` `attribute`), 182
`NUMPY_SYNAPSES_DTYPE` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon` `attribute`), 183
`naker.pyNN.models.neural_projections.connectors.AbstractConnector` `attribute`), 199
`pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.external_spinnaker_link_protocol.RetinaDevice` `attribute`), 42
`ONE_TO_ONE_CONNECTOR` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.external_spinnaker_link_protocol.RetinaDevice` `attribute`), 50
`naker.pyNN.models.neural_projections.connectors.abstract_connector_on_machine.ConnectorOnMachine` `attribute`), 73
`OneToOneConnector` (class in spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.RetinaDevice` `attribute`), 46
`naker.pyNN.models.neural_projections.connectors`), `naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.RetinaDevice` `attribute`), 46
`98`
`OneToOneConnector` (class in spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.MunichRetinaDevice` `attribute`), 53
`naker.pyNN.models.neural_projections.connectors.one_to_one_connector`), `naker.pyNN.external_devices_models.MunichRetinaDevice` `attribute`), 53
`85`
`pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.abstract_push_bot` `attribute`), 36
`P`
`p_rew` (spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon `attribute`), 181
`pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.AbstractPushBot` `attribute`), 37
`p_rew` (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon `attribute`), 195
`PARAMS_REGION` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 13
`naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.MunichMotorDevice` `attribute`), 43
`PARAMS_REGION` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 14
`naker.pyNN.external_devices_models.MunichMotorDevice` `attribute`), 51
`PARAMS_SIZE` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 14
`naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.MunichMotorDevice` `attribute`), 43
`PARAMS_SIZE` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 16
`naker.pyNN.external_devices_models.MunichMotorDevice` `attribute`), 51
`partner_selection` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 20
`naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon` `attribute`), 175
`partner_selection` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 21
`naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamicsStructuralCommon` `attribute`), 193
`partner_selection` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 22
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon` `attribute`), 182
`partner_selection` (spyn- `pause_stop_commands` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne` `attribute`), 23
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStatic` `attribute`), 183
`partner_selection` (spyn- `pixels` (spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey `attribute`), 274
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp.SynapseDynamicsStructuralSTDP` `attribute`), 186
`pixels` (spynaker.pyNN.protocols.RetinaKey `attribute`), 279
`partner_selection` (spyn- `PLASTIC_SYNAPTIC_WEIGHT_SATURATION_COUNT` `attribute`), 279
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon`

(spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES attribute), 233

plot_spikes() (in module spynnaker.plot_utils), 294

POISSON_PARAMS_REGION (spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex attribute), 247

poll_individual_sensor_continuously() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 272

poll_individual_sensor_continuously() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 181

poll_individual_sensor_continuously_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272

poll_individual_sensor_continuously_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 104

poll_sensors_once() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 272

poll_sensors_once() (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats method), 280

poll_sensors_once() (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats method), 280

poll_sensors_once_key (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272

poll_sensors_once_key (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 104

POPULATION_BASED_REGIONS (class in spynnaker.pyNN.utilities.constants), 283

POPULATION_TABLE (spynnaker.pyNN.utilities.constants attribute), 181

POPULATION_TABLE (spynnaker.pyNN.utilities.constants attribute), 283

PopulationMachineVertex (class in spynnaker.pyNN.models.neuron), 242

PopulationMachineVertex (class in spynnaker.pyNN.models.neuron.population_machine_vertex), 232

PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES (class in spynnaker.pyNN.models.neuron), 242

PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES (class in spynnaker.pyNN.models.neuron.population_machine_vertex), 233

port (spynnaker.pyNN.models.utility_models.spike_injector.spike_injector.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES attribute), 263

positions (spynnaker.pyNN.models.pyNN_population_common.pyNNPopulationCommon attribute), 267

post_as_pre() (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract attribute), 152

pre_trace_n_bytes (spyn- naker.pyNN.models.neural_projections), 106
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractTimingDependence (class in spyn-
 attribute), 156 naker.pyNN.models.neural_projections.projection_application_edge

pre_trace_n_bytes (spyn- 103
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_pfister_spike_triplet.TimingDependencePfisterSpikeTriplet (class in spyn-
 attribute), 153 naker.pyNN.models.neural_projections),

pre_trace_n_bytes (spyn- 106
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent.TimingDependenceRecurrent (class in spyn-
 attribute), 153 naker.pyNN.models.neural_projections.projection_machine_edge

pre_trace_n_bytes (spyn- 103
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair.TimingDependenceSpikePair (class in spyn-
 attribute), 154 attribute), 12

pre_trace_n_bytes (spyn- protocol (spynnaker.pyNN.external_devices_models.push_bot.push_bot_protocol.TimingDependenceSpikePair
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair.TimingDependenceSpikePair
 attribute), 155 protocol_instance (spyn-

pre_trace_n_bytes (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels_2011.TimingDependenceVogels2011 (class in spyn-
 attribute), 155 protocol_instance (spyn-

pre_trace_n_bytes (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_pfister_spike_triplet.TimingDependencePfisterSpikeTriplet
 attribute), 157 protocol_property (spyn-

pre_trace_n_bytes (spyn- naker.pyNN.external_devices_models.push_bot.abstract_push_bot_protocol.TimingDependenceRecurrent
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_recurrent.TimingDependenceRecurrent
 attribute), 158 protocol_property (spyn-

pre_trace_n_bytes (spyn- naker.pyNN.external_devices_models.push_bot.AbstractPushBotProtocol.TimingDependenceSpikeNearestPair
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair.TimingDependenceSpikeNearestPair
 attribute), 158 PROVENANCE_DATA (spyn-

pre_trace_n_bytes (spyn- naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair.TimingDependenceSpikePair
 attribute), 156 PROVENANCE_REGION (spyn-

pre_trace_n_bytes (spyn- naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels_2011.TimingDependenceVogels2011 (class in spyn-
 attribute), 159 pull_off_cached_lists() (in module spyn-

prepop_is_view (spyn- naker.pyNN.models.common), 71
 naker.pyNN.models.neural_projections.synapse_information.SynapseInformation (class in spyn-
 attribute), 104 naker.pyNN.models.common.recording_utils),

prepop_is_view (spyn- 66
 naker.pyNN.models.neural_projections.SynapseInformation (class in spyn-
 attribute), 107 attribute), 271

PROFILE_TAG_LABELS (spyn- PUSH_BOT (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex
 attribute), 233 push_bot_laser_config_active_time()

PROFILE_TAG_LABELS (spyn- (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.models.neuron.PopulationMachineVertex method), 272
 attribute), 243 push_bot_laser_config_active_time()

PROFILE_TAG_LABELS (spyn- (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex
 attribute), 247 push_bot_laser_config_active_time_key

PROFILER_REGION (spyn- (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex.POISSION_SPIKE
 attribute), 247 push_bot_laser_config_active_time_key

PROFILING (spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS, spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 attribute), 283 attribute), 277

ProjectionApplicationEdge (class in spyn- push_bot_laser_config_total_period()

```

(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_laser_config_total_period() push_bot_led_set_frequency_key (spy-
(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 277
push_bot_laser_config_total_period_key push_bot_led_total_period() (spy-
(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_laser_config_total_period_key push_bot_led_total_period() (spy-
(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 277
push_bot_laser_set_frequency() (spy- push_bot_led_total_period_key (spy-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_laser_set_frequency() (spy- push_bot_led_total_period_key (spy-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
method), 277 attribute), 277
push_bot_laser_set_frequency_key (spy- push_bot_motor_0_leaking_towards_zero()
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_laser_set_frequency_key (spy- push_bot_motor_0_leaking_towards_zero()
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
attribute), 277 method), 277
push_bot_led_back_active_time() (spy- push_bot_motor_0_leaking_towards_zero_key
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_led_back_active_time() (spy- push_bot_motor_0_leaking_towards_zero_key
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
method), 277 attribute), 277
push_bot_led_back_active_time_key (spy- push_bot_motor_0_permanent() (spy-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_led_back_active_time_key (spy- push_bot_motor_0_permanent() (spy-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
attribute), 277 method), 277
push_bot_led_front_active_time() (spy- push_bot_motor_0_permanent_key (spy-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_led_front_active_time() (spy- push_bot_motor_0_permanent_key (spy-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
method), 277 attribute), 277
push_bot_led_front_active_time_key push_bot_motor_1_leaking_towards_zero()
(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_led_front_active_time_key push_bot_motor_1_leaking_towards_zero()
(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
attribute), 277 method), 277
push_bot_led_set_frequency() (spy- push_bot_motor_1_leaking_towards_zero_key
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol attribute), 272
push_bot_led_set_frequency() (spy- push_bot_motor_1_leaking_towards_zero_key
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
method), 277 attribute), 277
push_bot_led_set_frequency_key (spy- push_bot_motor_1_permanent() (spy-

