
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	252
1.1.1.3	spynnaker.pyNN.abstract_spinnaker_common module	252
1.1.1.4	spynnaker.pyNN.exceptions module	254
1.1.1.5	spynnaker.pyNN.spynnaker_external_device_plugin_manager module	255
1.1.1.6	spynnaker.pyNN.spynnaker_simulator_interface module	257
1.1.1.7	Module contents	258
1.2	Submodules	258
1.3	spynnaker.gsyn_tools module	258
1.4	spynnaker.plot_utils module	258
1.5	spynnaker.spike_checker module	258
1.6	Module contents	259
2	Indices and tables	261
	Python Module Index	263
	Index	269

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

spynnaker.pyNN.connections.ethernet_control_connection module

class spynnaker.pyNN.connections.ethernet_control_connection.**EthernetControlConnection** (*translation*
local_host
local_port

Bases: spinnman.connections.udp_packet_connections.eieio_connection.
EIEIOConnection

A connection that can translate Ethernet control messages received from a Population

Parameters

- **translator** – The translator of multicast to control commands
- **local_host** – The optional host to listen on
- **local_port** – The optional port to listen on

```
close()
    Close the connection

run()
```

spynnaker.pyNN.connections.spynnaker_live_spikes_connection module

```
class spynnaker.pyNN.connections.spynnaker_live_spikes_connection.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

Module contents

`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

`spynnaker.pyNN.external_devices_models.push_bot.push_bot_control_modules.push_bot_lif_spinnaker_link` module

Module contents

`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_`

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

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_ethernet_speaker_device module

spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_retina_connection module

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

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 bytestring

Raises

- **SpinnmanTimeoutException** – If a timeout occurs before any data is received
- **SpinnmanIOException** – If an error occurs receiving the data

remote_ip_address
The remote IP address to which the connection is connected.

Returns The remote IP address as a dotted string, or None if not connected remotely

Return type str

remote_port
The remote port to which the connection is connected.

Returns The remote port, or None if not connected remotely

Return type int

send (*data*)
Send data down this connection

Parameters `data` (*bytestring*) – The data to be sent

Raises `SpinnmanIOException` – If there is an error sending the data

`spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet.push_bot_wifi_connection`

Get an existing connection to a PushBot, or make a new one.

Parameters

- **remote_host** (*str*) – The IP address of the PushBot
- **remote_port** (*int*) – The port number of the PushBot (default 56000)

Module contents

`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.P

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_

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_v

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_de
module**

**spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_led_devi
module**

**spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_motor_d
module**

**spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_retina_d
module**

`spynnaker.pyNN.external_devices_models.push_bot.push_bot_spinnaker_link.push_bot_spinnaker_link_speaker`
module

Module contents

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.Abst
    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.Abst

    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_ne*

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.**ExternalDeviceLif**

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, sam-  
                                                                ple_time=4096,  
                                                                up-  
                                                                date_time=512,  
                                                                delay_time=5,  
                                                                delta_threshold=23,  
                                                                con-  
                                                                tinue_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*, *label=None*, *polarity=None*, *board_address=None*)

Bases: *pacman.model.graphs.application.application_spinnaker_link_vertex.ApplicationSpiNNakerLinkVertex*, *spinn_front_end_common.abstract_models.abstract_send_me_multicast_commands_vertex.AbstractSendMeMulticastCommandsVertex*, *spinn_front_end_common.abstract_models.abstract_provides_outgoing_partition_constraints*.