```


`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_motor_1_permanent()` (`spyn- PushBotEthernetDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 277

`push_bot_motor_1_permanent_key` (`spyn- PushBotEthernetDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_motor_1_permanent_key` (`spyn- PushBotEthernetDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 277

`push_bot_speaker_config_active_time()` (`spyn- PushBotEthernetLaserDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_speaker_config_active_time()` (`spyn- PushBotEthernetLaserDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 277

`push_bot_speaker_config_active_time_key` (`spyn- PushBotEthernetLEDDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_speaker_config_active_time_key` (`spyn- PushBotEthernetLEDDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 277

`push_bot_speaker_config_total_period()` (`spyn- PushBotEthernetMotorDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_speaker_config_total_period()` (`spyn- PushBotEthernetMotorDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 277

`push_bot_speaker_config_total_period_key` (`spyn- PushBotEthernetRetinaDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_speaker_config_total_period_key` (`spyn- PushBotEthernetRetinaDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 277

`push_bot_speaker_set_melody()` (`spyn- PushBotEthernetSpeakerDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_speaker_set_melody()` (`spyn- PushBotEthernetSpeakerDevice` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 278

`push_bot_speaker_set_melody_key` (`spyn- PushBotLaser` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_speaker_set_melody_key` (`spyn- PushBotLaser` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 278

`push_bot_speaker_set_tone()` (`spyn- PushBotLED` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 272

`push_bot_speaker_set_tone()` (`spyn- PushBotLED` (class in `spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`), 278

`push_bot_speaker_set_tone_key` (`spyn- PushBotLifEthernet` (class in `spyn-`

`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),` in `spyn-`
11 `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
PushBotLifEthernet (class in `spyn-` 30
`naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
10 (class in `spyn-`
PushBotLifSpinnakerLink (class in `spyn-` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),`
11 PushBotSpiNNakerLinkMotorDevice
PushBotLifSpinnakerLink (class in `spyn-` (class in `spyn-`
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
11 31
PushBotMotor (class in `spyn-` PushBotSpiNNakerLinkRetinaDevice
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),` in `spyn-`
28 `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
PushBotMotor (class in `spyn-` 34
`naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
26 (class in `spyn-`
PushBotRetinaConnection (class in `spyn-` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
`naker.pyNN.external_devices_models.push_bot.push_bot_ethernet),`
24 PushBotSpiNNakerLinkSpeakerDevice
PushBotRetinaConnection (class in `spyn-` (class in `spyn-`
`naker.pyNN.external_devices_models.push_bot.push_bot_ethernet),` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
17 35
PushBotRetinaResolution (class in `spyn-` PushBotSpiNNakerLinkSpeakerDevice
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),` in `spyn-`
28 `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
PushBotRetinaResolution (class in `spyn-` 32
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),` `spyn-`
27 `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet`
PushBotRetinaViewer (class in `spyn-` 24
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),`
29 PushBotRetinaViewer (class in `spyn-`
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet`
17 `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet`
PushBotRetinaViewer (class in `spyn-` 17
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet`
27 `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet`
PushBotSpeaker (class in `spyn-` 24
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),`
28 PushBotSpeakerConnection (class in `spyn-`
`naker.pyNN.external_devices_models.push_bot.push_bot_ethernet`
PushBotSpeaker (class in `spyn-` 17
`naker.pyNN.external_devices_models.push_bot.push_bot_control_modules),`
27 `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet`
PushBotSpiNNakerLinkLaserDevice (class in `spyn-` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
33 `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
PushBotSpiNNakerLinkLaserDevice (class in `spyn-` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
29 `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
PushBotSpiNNakerLinkLEDDevice (class in `spyn-` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
33 `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
PushBotSpiNNakerLinkLEDDevice (class in `spyn-` `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`
33 `naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker`

[pwm_pin_output_timer_a_channel_1_ratio\(\)](#) [pwm_pin_output_timer_b_duration_key](#)
 ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
 method), 278 attribute), 278
[pwm_pin_output_timer_a_channel_1_ratio_key](#) [pwm_pin_output_timer_c_channel_0_ratio\(\)](#)
 ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#) [spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M
 attribute), 273 method), 273
[pwm_pin_output_timer_a_channel_1_ratio_key](#) [pwm_pin_output_timer_c_channel_0_ratio\(\)](#)
 ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
 attribute), 278 method), 278
[pwm_pin_output_timer_a_duration\(\)](#) ([spyn- \[pwm_pin_output_timer_c_channel_0_ratio_key\]\(#\)](#)
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M [spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M
 method), 273 attribute), 273
[pwm_pin_output_timer_a_duration\(\)](#) ([spyn- \[pwm_pin_output_timer_c_channel_0_ratio_key\]\(#\)](#)
[naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
 method), 278 attribute), 278
[pwm_pin_output_timer_a_duration_key](#) [pwm_pin_output_timer_c_channel_1_ratio\(\)](#)
 ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#) [spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M
 attribute), 273 method), 273
[pwm_pin_output_timer_a_duration_key](#) [pwm_pin_output_timer_c_channel_1_ratio\(\)](#)
 ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
 attribute), 278 method), 278
[pwm_pin_output_timer_b_channel_0_ratio\(\)](#) [pwm_pin_output_timer_c_channel_1_ratio_key\(\)](#)
 ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#) [spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M
 method), 273 method), 273
[pwm_pin_output_timer_b_channel_0_ratio\(\)](#) [pwm_pin_output_timer_c_channel_1_ratio_key\(\)](#)
 ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
 method), 278 method), 278
[pwm_pin_output_timer_b_channel_0_ratio_key\(\)](#) [pwm_pin_output_timer_c_duration\(\)](#) ([spyn- \[pwm_pin_output_timer_c_channel_1_ratio_key\]\(#\)](#)
 ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#) [spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M
 method), 273 method), 273
[pwm_pin_output_timer_b_channel_0_ratio_key\(\)](#) [pwm_pin_output_timer_c_duration\(\)](#) ([spyn- \[pwm_pin_output_timer_c_channel_1_ratio_key\]\(#\)](#)
 ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
 method), 278 method), 278
[pwm_pin_output_timer_b_channel_1_ratio\(\)](#) [pwm_pin_output_timer_c_duration_key](#)
 ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#) [spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M
 method), 273 attribute), 273
[pwm_pin_output_timer_b_channel_1_ratio\(\)](#) [pwm_pin_output_timer_c_duration_key](#)
 ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
 method), 278 attribute), 278
[pwm_pin_output_timer_b_channel_1_ratio_key](#) [PyNNPopulationCommon](#) (class in [spyn- \[pwm_pin_output_timer_c_channel_1_ratio_key\]\(#\)](#)
 ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#) [spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M
 attribute), 273 266
[pwm_pin_output_timer_b_channel_1_ratio_key](#) [PyNNProjectionCommon](#) (class in [spyn- \[pwm_pin_output_timer_c_channel_1_ratio_key\]\(#\)](#)
 ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [spynnaker.pyNN.models.pyynn_projection_common](#)),
 attribute), 278 268
[pwm_pin_output_timer_b_duration\(\)](#) ([spyn- \[PyNNSynapseDynamics\]\(#\)](#) (class in [spyn- \[pwm_pin_output_timer_c_channel_1_ratio_key\]\(#\)](#)
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).M [spynnaker.pyNN.models.pyynn_synapse_dynamics](#)),
 method), 273 189
[pwm_pin_output_timer_b_duration\(\)](#) ([spyn- \[PyNNSynapseDynamics\]\(#\)](#) (class in [spyn- \[pwm_pin_output_timer_c_channel_1_ratio_key\]\(#\)](#)
[naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#) [naker.pyNN.models.neuron.synapse_dynamics.pyynn_synapse_dyn](#)
 method), 278 176
[pwm_pin_output_timer_b_duration_key](#)
 ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol](#).MunichIoSpiNNakerLinkProtocol
 attribute), 273 [query_state_of_io_lines\(\)](#) ([spyn-](#)

Index	369
--------------	------------

requires_mapping (spyn- attribute), 278
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSpikeSource (spyn-
attribute), 190 naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol

requires_mapping (spyn- attribute), 271
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDPLT (spyn-
attribute), 192 naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES

requires_mapping (spyn- attribute), 276
naker.pyNN.models.pynn_population_common.PyNNPopulationCommon.timestep() (spyn-
attribute), 267 naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex

requires_mapping (spyn- method), 229
naker.pyNN.models.pynn_projection_common.PyNNProjectionCommon.timestep() (spyn-
attribute), 269 naker.pyNN.models.neuron.AbstractPopulationVertex

requires_mapping (spyn- method), 240
naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex (spyn-
attribute), 246 naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex

requires_mapping (spyn- attribute), 233
naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex (spyn-
attribute), 252 naker.pyNN.models.neuron.PopulationMachineVertex

requires_memory_regions_to_be_reloaded() (attribute), 243
(spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex (spyn-
method), 228 naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex

requires_memory_regions_to_be_reloaded() (attribute), 248
(spynnaker.pyNN.models.neuron.AbstractPopulationVertex.method) requires_memory_regions_to_be_reloaded() (spyn-
method), 240 naker.pyNN.models.utility_models.delays.delay_extension_machine_vertex.DelayExtensionMachineVertex

requires_memory_regions_to_be_reloaded() (attribute), 255
(spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex (spyn-
method), 252 naker.pyNN.models.utility_models.delays.DelayExtensionMachineVertex

reserve_memory_regions() (spyn- attribute), 259
naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol (class in spynnaker.pyNN.external_devices_models), 279

reserve_memory_regions() (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol),
naker.pyNN.external_devices_models.MunichMotorDevice 274
method), 52 RetinaPayload (class in spynnaker.pyNN.protocols),

reserve_memory_regions() (spyn- 279
naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex in spyn-
method), 252 naker.pyNN.protocols.munich_io_spinnaker_link_protocol),

reset() (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface
method), 293 REWIRING_DATA_SIZE (spyn-
attribute), 181

reset() (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState
static method), 285 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamicsSpikeSource (spyn-
attribute), 190

reset_number_of_neurons_per_core() REWIRING_DATA_SIZE (spyn-
(spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon.method), 289 naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSpikeSource (spyn-
attribute), 194

reset_retina() (spyn- RIGHT_RETINA (spyn-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.method), 273 naker.pyNN.models.munich_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol (class in spynnaker.pyNN.protocols),

reset_retina() (spyn- RIGHT_RETINA (spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.external_devices_models.MunichRetinaDevice
method), 278 attribute), 53

reset_retina_key (spyn- RIGHT_RETINA_DISABLE (spyn-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.attribute), 273 naker.pyNN.models.munich_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol (class in spynnaker.pyNN.protocols),

reset_retina_key (spyn- RIGHT_RETINA_DISABLE (spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.external_devices_models.MunichRetinaDevice
attribute), 273 attribute), 45

attribute), 53
 RIGHT_RETINA_ENABLE (spyn- naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.PopulationVertex attribute), 45
 RIGHT_RETINA_ENABLE (spyn- naker.pyNN.external_devices_models.MunichRetinaDevice attribute), 53
 RIGHT_RETINA_KEY_SET (spyn- s_max (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse attribute), 175
 naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 193
 RIGHT_RETINA_KEY_SET (spyn- s_max (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 182
 naker.pyNN.external_devices_models.MunichRetinaDevice attribute), 184
 ring_buffer_sigma (spyn- s_max (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 186
 naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 195
 ring_buffer_sigma (spyn- s_max (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics attribute), 197
 naker.pyNN.models.neuron.AbstractPopulationVertex attribute), 199
 ring_buffer_sigma (spyn- s_max (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics attribute), 199
 naker.pyNN.models.neuron.synaptic_manager.SynapticManager attribute), 199
 ring_buffer_sigma (spyn- safe (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector attribute), 72
 naker.pyNN.models.neuron.SynapticManager attribute), 72
 rng (spynnaker.pyNN.models.neural_projections.synapse_information.SynapseInformation attribute), 104
 rng (spynnaker.pyNN.models.neural_projections.SynapseInformation naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 108
 routing_info() (spyn- SARK_BLOCK_SIZE (spyn- naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_retina_device.PushBotSpinnakerLinkRetinaDevice attribute), 68
 method), 31
 routing_info() (spyn- SATURATION_COUNT (spyn- naker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link_retina_device.PushBotSpinnakerLinkRetinaDevice attribute), 233
 method), 35
 routing_key_partition_atom_mapping() SATURATION_COUNT (spyn- naker.pyNN.external_devices_models.external_device_external_device_lif_control_vertex.ExternalDeviceLifControlVertex attribute), 242
 method), 41
 ROW_LENGTH_MASK (spyn- seed (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse attribute), 139
 naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch attribute), 139
 ROW_LENGTH_MASK (spyn- seed (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse attribute), 193
 naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableAsBinarySearch attribute), 140
 seed (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 182
 run() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpinnakerCommon attribute), 289
 method), 289
 run() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_viewer.PushBotRetinaViewer attribute), 27
 method), 27
 run() (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotRetinaViewer attribute), 29
 method), 29
 RunningStats (class in spyn- seed (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics attribute), 284
 naker.pyNN.utilities.running_stats), 284
 RUNTIME_SDP_PORT_SIZE (spyn- seed (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics attribute), 197
 naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 197

seed (spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex attribute), 252
set () (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 267
send () (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetConnection method), 19
send () (spynnaker.pyNN.external_devices_models.push_bot.push_bot_wifi.PushBotWiFiConnection method), 26
send_spike () (spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractStdpWeightDependence method), 5
send_spike () (spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpynnakerLiveSpikesConnection method), 162
send_spikes () (spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 267
send_spikes () (spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpynnakerLiveSpikesConnection method), 5
send_spikes () (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetConnection method), 12
send_type (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice attribute), 36
send_type (spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.PushBotEthernetConnection method), 13
SEND_TYPE_ACCUM (spynnaker.pyNN.external_devices_models.abstract_municast_controllable_device.SendType attribute), 39
SEND_TYPE_FRACT (spynnaker.pyNN.external_devices_models.abstract_municast_controllable_device.SendType attribute), 39
SEND_TYPE_INT (spynnaker.pyNN.external_devices_models.abstract_municast_controllable_device.SendType attribute), 39
SEND_TYPE_UACCUM (spynnaker.pyNN.external_devices_models.abstract_municast_controllable_device.SendType attribute), 39
SEND_TYPE_UFRACT (spynnaker.pyNN.external_devices_models.abstract_municast_controllable_device.SendType attribute), 39
SEND_TYPE_UINT (spynnaker.pyNN.external_devices_models.abstract_municast_controllable_device.SendType attribute), 39
SendType (class in spynnaker.pyNN.external_devices_models.abstract_municast_controllable_device), 39
sensor_transmission_key () (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 273
sensor_transmission_key () (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics method), 175
sent_mode_command () (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol static method), 273
sent_mode_command () (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics method), 184

`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp.SynapseDynamicsStructuralSTDP`
`method), 186`

`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic` `neurons_per_core()` (spyn-
`method), 197`

`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralSTDP` `neurons_per_core()` (spyn-
`method), 199`

`naker.pyNN.models.neuron.abstract_pynn_neuron_model.AbstractPynnNeuronModel`
`set_constraint()` (spyn- `class method), 230`

`naker.pyNN.models.pynn_population_common.PynnPopulationCommon` `atoms_per_core()` (spyn-
`method), 268`

`naker.pyNN.models.neuron.AbstractPynnNeuronModel`
`set_delay()` (spyn- `class method), 244`

`naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics` `spyn-`
`method), 175`

`naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson`
`set_delay()` (spyn- `class method), 247`

`naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics` `atoms_per_core()` (spyn-
`method), 187`

`naker.pyNN.models.spike_source.spike_source_poisson_variable.SpikeSourcePoissonVariable`
`set_delay()` (spyn- `class method), 249`