```
AbstractProvidesOutgoingPartitionConstraints,      spinn_front_end_common.  
abstract_models.impl.provides_key_to_atom_mapping_impl.  
ProvidesKeyToAtomMappingImpl
```

```
DOWN_POLARITY = 'DOWN'
```

```
LEFT_RETINA = 'LEFT'
```

```
LEFT_RETINA_DISABLE = 69
```

```
LEFT_RETINA_ENABLE = 69
```

```
LEFT_RETINA_KEY_SET = 67
```

```
MANAGEMENT_BIT = 1024
```

```
MANAGEMENT_MASK = 4294965248
```

```
MERGED_POLARITY = 'MERGED'
```

```
RIGHT_RETINA = 'RIGHT'
```

```
RIGHT_RETINA_DISABLE = 70
```

```
RIGHT_RETINA_ENABLE = 70
```

```
RIGHT_RETINA_KEY_SET = 68
```

```
UP_POLARITY = 'UP'
```

```
default_parameters = {'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

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

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_BYTES_PER_WORD = 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_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_step*)

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_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*)

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_BYTES_PER_WORD = 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_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(*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

get_delay_maximum(*delays*)

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(*delays, post_vertex_slice, 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()

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

get_provenance_data ()

get_weight_maximum (*weights*)
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.

post_population

pre_population

safe

set_projection_information (*pre_population, post_population, rng, machine_time_step*)

set_space (*space*)
Set the space object (allowed after instantiation).

Parameters *space* –

Returns

space

verbose

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

class spynnaker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine

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

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

get_weight_maximum (*weights*)
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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)
Create a synaptic block from the data.

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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()`

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

`get_weight_maximum(weights)`

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, 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(weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type)`

Create a synaptic block from the data.

`get_delay_maximum(delays)`

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

`get_n_connections_from_pre_vertex_maximum(delays, post_vertex_slice, 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()`

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

`get_weight_maximum(weights)`

Get the maximum of the weights for this connection.

`show_connection_set()`

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

d_expression

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

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

set_projection_information (*pre_population, post_population, rng, machine_time_step*)

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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

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*)

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 (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()

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

get_weight_maximum (*weights*)

Get the maximum of the weights for this connection.

set_projection_information (*pre_population, post_population, rng, machine_time_step*)

spynaker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector module

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

Bases: [*spynaker.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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

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*)

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 (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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()

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

get_weight_maximum(weights)

Get the maximum of the weights for this connection.

set_projection_information(pre_population, post_population, rng, machine_time_step)

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(weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type)
Create a synaptic block from the data.

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)

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(delays)

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

get_n_connections_from_pre_vertex_maximum (*delays*, *post_vertex_slice*,
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 ()

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

get_weight_maximum (*weights*)

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 (*weights*, *delays*, *pre_slices*, *pre_slice_index*, *post_slices*,
post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type)

Create a synaptic block from the data.

get_delay_maximum (*delays*)

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(*delays*, *post_vertex_slice*,
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()

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

get_weight_maximum(*weights*)

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(*weights*, *delays*, *pre_slices*, *pre_slice_index*, *post_slices*,
post_slice_index, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

Create a synaptic block from the data.

get_delay_maximum(*delays*)

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

get_n_connections_from_pre_vertex_maximum(*delays*, *post_vertex_slice*,
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()`

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

`get_weight_maximum(weights)`

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.ConvolutionKernel`

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 (`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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)
Create a synaptic block from the data.

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*)

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

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

get_delay_maximum (*delays*)

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 (*delays*, *post_vertex_slice*,
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 ()

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

get_weight_maximum (*weights*)

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.MultipseConnector`

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 (*weights, delays, pre_slices, pre_slice_index, post_slices,*
post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type)

Create a synaptic block from the data.

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)

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 (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays*, *post_vertex_slice*,
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 ()

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 (*weights*)

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 (*weights*, *delays*, *pre_slices*, *pre_slice_index*, *post_slices*,
post_slice_index, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

get_delay_maximum (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays*, *post_vertex_slice*,
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 ()

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

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

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

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

get_n_connections_from_pre_vertex_maximum(*delays, post_vertex_slice, 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()
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

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

set_projection_information(*pre_population, post_population, rng, machine_time_step*)

Module contents

class spynnaker.pyNN.models.neural_projections.connectors.**AbstractConnector**(*safe=True, 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(*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)
Create a synaptic block from the data.

get_delay_maximum(*delays*)
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 (*delays*, *post_vertex_slice*,
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 ()

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

get_provenance_data ()

get_weight_maximum (*weights*)

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.

post_population

pre_population

safe

set_projection_information (*pre_population*, *post_population*, *rng*, *machine_time_step*)

set_space (*space*)

Set the space object (allowed after instantiation).

Parameters *space* –

Returns

space

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*)

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,
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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

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*)

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 (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays*, *post_vertex_slice*,
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 ()

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

get_weight_maximum (*weights*)

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 (*weights*, *delays*, *pre_slices*, *pre_slice_index*, *post_slices*,
post_slice_index, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

Create a synaptic block from the data.

get_delay_maximum (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays*, *post_vertex_slice*,
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 ()

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

get_weight_maximum (*weights*)

Get the maximum of the weights for this connection.

class spynnaker.pyNN.models.neural_projections.connectors.**CSAConnector** (*cset*,
safe=True,
call-
back=None,
ver-
bose=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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)
Create a synaptic block from the data.

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

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

show_connection_set ()

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

d_expression

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()
Get the maximum number of connections between those to any neuron in the post vertex from neurons in the pre vertex.

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

set_projection_information (*pre_population, post_population, rng, machine_time_step*)

class `spynnaker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector` (*n, allow_self_connections, with_replacement, safe=True, verbose=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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)
Create a synaptic block from the data.

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*)

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 (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()

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

get_weight_maximum (*weights*)

Get the maximum of the weights for this connection.

set_projection_information (*pre_population, post_population, rng, machine_time_step*)

class spynnaker.pyNN.models.neural_projections.connectors.**FixedNumberPreConnector** (*n, allow_self_connections, with_replacement, safe=True, 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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

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*)

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 (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()

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

get_weight_maximum (*weights*)

Get the maximum of the weights for this connection.

set_projection_information (*pre_population, post_population, rng, machine_time_step*)

class spynnaker.pyNN.models.neural_projections.connectors.**FixedProbabilityConnector** (*p_connect, allow_self_connections, safe=True, verbose=False, rng=None*)

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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

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*)

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 (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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 ()

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

get_weight_maximum (*weights*)

Get the maximum of the weights for this connection.

class spynnaker.pyNN.models.neural_projections.connectors.**FromListConnector** (*conn_list, safe=True, 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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

get_delay_maximum (*delays*)

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 (*delays, post_vertex_slice, 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()

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

get_weight_maximum (*weights*)
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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)
Create a synaptic block from the data.

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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()

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

get_weight_maximum(weights)

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*, *safe*, *verbose*, *bose*, *rng*)

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 (*weights*, *delays*, *pre_slices*, *pre_slice_index*, *post_slices*, *post_slice_index*, *pre_vertex_slice*, *post_vertex_slice*, *synapse_type*)

Create a synaptic block from the data.

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*)

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(delays)

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

get_n_connections_from_pre_vertex_maximum (*delays*, *post_vertex_slice*, *min_delay*, *max_delay*)

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()

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 (*weights*)

Get the maximum of the weights for this connection.

class spynnaker.pyNN.models.neural_projections.connectors.**OneToOneConnector** (*random_number_class, safe=True, verbose=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 (*weights, delays, pre_slices, pre_slice_index, post_slices, post_slice_index, pre_vertex_slice, post_vertex_slice, synapse_type*)

Create a synaptic block from the data.

gen_connector_id

Get the id of the connection generator on the machine

Return type int

get_delay_maximum (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays, post_vertex_slice, 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()

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

get_weight_maximum (*weights*)

Get the maximum of the weights for this connection.

class spynnaker.pyNN.models.neural_projections.connectors.**SmallWorldConnector** (*degree, rewiring, allow_self_connections, safe=True, verbose=False, n_connections=None*)

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

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

Create a synaptic block from the data.

get_delay_maximum (*delays*)