`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic` (spyn-
`method), 177`

`naker.pyNN.models.spike_source.SpikeSourcePoisson`
`set_delay()` (spyn- `class method), 253`

`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP` (spyn-
`method), 180`

`naker.pyNN.models.spike_source.SpikeSourcePoissonVariable`
`set_delay()` (spyn- `class method), 254`

`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic` `neurons_per_core()` (spyn-
`method), 190`

`naker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon`
`set_delay()` (spyn- `method), 289`

`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP` `neurons_per_core()` (spyn-
`method), 193`

`naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface`
`set_initial_value()` (spyn- `method), 293`

`naker.pyNN.models.abstract_models.abstract_population_initializable.AbstractPopulationInitializable` `spyn-`
`method), 56`

`naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState`
`set_initial_value()` (spyn- `method), 285`

`naker.pyNN.models.abstract_models.AbstractPopulationInitializable` `pattern_for_payload()` (spyn-
`method), 59`

`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`set_initial_value()` (spyn- `method), 273`

`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex` `pattern_for_payload()` (spyn-
`method), 229`

`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`set_initial_value()` (spyn- `method), 278`

`naker.pyNN.models.neuron.AbstractPopulationVertex` `set_output_pattern_for_payload_key`
`method), 240`

`(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.M`
`set_mapping_constraint()` (spyn- `attribute), 273`

`naker.pyNN.models.pynn_population_common.PynnPopulationCommon` `pattern_for_payload_key`
`method), 268`

`(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`set_max_atoms_per_core()` (spyn- `attribute), 278`

`naker.pyNN.models.pynn_population_common.PynnPopulationCommon` `connect_to_high_impedance()`
`method), 268`

`(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.M`
`set_mode()` (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
`method), 273`

`set_payload_pins_to_high_impedance()`
`set_mode()` (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol) `spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method), 278`

`set_payload_pins_to_high_impedance_key`
`set_mode_key` (spyn- `set_payload_pins_to_high_impedance_key`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`attribute), 273`

set_payload_pins_to_high_impedance_key (method), 10
 (spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.set_payload_pins_to_high_impedance_key (method), 279)

set_projection_information() (spynaker.pyNN.connections.spynnaker_poisson_control_connection.set_projection_information() (method), 7
 naker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector (spynaker.pyNN.connections.SpynnakerPoissonControlConnection.set_projection_information() (method), 10
 naker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector (spynaker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable.set_projection_information() (method), 62
 naker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector (spynaker.pyNN.models.common.AbstractNeuronRecordable.set_projection_information() (method), 67
 naker.pyNN.models.neural_projections.connectors.distance_dependent_probability_connector.DistanceDependentProbabilityConnector (spynaker.pyNN.models.common.eieio_spike_recorder.EIEIOSpikeRecorder.set_projection_information() (method), 63
 naker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector (spynaker.pyNN.models.common.EIEIOSpikeRecorder.set_projection_information() (method), 68
 naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector (spynaker.pyNN.models.common.neuron_recorder.NeuronRecorder.set_projection_information() (method), 65
 naker.pyNN.models.neural_projections.connectors.fixed_number_post_connector.FixedNumberPostConnector (spynaker.pyNN.models.common.NeuronRecorder.set_projection_information() (method), 70
 naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector (spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex.set_projection_information() (method), 229
 naker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector (spynaker.pyNN.models.neuron.AbstractPopulationVertex.set_projection_information() (method), 240
 naker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector (spynaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable.set_projection_information() (method), 62
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon (spynaker.pyNN.models.common.AbstractSpikeRecordable.set_projection_information() (method), 68
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStatic (spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex.set_projection_information() (method), 229
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp.SynapseDynamicsStructuralSTDP (spynaker.pyNN.models.neuron.AbstractPopulationVertex.set_projection_information() (method), 240
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon (spynaker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex.set_projection_information() (method), 246
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStatic (spynaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex.set_projection_information() (method), 252
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp.SynapseDynamicsStructuralSTDP (spynaker.pyNN.models.utility_models.spike_injector.spike_injector_v2.SpikeInjectorV2.set_rate() (spynnaker.pyNN.connections.spynnaker_poisson_control_connection.set_rate() (method), 260
 SpynnakerPoissonControlConnection.set_rate() (method), 6
 set_retina_key() (spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.set_retina_key() (method), 279)

method), 273

set_retina_key() (spyn- set_value() (spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.models.common.SimplePopulationSettable
method), 279 method), 70

set_retina_key_key (spyn- set_value() (spyn-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
attribute), 273 method), 229

set_retina_key_key (spyn- set_value() (spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.models.neuron.AbstractPopulationVertex
attribute), 279 method), 240

set_retina_transmission() (spyn- set_value() (spyn-
naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics
static method), 270 method), 177

set_retina_transmission() (spyn- set_value() (spyn-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
method), 273 method), 180

set_retina_transmission() (spyn- set_value() (spyn-
naker.pyNN.protocols.MunichIoEthernetProtocol naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics
static method), 275 method), 190

set_retina_transmission() (spyn- set_value() (spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics
method), 279 method), 193

set_retina_transmission_key (spyn- set_value() (spyn-
naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
attribute), 274 method), 252

set_retina_transmission_key (spyn- set_value_by_selector() (spyn-
naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol naker.pyNN.models.abstract_models.abstract_population_settable
attribute), 279 method), 57

set_space() (spyn- set_value_by_selector() (spyn-
naker.pyNN.models.neural_projections.connectors.abstract_neural_projections_connectors.AbstractNeuralProjectionsConnectors
method), 72 method), 60

set_space() (spyn- shape2word() (in module spyn-
naker.pyNN.models.neural_projections.connectors.AbstractNeuralProjectionsConnectors
method), 87 84

set_synapse_dynamics() (spyn- show_connection_set() (spyn-
naker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses
method), 55 method), 75

set_synapse_dynamics() (spyn- show_connection_set() (spyn-
naker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses naker.pyNN.models.neural_projections.connectors.CSAConnectors
method), 58 method), 90

set_synapse_dynamics() (spyn- SimplePopulationSettable (class in spyn-
naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex
method), 229 naker.pyNN.models.common), 70

set_synapse_dynamics() (spyn- SimplePopulationSettable (class in spyn-
naker.pyNN.models.neuron.AbstractPopulationVertex naker.pyNN.models.common.simple_population_settable),
method), 240 66

set_value() (spyn- SINGLE_BIT_FLAG_BIT (spyn-
naker.pyNN.models.abstract_models.abstract_settable.AbstractSettable
method), 58 method), 139

set_value() (spyn- SINGLE_BIT_FLAG_BIT (spyn-
naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators
attribute), 141

set_value() size (spynnaker.pyNN.models.neuron.generator_data.GeneratorData
attribute), 232

set_value() (spyn- naker.pyNN.models.common.simple_population_settable.SimplePopulationSettable
naker.pyNN.models.common.PyNNPopulationSettable)

attribute), 268
 size (spynnaker.pyNN.models.utility_models.delays.delay_extension.DelayExtensionData (spyn-
 attribute), 258
 size() (spynnaker.pyNN.models.pynn_projection_common.PyNNProjectionCommon
 method), 269
 slow (spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_synapse_dynamics.PyNNSynapseDynamics (spyn-
 attribute), 176
 slow (spynnaker.pyNN.models.neuron.synapse_dynamics.PyNNSynapseDynamics (spyn-
 attribute), 189
 SmallWorldConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors) 98
 SmallWorldConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors) 86
 source_vertex (spyn-
 naker.pyNN.models.utility_models.delays.delay_extension.DelayExtensionData (spyn-
 attribute), 257
 source_vertex (spyn-
 naker.pyNN.models.utility_models.delays.DelayExtensionData (spyn-
 attribute), 261
 space (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector (spyn-
 attribute), 72
 space (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector (spyn-
 attribute), 87
 SPEAKER_ACTIVE_TIME (spyn-
 naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters (spyn-
 attribute), 27
 SPEAKER_ACTIVE_TIME (spyn-
 naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters (spyn-
 attribute), 28
 speaker_active_time() (spyn-
 naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol (spyn-
 static method), 270
 speaker_active_time() (spyn-
 naker.pyNN.protocols.MunichIoEthernetProtocol (spyn-
 static method), 275
 speaker_frequency() (spyn-
 naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol (spyn-
 static method), 270
 speaker_frequency() (spyn-
 naker.pyNN.protocols.MunichIoEthernetProtocol (spyn-
 static method), 275
 SPEAKER_MELODY (spyn-
 naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters (spyn-
 attribute), 27
 SPEAKER_MELODY (spyn-
 naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters (spyn-
 attribute), 28
 SPEAKER_TONE (spyn-
 naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters (spyn-
 attribute), 28
 SPEAKER_TONE (spyn-
 naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters (spyn-
 attribute), 28