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

get_n_connections_from_pre_vertex_maximum (*delays*, *post_vertex_slice*,
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 ()

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

get_weight_maximum (*weights*)

Get the maximum of the weights for this connection.

set_projection_information (*pre_population, post_population, rng, machine_time_step*)

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,
space,
verbose)

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 <=> (*startX, endX, _stepX_*) None or 2-item array
- **(optional)** – Starting row/col for pre/post sampling <=> (*_startX_, endX, stepX*) None or 2-item array

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

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

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*)
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
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

get_delay_maximum (*delays*)
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 (*delays, post_vertex_slice, 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()`

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

`get_weight_maximum(weights)`

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.**ProjectionApplic**

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.**ProjectionMachineEdg**

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, synapse_type, synapse_dynamics, weight, delay*)

Bases: `object`

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

connector

delay

synapse_dynamics

synapse_type

weight

Module contents

class `spynnaker.pyNN.models.neural_projections.DelayAfferentApplicationEdge` (*prevertex, postvertex, label, delay=0, weight=1.0*)

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 ma-
      chine 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,  
                                                                synapse_dynamics,  
                                                                synapse_type,  
                                                                weight=None,  
                                                                de-  
                                                                lay=None)
```

Bases: object

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

connector

delay

synapse_dynamics

synapse_type

weight

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.**AbstractAddi**

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_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.**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 (*spec, master_population_table_region*)

Initialise the master pop data structure

Parameters

- **spec** – the DSG writer
- **master_population_table_region** – the region in memory that the master pop table will be written in

Return type `None`

update_master_population_table (*spec, block_start_addr, row_length, key_and_mask, master_pop_table_region, is_single=False*)

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 (*spec, master_population_table_region*)

Initialise the master pop data structure

Parameters

- **spec** – the DSG writer
- **master_population_table_region** – the region in memory that the master pop table will be written in

Return type None

update_master_population_table (*spec, block_start_addr, row_length, key_and_mask, master_pop_table_region, is_single=False*)

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_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_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_dtcm_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_p)

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers.**get_lut_provenance**(*p,*
l,
n,
e,
t,
h,
h)

spynnaker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers.**write_exp_lut**(*spec,*
time_co,
size,
shift,
fixed_po)

Module contents

spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure package

Submodules

spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure
module

```
class spynnaker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structu
    Bases: object

    get_n_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_weig
    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_weig
    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.SynapseStructureWeight
    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.SynapseStructureWeight

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_pfi

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

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

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, 'n_weight_terms': 10, 'pre_trace_n_bytes': 10, 'synaptic_structure': 10, 'vertex_executable_suffix': ''}

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

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

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

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

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

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

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.**TimingDependencePfist**

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

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

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

class spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependenceRecur

```

```

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

```
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

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

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.TimingDependenceVogel
```

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.AbstractWeightDependence
    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.

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.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.**AbstractG**

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.**MatrixGen**

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.**Abst**

Bases: [spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics](#)

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`

get_delay_maximum (*connector*, *delays*)

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*, *weights*)
 Get the maximum weight for the synapses

get_weight_mean (*connector*, *weights*)
 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

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

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.structural_dynamics module

class spynnaker.pyNN.models.neuron.synapse_dynamics.structural_dynamics.**StructuralDynamics**
 Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.pynn_synapse_dynamics.PyNNSynapseDynamics*

structure

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

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.

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_value(*key, value*)

Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

write_parameters(*spec, region, machine_time_step, weight_scales*)

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

changes_during_run

Determine if the synapses change during a run

Return type bool

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, weights*)

Get the maximum weight for the synapses

get_weight_mean (*connector, weights*)

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.

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_value (*key, value*)

Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

timing_dependence

weight_dependence

write_parameters (*spec, region, machine_time_step, weight_scales*)

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: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural*

Common class that enables synaptic rewiring. It acts as a wrapper around SynapseDynamicsStatic or SynapseDynamicsSTDP. This means rewiring can operate in parallel with these types of synapses.

Written by Petrut Bogdan.

Parameters

- **f_rew** (*int*) – Frequency of rewiring (Hz). How many rewiring attempts will be done per second.
- **weight** (*float*) – Initial weight assigned to a newly formed connection
- **delay** (*int*) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **sigma_form_forward** (*float*) – Spread of feed-forward formation receptive field
- **sigma_form_lateral** (*float*) – Spread of lateral formation receptive field
- **p_form_forward** (*float*) – Peak probability for feed-forward formation
- **p_form_lateral** (*float*) – Peak probability for lateral formation
- **p_elim_pot** (*float*) – Probability of elimination of a potentiated synapse
- **p_elim_dep** (*float*) – Probability of elimination of a depressed synapse
- **grid** (*2d int array*) – Grid shape
- **lateral_inhibition** (*bool*) – Flag whether to mark synapses formed within a layer as inhibitory or excitatory
- **random_partner** (*bool*) – Flag whether to randomly select pre-synaptic partner for formation
- **seed** (*int*) – seed the random number generators

actual_sdram_usage

Actual SDRAM usage (based on what is written to spec).

Returns actual SDRAM usage

Return type int

default_parameters = {'delay': 1, 'f_rew': 10000, 'grid': array([16, 16]), 'lateral

distance (*x0, x1, grid=array([16, 16]), type='euclidian'*)

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
- **type** (*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_extra_sdram_usage_in_bytes (*machine_in_edges*)

Better approximation of SDRAM usage based on incoming machine edges

Parameters **machine_in_edges** (*machine edges*) – incoming machine edges

Returns SDRAM usage

Return type int

get_n_synapses_in_rows (*pp_size, fp_size=None*)

Get number of synapses in a row.

get_parameter_names ()

get_parameters_sdram_usage_in_bytes (*n_neurons, n_synapse_types, in_edges*)

Approximate SDRAM usage

Parameters

- **n_neurons** (*int*) – number of neurons
- **n_synapse_types** (*int*) – number of synapse types (i.e. excitatory and inhibitory)
- **in_edges** (*edges*) – incoming edges

Returns SDRAM usage

Return type int

get_vertex_executable_suffix ()

is_same_as (*synapse_dynamics*)

n_words_for_plastic_connections (*value*)

Get size of plastic connections in words

n_words_for_static_connections (*value*)

Get size of static connections in words

p_rew

The period of rewiring.

Returns The period of rewiring

Return type int

synaptic_data_update (*connections, post_vertex_slice, app_edge, machine_edge*)

Get static synaptic data

weight_dynamics

write_parameters (*spec, region, machine_time_step, weight_scales, application_graph, machine_graph, app_vertex, post_slice, machine_vertex, graph_mapper, routing_info*)

Write the synapse parameters to the spec.

Parameters

- **spec** (*spec*) – the data spec
- **region** (*int*) – memory region
- **machine_time_step** (*int*) – the duration of a machine time step (ms)
- **weight_scales** (*list(float)*) – scaling the weights
- **application_graph** (*ApplicationGraph*) – the entire, highest level, graph of the network to be simulated
- **machine_graph** (*MachineGraph*) – the entire, lowest level, graph of the network to be simulated
- **app_vertex** (*ApplicationVertex*) – the highest level object of the post-synaptic population
- **post_slice** (*Slice*) – the slice of the app vertex corresponding to this machine vertex
- **machine_vertex** (*MachineVertex*) – the lowest level object of the post-synaptic population
- **graph_mapper** (*GraphMapper*) – for looking up application vertices
- **routing_info** (*RoutingInfo*) – All of the routing information on the network

Returns None

Return type None

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.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural, spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic*

Class that enables synaptic rewiring. It acts as a wrapper around SynapseDynamicsStatic. This means rewiring can operate in parallel with these types of synapses.

Written by Petrut Bogdan.