attribute), 240
 spikes_per_second (spyn- (module), 4
 naker.pyNN.models.neuron.synaptic_manager.SynapticManager.pyNN.connections.spynnaker_live_spikes_co
 attribute), 234 (module), 4
 spikes_per_second (spyn- spynnaker.pyNN.connections.spynnaker_poisson_contr
 naker.pyNN.models.neuron.SynapticManager (module), 5
 attribute), 242 spynnaker.pyNN.exceptions (module), 289
 SpikeSourceArray (class in spyn- spynnaker.pyNN.external_devices_models
 naker.pyNN.models.spike_source), 253 (module), 47
 SpikeSourceArray (class in spyn- spynnaker.pyNN.external_devices_models.abstract_etl
 naker.pyNN.models.spike_source.spike_source_array), (module), 37
 244 spynnaker.pyNN.external_devices_models.abstract_etl
 SpikeSourceArrayVertex (class in spyn- (module), 37
 naker.pyNN.models.spike_source.spike_source_array_vertex), spynnaker.pyNN.external_devices_models.abstract_etl
 245 (module), 38
 SpikeSourceFromFile (class in spyn- spynnaker.pyNN.external_devices_models.abstract_mu
 naker.pyNN.models.spike_source), 253 (module), 38
 SpikeSourceFromFile (class in spyn- spynnaker.pyNN.external_devices_models.arbitrary_fr
 naker.pyNN.models.spike_source.spike_source_from_file), (module), 39
 246 spynnaker.pyNN.external_devices_models.external_dev
 SpikeSourcePoisson (class in spyn- (module), 39
 naker.pyNN.models.spike_source), 253 spynnaker.pyNN.external_devices_models.external_dev
 SpikeSourcePoisson (class in spyn- (module), 40
 naker.pyNN.models.spike_source.spike_source_poisson), spynnaker.pyNN.external_devices_models.external_sp
 246 (module), 41
 SpikeSourcePoissonMachineVertex spynnaker.pyNN.external_devices_models.external_sp
 (class in spyn- (module), 42
 naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex), spynnaker.pyNN.external_devices_models.munich_spinn
 247 (module), 43
 SpikeSourcePoissonMachineVertex.POISSON_SPIKE_SOURCE_CONNECTIONS spynnaker.pyNN.external_devices_models.munich_spinn
 (class in spyn- (module), 45
 naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex), spynnaker.pyNN.external_devices_models.push_bot
 247 (module), 36
 SpikeSourcePoissonVariable (class in spyn- spynnaker.pyNN.external_devices_models.push_bot.abs
 naker.pyNN.models.spike_source), 254 (module), 36
 SpikeSourcePoissonVariable (class in spyn- spynnaker.pyNN.external_devices_models.push_bot.abs
 naker.pyNN.models.spike_source.spike_source_poisson_variable), (module), 36
 248 spynnaker.pyNN.external_devices_models.push_bot.pus
 SpikeSourcePoissonVertex (class in spyn- (module), 11
 naker.pyNN.models.spike_source.spike_source_poisson_vertex), pyNN.external_devices_models.push_bot.pus
 249 (module), 10
 SPOMNIBOT (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES spynnaker.pyNN.external_devices_models.push_bot.pus
 attribute), 271 (module), 11
 SPOMNIBOT (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES spynnaker.pyNN.external_devices_models.push_bot.pus
 attribute), 276 (module), 19
 spynnaker (module), 1, 294 spynnaker.pyNN.external_devices_models.push_bot.pus
 spynnaker.gsyn_tools (module), 293 (module), 11
 spynnaker.plot_utils (module), 294 spynnaker.pyNN.external_devices_models.push_bot.pus
 spynnaker.pyNN (module), 293 (module), 12
 spynnaker.pyNN.abstract_spinnaker_commonspynnaker.pyNN.external_devices_models.push_bot.pus
 (module), 288 (module), 13
 spynnaker.pyNN.connections (module), 7 spynnaker.pyNN.external_devices_models.push_bot.pus
 spynnaker.pyNN.connections.ethernet_command_cor(module), 14
 (module), 3 spynnaker.pyNN.external_devices_models.push_bot.pus

[spynnaker.pyNN.models.neural_projections.delay_application_edge](#) ([module](#)), 86
[spynnaker.pyNN.models.neural_projections.delay_machine_edge](#) ([module](#)), 101
[spynnaker.pyNN.models.neural_projections.delay_ild4_connection_edge](#) ([module](#)), 102
[spynnaker.pyNN.models.neural_projections.delay_ild4_edge](#) ([module](#)), 102
[spynnaker.pyNN.models.neural_projections.projected_application_edge](#) ([module](#)), 103
[spynnaker.pyNN.models.neural_projections.projected_line_edge](#) ([module](#)), 103
[spynnaker.pyNN.models.neural_projections.synapse_and_connection](#) ([module](#)), 104
[spynnaker.pyNN.models.neural_properties](#) ([module](#)), 108
[spynnaker.pyNN.models.neural_properties.neural_delay](#) ([module](#)), 108
[spynnaker.pyNN.models.neuron](#) ([module](#)), 235
[spynnaker.pyNN.models.neuron.abstract_population_neuron](#) ([module](#)), 223
[spynnaker.pyNN.models.neuron.abstract_pysynapse](#) ([module](#)), 229
[spynnaker.pyNN.models.neuron.abstract_pysynapse_model_standard](#) ([module](#)), 230
[spynnaker.pyNN.models.neuron.additional_synapses](#) ([module](#)), 110
[spynnaker.pyNN.models.neuron.additional_synapses_from_input_types.input_type](#) ([module](#)), 109
[spynnaker.pyNN.models.neuron.additional_synapses_to_output_types.input_type](#) ([module](#)), 109
[spynnaker.pyNN.models.neuron.builds](#) ([module](#)), 114
[spynnaker.pyNN.models.neuron.builds.eif_spinnaker_base_models](#) ([module](#)), 111
[spynnaker.pyNN.models.neuron.builds.hh_compartment](#) ([module](#)), 112
[spynnaker.pyNN.models.neuron.builds.if_compartment](#) ([module](#)), 112
[spynnaker.pyNN.models.neuron.builds.if_compartment_base](#) ([module](#)), 112
[spynnaker.pyNN.models.neuron.builds.if_compartment_step](#) ([module](#)), 112
[spynnaker.pyNN.models.neuron.builds.if_compartment_base_neuron_models](#) ([module](#)), 112
[spynnaker.pyNN.models.neuron.builds.if_compartment_base_neuron_models.neuron_r](#) ([module](#)), 113
[spynnaker.pyNN.models.neuron.builds.if_compartment_base_neuron_models.neuron_r](#) ([module](#)), 113
[spynnaker.pyNN.models.neuron.builds.if_compartment_base_neuron_models.plasticity](#) ([module](#)), 113
[spynnaker.pyNN.models.neuron.builds.if_compartment_base_neuron_models.plasticity.stdp](#) ([module](#)), 113
[spynnaker.pyNN.models.neuron.builds.if_compartment_base_neuron_models.plasticity.stdp.common](#)

(module), 150
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.plasticity.helper.structural_plasticity
 (module), 150
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.structural_plasticity
 (module), 151
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.abstract_synapse_plasticity
 (module), 150
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_set_structural_weight_decay
 (module), 151
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_set_dynamic_weight_on
 (module), 151
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 156
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.abstract_synapse_dynamics.abstract
 (module), 152
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 152
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 153
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 154
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 154
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 155
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 162
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.abstract_synapse_dynamics.abstract
 (module), 159
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.abstract_synapse_dynamics.abstract
 (module), 159
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 160
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 161
 spynnaker.pyNN.models.neuron.plasticity.spynnaker.pyNN.models.neuron.synapse_dynamics.abstract
 (module), 161
 spynnaker.pyNN.models.neuron.population_max_row_index
 (module), 232
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_io.max_row_index
 (module), 172
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_io.synapse_io
 (module), 172
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_types
 (module), 172
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_types.abstract
 (module), 166
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_types.abstract
 (module), 165
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_types.abstract
 (module), 165
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_types.abstract
 (module), 168
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_types.abstract
 (module), 167
 spynnaker.pyNN.models.neuron.structural_plasticity.spynnaker.pyNN.models.neuron.synapse_types.abstract

Index 381

SpynnakerDataSpecificationWriter	<i>(class in spyn- naker.pyNN.overridden_pacman_functions.spynnaker_data_specification_writer),</i> 269	<i>naker.pyNN.external_devices_models.ExternalFPGARetinaDevice</i> <i>attribute), 50</i>
SpynnakerException	290	<i>naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device</i> <i>attribute), 46</i>
SpynnakerExternalDevicePluginManager	<i>(class in spyn- naker.pyNN.spynnaker_external_device_plugin_manager),</i> 290	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.MunichRetinaDevice</i> <i>attribute), 53</i>
SpynnakerFailedState	<i>(class in spyn- naker.pyNN.utilities.spynnaker_failed_state),</i> 285	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.abstract_push_bot</i> <i>attribute), 36</i>
SpynnakerLiveSpikesConnection	<i>(class in</i> <i>spynnaker.pyNN.connections), 8</i>	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.AbstractPushBot</i> <i>attribute), 37</i>
SpynnakerLiveSpikesConnection	<i>(class in spyn- naker.pyNN.connections.spynnaker_live_spikes_connection),</i> 4	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 13</i>
SpYNNakerNeuronGraphNetworkSpecificationReport	<i>(class in spyn- naker.pyNN.utilities.spynnaker_neuron_network_specification_report),</i> 285	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 14</i>
SpynnakerPoissonControlConnection	<i>(class in</i> <i>spynnaker.pyNN.connections), 8</i>	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 14</i>
SpynnakerPoissonControlConnection	<i>(class in spyn- naker.pyNN.connections.spynnaker_poisson_control_connection),</i> 5	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 16</i>
SpynnakerRangeDictionary	<i>(class in spyn- naker.pyNN.utilities.ranged),</i> 282	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 20</i>
SpynnakerRangeDictionary	<i>(class in spyn- naker.pyNN.utilities.ranged.spynnaker_ranged_dict),</i> 281	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 21</i>
SpynnakerRangedList	<i>(class in spyn- naker.pyNN.utilities.ranged),</i> 282	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 22</i>
SpynnakerRangedList	<i>(class in spyn- naker.pyNN.utilities.ranged.spynnaker_ranged_list),</i> 281	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_etherne</i> <i>attribute), 23</i>
SpynnakerSimulatorInterface	<i>(class in spyn- naker.pyNN.spynnaker_simulator_interface),</i> 293	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_spinnak</i> <i>attribute), 31</i>
SpYNNakerSynapticMatrixReport	<i>(class in spyn- naker.pyNN.utilities.spynnaker_synaptic_matrix_report),</i> 285	<i>start_resume_commands</i> <i>naker.pyNN.external_devices_models.push_bot.push_bot_spinnak</i> <i>attribute), 35</i>
standard_deviation	<i>(spyn- naker.pyNN.utilities.running_stats.RunningStats</i> <i>attribute), 285</i>	<i>starts</i> (<i>spynnaker.pyNN.models.spike_source.spike_source_poisson_vert</i> <i>attribute), 252</i>
start	<i>(spynnaker.pyNN.models.spike_source.spike_source_poisson</i> <i>attribute), 252</i>	<i>std()</i> (<i>spynnaker.pyNN.utilities.random_stats.abstract_random_stats.Abs</i> <i>method), 280</i>
start_resume_commands	<i>(spyn- naker.pyNN.external_devices_models.external_spinnaker</i> <i>attribute), 43</i>	<i>std()</i> (<i>spynnaker.pyNN.utilities.random_stats.abstract_random_stats</i> <i>method), 281</i>
start_resume_commands	<i>(spyn-</i> <i>STDP_MATRIX</i>	<i>(spyn-</i>

stop()	(spynnaker.pyNN.abstract_spinnaker_common.AbstractSpinnakerCommon method), 289	stop()	(spynnaker.pyNN.models.neural_projections.ProjectionMachineEdge attribute), 107
Structure	(class in spynnaker.pyNN.models.neuron.implementations), 125	SYNAPSE_PARAMS	(spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 283
Structure	(class in spynnaker.pyNN.models.neuron.implementations.struct), 122	synapse_type	(spynnaker.pyNN.models.neural_projections.synapse_information.SynapseInformation attribute), 104
struct	(spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent attribute), 119	struct	(spynnaker.pyNN.models.neural_projections.SynapseInformation attribute), 104
struct	(spynnaker.pyNN.models.neuron.implementations.AbstractStandardNeuronComponent attribute), 124	struct	(spynnaker.pyNN.models.neural_projections.SynapseInformation attribute), 104
structure	(spynnaker.pyNN.models.pynn_population_common.PynnPopulationCommon attribute), 268	structure	(spynnaker.pyNN.models.neuron.synapse_dynamics), 189
synapse_dynamics	(spynnaker.pyNN.models.neural_projections.synapse_information.SynapseInformation attribute), 104	synapse_dynamics	(spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 176
synapse_dynamics	(spynnaker.pyNN.models.neural_projections.SynapseInformation attribute), 108	synapse_dynamics	(spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 190
synapse_dynamics	(spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 229	synapse_dynamics	(spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 178
synapse_dynamics	(spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 240	synapse_dynamics	(spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 194
synapse_dynamics	(spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager attribute), 234	synapse_dynamics	(spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 180
SYNAPSE_DYNAMICS	(spynnaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute), 283	SYNAPSE_DYNAMICS	(spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics attribute), 195
synapse_expander()	(in module spynnaker.pyNN.models.utility_models.synapse_expander.synapse_expander), 264	synapse_expander()	(in module spynnaker.pyNN.models.utility_models.synapse_expander.synapse_expander), 264
synapse_information	(spynnaker.pyNN.models.neural_projections.delayed_application_edge.DelayedApplicationEdge attribute), 102	synapse_information	(spynnaker.pyNN.models.neural_projections.delayed_application_edge.DelayedApplicationEdge attribute), 106
synapse_information	(spynnaker.pyNN.models.neural_projections.DelayedApplicationEdge attribute), 106	synapse_information	(spynnaker.pyNN.models.neural_projections.DelayedApplicationEdge attribute), 106
synapse_information	(spynnaker.pyNN.models.neural_projections.projection_application_edges.ProjectionApplicationEdges attribute), 103	synapse_information	(spynnaker.pyNN.models.neural_projections.projection_application_edges.ProjectionApplicationEdges attribute), 103
synapse_information	(spynnaker.pyNN.models.neural_projections.projection_machine_edges.ProjectionMachineEdges attribute), 104	synapse_information	(spynnaker.pyNN.models.neural_projections.projection_machine_edges.ProjectionMachineEdges attribute), 104
synapse_information	(spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdges attribute), 104	synapse_information	(spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdges attribute), 104

104		method), 182
SynapseIORowBased	(class in spyn- naker.pyNN.models.neuron.synapse_io), 202	synaptic_data_update () (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 195
SynapseIORowBased	(class in spyn- naker.pyNN.models.neuron.synapse_io.synapse_io_row_based), 201	SYNAPTIC_MATRIX (spyn- naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS. attribute), 283
SynapseRowTooBigException, 290		synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abs attribute), 152
SynapseStructureWeightAccumulator	(class in spyn- naker.pyNN.models.neuron.plasticity.stdp.synapse_structur 151	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Abs attribute), 156
SynapseStructureWeightAccumulator	(class in spyn- naker.pyNN.models.neuron.plasticity.stdp.synapse_structur 151	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 153
SynapseStructureWeightOnly	(class in spyn- naker.pyNN.models.neuron.plasticity.stdp.synapse_structur 151	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 153
SynapseStructureWeightOnly	(class in spyn- naker.pyNN.models.neuron.plasticity.stdp.synapse_structur 151	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 154
SynapseTypeAlpha	(class in spyn- naker.pyNN.models.neuron.synapse_types), 215	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 155
SynapseTypeAlpha	(class in spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_alpha), 203	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 155
SynapseTypeDelta	(class in spyn- naker.pyNN.models.neuron.synapse_types), 214	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tim attribute), 157
SynapseTypeDelta	(class in spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_delta), 204	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tim attribute), 158
SynapseTypeDualExponential	(class in spyn- naker.pyNN.models.neuron.synapse_types), 211	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tim attribute), 158
SynapseTypeDualExponential	(class in spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential), 206	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tim attribute), 156
SynapseTypeExponential	(class in spyn- naker.pyNN.models.neuron.synapse_types), 212	synaptic_structure (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Tim attribute), 159
SynapseTypeExponential	(class in spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_exponential), 207	SynapticBlockGenerationException, 290 SynapticConfigurationReadException, 290 SynapticConfigurationException, 290
SynapseTypeSEMD	(class in spyn- naker.pyNN.models.neuron.synapse_types), 217	SynapticManager (class in spyn- naker.pyNN.models.neuron), 241
SynapseTypeSEMD	(class in spyn- naker.pyNN.models.neuron.synapse_types.synapse_type_semd), 209	SynapticManager (class in spyn- naker.pyNN.models.neuron.synaptic_manager), 241
synaptic_data_update ()	(spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamic 290	SynapticMaxIncomingAtomsSupportException, 290
		synaptic_structural_common.SynapseDynamic (StructuralCommon

(in module spynnaker.spike_checker), 294
 synfire_spike_checker() (in module spyn-
 naker.spike_checker), 294
 SYSTEM (spynnaker.pyNN.utilities.constants.POPULATION_BASED_MEDIUM), 204
 attribute), 283
 SYSTEM_REGION (spyn-
 naker.pyNN.external_devices_models.munich_spinnaker_link_attribute), 43
 SYSTEM_REGION (spyn-
 naker.pyNN.external_devices_models.MunichMotorDevice attribute), 51
 SYSTEM_REGION (spyn-
 naker.pyNN.models.spike_source.spike_source_poisson_machine_attribute), 247
 T
 tau (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 155
 tau (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 159
 tau_ca2 (spynnaker.pyNN.models.neuron.additional_inputs_additional_inputs_synapse_types_synapse_type attribute), 110
 tau_ca2 (spynnaker.pyNN.models.neuron.additional_inputs_additional_inputs_synapse_types_synapse_type attribute), 111
 tau_m (spynnaker.pyNN.models.neuron.neuron_models.neuron_model.LeadSpinnakerpyNNmodels.neuron_models.neuron_models attribute), 145
 tau_m (spynnaker.pyNN.models.neuron.neuron_models.NeuronModel.LeadSpinnakerpyNNmodels.neuron_models.neuron_models attribute), 149
 tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 153
 tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 154
 tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 155
 tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 157
 tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 158
 tau_minus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 156
 tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 153
 tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 154
 tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 155
 tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 157
 tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 158
 tau_plus (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_timing_dependence_models_2011.TimingDependenceModels2011 attribute), 156
 tau_refrac (spynnaker.pyNN.models.neuron.neuron_models.neuron_model.LeadSpinnakerpyNNmodels.neuron_models.neuron_models attribute), 145

ThresholdTypeMaassStochastic (class in spyn- attribute), 14
 naker.pyNN.models.neuron.threshold_types), 222
 timed_commands (spyn-
 naker.pyNN.external_devices_models.push_bot.push_bot_etherne

ThresholdTypeMaassStochastic (class in spyn- attribute), 16
 naker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic), (spyn-
 219 naker.pyNN.external_devices_models.push_bot.push_bot_etherne

ThresholdTypeMulticastDeviceControl attribute), 21
 (class in spyn- timed_commands (spyn-
 naker.pyNN.external_devices_models), 53 naker.pyNN.external_devices_models.push_bot.push_bot_etherne

ThresholdTypeMulticastDeviceControl attribute), 21
 (class in spyn- timed_commands (spyn-
 naker.pyNN.external_devices_models.threshold_type_multicast_device_control), 46 naker.pyNN.external_devices_models.push_bot.push_bot_etherne

ThresholdTypeStatic (class in spyn- timed_commands (spyn-
 naker.pyNN.models.neuron.threshold_types), 221 naker.pyNN.external_devices_models.push_bot.push_bot_etherne

ThresholdTypeStatic (class in spyn- timing_dependence (spyn-
 naker.pyNN.models.neuron.threshold_types.threshold_type_static), 220 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics

time_between_send (spyn- timing_dependence (spyn-
 naker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device), 36 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics

time_between_send (spyn- TimingDependencePfisterSpikeTriplet
 naker.pyNN.external_devices_models.push_bot.AbstractPushBotOutputDevice in spyn-
 attribute), 36 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence), 157

time_scale_factor() (spyn- 157
 naker.pyNN.spynnaker_external_device_plugin_manager.SpyNNakerExternalDevicePluginManager
 static method), 293 (class in spyn-
 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim

timed_commands (spyn- naker.pyNN.external_devices_models.external_spinnaker_link2fpga_retina_device.ExternalFPGA retina_device
 attribute), 43 TimingDependenceRecurrent (class in spyn-
 naker.pyNN.external_devices_models.ExternalFPGA retina_device TimingDependenceRecurrent (class in spyn-
 attribute), 50 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim

timed_commands (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim
 naker.pyNN.external_devices_models.munich_spinnaker_link5_retina_device.MunichRetinaDevice
 attribute), 46 TimingDependenceSpikeNearestPair

timed_commands (spyn- (class in spyn-
 naker.pyNN.external_devices_models.MunichRetinaDevice naker.pyNN.models.neuron.plasticity.stdp.timing_dependence),
 attribute), 53 158

timed_commands (spyn- TimingDependenceSpikeNearestPair
 naker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice
 attribute), 36 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim

timed_commands (spyn- 154
 naker.pyNN.external_devices_models.push_bot.AbstractPushBotRetinaDevice SpikePair (class in spyn-
 attribute), 37 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence),
 156

timed_commands (spyn- 156
 naker.pyNN.external_devices_models.push_bot.push_bot_etherne_laser_device.PushBotEthernetLaserDe
 attribute), 13 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim

timed_commands (spyn- 154
 naker.pyNN.external_devices_models.push_bot.push_bot_etherne_led_device.PushBotEthernetLEDDevi
 attribute), 14 naker.pyNN.models.neuron.plasticity.stdp.timing_dependence),
 158

timed_commands (spyn- 158
 naker.pyNN.external_devices_models.push_bot.push_bot_etherne_motor_device.PushBotEthernetMotorL

`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_vogels_2011`, (spyn-
 155 `naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo`
`to_post_coords()` (spyn- `attribute`), 201
`naker.pyNN.models.neural_projections.connectors.KernelConnector` (spyn-
`method`), 84 `naker.pyNN.external_devices_models.external_spinnaker_link_fp`
`to_post_coords()` (spyn- `attribute`), 42
`naker.pyNN.models.neural_projections.connectors.KernelConnector` (spyn-
`method`), 101 `naker.pyNN.external_devices_models.ExternalFPGA`
`TRAFFIC_IDENTIFIER` (spyn- `attribute`), 50
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex` (spyn-
`attribute`), 224 `naker.pyNN.external_devices_models.munich_spinnaker_link_ret`
`TRAFFIC_IDENTIFIER` (spyn- `attribute`), 45
`naker.pyNN.models.neuron.AbstractPopulationVertex` (spyn-
`attribute`), 235 `naker.pyNN.external_devices_models.MunichRetinaDevice`
`translate_control_packet()` (spyn- `attribute`), 53
`naker.pyNN.external_devices_models.abstract_ethernet_translator.AbstractEthernetTranslator` (spyn-
`method`), 38 `naker.pyNN.spynaker_external_device_plugin_manager.SpynakerExternalDevicePluginManager`
`translate_control_packet()` (spyn- `static method`), 293
`naker.pyNN.external_devices_models.AbstractEthernetTranslator` (spyn-
`method`), 48 `naker.pyNN.models.neuron.master_pop_table_generators.abstract`
`translate_control_packet()` (spyn- `method`), 138
`naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_translator.PushBotEthernetTranslator` (spyn-
`method`), 17 `naker.pyNN.models.neuron.master_pop_table_generators.master`
`translate_control_packet()` (spyn- `method`), 140
`naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_translator.PushBotEthernetTranslator` (spyn-
`method`), 24 `naker.pyNN.models.neuron.master_pop_table_generators.Master`
`turn_off_sensor_reporting()` (spyn- `method`), 142
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (spyn-
`method`), 274 `naker.pyNN.external_devices_models.threshold_type_multicast_d`
`turn_off_sensor_reporting()` (spyn- `method`), 47
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (spyn-
`method`), 279 `naker.pyNN.external_devices_models.ThresholdTypeMulticastDev`
`turn_off_sensor_reporting_key` (spyn- `method`), 54
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (spyn-
`attribute`), 274 `naker.pyNN.models.neuron.additional_inputs.additional_input_co`
`turn_off_sensor_reporting_key` (spyn- `method`), 110
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (spyn-
`attribute`), 279 `naker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2`
 U `update_values()` (spyn-
`u_init` (spynnaker.pyNN.models.neuron.neuron_models.neuron_model.GEN_NeuronModel) (spyn-
`attribute`), 144 `naker.pyNN.models.neuron.implementations.abstract_standard_n`
`u_init` (spynnaker.pyNN.models.neuron.neuron_models.NeuronModel) (spyn-
`attribute`), 148 `naker.pyNN.models.neuron.implementations.AbstractStandardNeu`
`uart_id` (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol) (spyn-
`attribute`), 274 `update_values()` (spyn-
`uart_id` (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol) (spyn-
`attribute`), 279 `naker.pyNN.models.neuron.input_types.input_type_conductance.I`
`undelayed_max_bytes` (spyn- `update_values()` (spyn-
`naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo` (spyn-
`attribute`), 201 `naker.pyNN.models.neuron.input_types.input_type_current.Input1`
`undelayed_max_n_synapses` (spyn- `update_values()` (spyn-
`naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo` (spyn-
`attribute`), 201 `naker.pyNN.models.neuron.input_types.input_type_current_semd`

update_values() naker.pyNN.models.neuron.input_types.InputTypeConductance method), 135	(spyn- update_values() naker.pyNN.models.neuron.threshold_types.threshold_type_static method), 221	(spyn- update_values() naker.pyNN.models.neuron.threshold_types.threshold_type_static method), 221
update_values() naker.pyNN.models.neuron.input_types.InputTypeCurrent method), 136	(spyn- update_values() naker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass method), 223	(spyn- update_values() naker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass method), 223
update_values() naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD method), 137	(spyn- update_values() naker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic method), 222	(spyn- update_values() naker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic method), 222
update_values() naker.pyNN.models.neuron.neuron_models.neuron_model_izh method), 144	(spyn- update_weight() naker.pyNN.models.neuron_models.neuron_model_izh method), 58	(spyn- update_weight() naker.pyNN.models.neuron_models.neuron_model_izh method), 58
update_values() naker.pyNN.models.neuron.neuron_models.neuron_model_leak method), 146	(spyn- update_weight() naker.pyNN.models.neuron_models.neuron_model_leak method), 61	(spyn- update_weight() naker.pyNN.models.neuron_models.neuron_model_leak method), 61
update_values() naker.pyNN.models.neuron.neuron_models.NeuronModelIzh method), 148	(spyn- update_weight() naker.pyNN.models.neural_projections.delay_afferent_machine_e method), 102	(spyn- update_weight() naker.pyNN.models.neural_projections.delay_afferent_machine_e method), 102
update_values() naker.pyNN.models.neuron.neuron_models.NeuronModelLeak method), 149	(spyn- update_weight() naker.pyNN.models.neural_projections.DelayAfferentMachineEdge method), 105	(spyn- update_weight() naker.pyNN.models.neural_projections.DelayAfferentMachineEdge method), 105
update_values() naker.pyNN.models.neuron.synapse_types.synapse_type_alpha method), 204	(spyn- update_weight() naker.pyNN.models.neural_projections.projection_machine_edge method), 104	(spyn- update_weight() naker.pyNN.models.neural_projections.projection_machine_edge method), 104
update_values() naker.pyNN.models.neuron.synapse_types.synapse_type_delta method), 205	(spyn- update_weight() naker.pyNN.models.neural_projections.ProjectionMachineEdge method), 107	(spyn- update_weight() naker.pyNN.models.neural_projections.ProjectionMachineEdge method), 107
update_values() naker.pyNN.models.neuron.synapse_types.synapse_type_dual method), 207	(spyn- use_direct_matrix() naker.pyNN.models.synapse_types.DualExponentialConnectors method), 72	(spyn- use_direct_matrix() naker.pyNN.models.synapse_types.DualExponentialConnectors method), 72
update_values() naker.pyNN.models.neuron.synapse_types.synapse_type_exponential method), 209	(spyn- use_direct_matrix() naker.pyNN.models.synapse_types.ExponentialConnectors method), 87	(spyn- use_direct_matrix() naker.pyNN.models.synapse_types.ExponentialConnectors method), 87
update_values() naker.pyNN.models.neuron.synapse_types.synapse_type_semi method), 210	(spyn- use_direct_matrix() naker.pyNN.models.neural_projections.connectors.one_to_one_co method), 86	(spyn- use_direct_matrix() naker.pyNN.models.neural_projections.connectors.one_to_one_co method), 86
update_values() naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha method), 217	(spyn- use_direct_matrix() naker.pyNN.models.neural_projections.connectors.OneToOneCon method), 98	(spyn- use_direct_matrix() naker.pyNN.models.neural_projections.connectors.OneToOneCon method), 98
update_values() naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta method), 215	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh attribute), 144	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh attribute), 144
update_values() naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential method), 212	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_le attribute), 146	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_le attribute), 146
update_values() naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential method), 214	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 148	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh attribute), 148
update_values() naker.pyNN.models.neuron.synapse_types.SynapseTypeSEMD method), 218	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLe attribute), 146	(spyn- v_init (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLe attribute), 146
update_values() naker.pyNN.models.neuron.threshold_types.threshold_type_maass method), 220	(spyn- v_reset (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelI method), 149	(spyn- v_reset (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelI method), 149

v_rest (spynnaker.pyNN.models.neuron.neuron_models.NeuronModel.integrateAndFire attribute), 146

v_rest (spynnaker.pyNN.models.neuron.neuron_models.NeuronModel.integrateAndFire attribute), 149

v_thresh (spynnaker.pyNN.models.neuron.threshold_types.threshold_types.StaticThresholdTypeMuissStatePersistence attribute), 220

v_thresh (spynnaker.pyNN.models.neuron.threshold_types.threshold_types.StaticThresholdTypeStatic attribute), 221

v_thresh (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypesStochastic attribute), 223

v_thresh (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypesStochastic attribute), 222

validate_mars_kiss_64_seed() (in module vertex_executable_suffix (spyn-
naker.pyNN.utilities.utility_calls), 287 naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weights attribute), 161

var () (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats
method), 280 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weights attribute), 161

variance (spynnaker.pyNN.utilities.running_stats.RunningStats attribute), 285 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weights attribute), 162

verbose (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector attribute), 72 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weights attribute), 163

vertex_executable_suffix (spyn- vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 152 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 156 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 153 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 153 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 154 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 155 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 157 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 158 vertex_executable_suffix (spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.weights attribute), 158


```

naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 166
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 195
write_parameters() (spyn- write_setup_info() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 167
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 257
write_parameters() (spyn- write_setup_info() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 168
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 261
write_parameters() (spyn- write_structural_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 168
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 175
write_parameters() (spyn- write_structural_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 169
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 193
write_parameters() (spyn- write_structural_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 170
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 184
write_parameters() (spyn- write_structural_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 171
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 186
write_parameters() (spyn- write_structural_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 170
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 197
write_parameters() (spyn- write_structural_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 171
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination.RandomByWeightsElimination.SynapseDynamicsStatic
method), 199
write_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.random_selection.RandomSelection
method), 171
write_parameters() (spyn-
naker.pyNN.models.neuron.structural_plasticity.synaptogenesis.partner_selection.RandomSelection
method), 172
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics
method), 175
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics
method), 187
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic
method), 177
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP
method), 180
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon
method), 182
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic
method), 190
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP
method), 193
write_parameters() (spyn-

```