Example usage to allow rewiring in parallel with STDP:

```
stdp_model = sim.STDPMechanism(...)

structure_model_with_stdp = sim.StructuralMechanismStatic(
    weight=0,
    s_max=32,
    grid=[np.sqrt(pop_size), np.sqrt(pop_size)],
    random_partner=True,
    f_rew=10 ** 4, # Hz
    sigma_form_forward=1.,
    delay=10
)
plastic_projection = sim.Projection(
    ...,
    synapse_dynamics=sim.SynapseDynamics(
        slow=structure_model_with_stdp
    )
)
```

Parameters

- **f_rew** (*int*) – Frequency of rewiring (Hz). How many rewiring attempts will be done per second.
- **weight** (*float*) – Initial weight assigned to a newly formed connection

- **delay** (*int*) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **sigma_form_forward** (*float*) – Spread of feed-forward formation receptive field
- **sigma_form_lateral** (*float*) – Spread of lateral formation receptive field
- **p_form_forward** (*float*) – Peak probability for feed-forward formation
- **p_form_lateral** (*float*) – Peak probability for lateral formation
- **p_elim_pot** (*float*) – Probability of elimination of a potentiated synapse
- **p_elim_dep** (*float*) – Probability of elimination of a depressed synapse
- **grid** (*2d int array*) – Grid shape
- **lateral_inhibition** (*bool*) – Flag whether to mark synapses formed within a layer as inhibitory or excitatory
- **random_partner** (*bool*) – Flag whether to randomly select pre-synaptic partner for formation
- **seed** (*int*) – seed the random number generators

changes_during_run

Determine if the synapses change during a run

Return type bool

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, in_edges*)

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, app_edge, machine_edge*)

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_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

write_parameters (*spec, region, machine_time_step, weight_scales, application_graph, machine_graph, app_vertex, post_slice, machine_vertex, graph_mapper, routing_info*)

Write the synapse parameters to the spec

spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp module**class** spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp.SynapseDynamicsSTDP

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural, spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP*

Class that enables synaptic rewiring. It acts as a wrapper around SynapseDynamicsSTDP. This means rewiring can operate in parallel with these types of synapses.

Written by Petrut Bogdan.

Example usage to allow rewiring in parallel with STDP:

```
stdp_model = sim.STDPMechanism(...)

structure_model_with_stdp = sim.StructuralMechanismSTDP(
    stdp_model=stdp_model,
    weight=0,
    s_max=32,
    grid=[np.sqrt(pop_size), np.sqrt(pop_size)],
    random_partner=True,
    f_rew=10 ** 4, # Hz
    sigma_form_forward=1.,
    delay=10
)
plastic_projection = sim.Projection(
    ...,
    synapse_dynamics=sim.SynapseDynamics(
        slow=structure_model_with_stdp
    )
)
```

Parameters

- **f_rew** (*int*) – Frequency of rewiring (Hz). How many rewiring attempts will be done per second.
- **weight** (*float*) – Initial weight assigned to a newly formed connection

- **delay** (*int*) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **sigma_form_forward** (*float*) – Spread of feed-forward formation receptive field
- **sigma_form_lateral** (*float*) – Spread of lateral formation receptive field
- **p_form_forward** (*float*) – Peak probability for feed-forward formation
- **p_form_lateral** (*float*) – Peak probability for lateral formation
- **p_elim_pot** (*float*) – Probability of elimination of a potentiated synapse
- **p_elim_dep** (*float*) – Probability of elimination of a depressed synapse
- **grid** (*2d int array*) – Grid shape
- **lateral_inhibition** (*bool*) – Flag whether to mark synapses formed within a layer as inhibitory or excitatory
- **random_partner** (*bool*) – Flag whether to randomly select pre-synaptic partner for formation
- **seed** (*int*) – seed the random number generators

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, in_edges*)

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, app_edge, machine_edge*)

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_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

write_parameters (*spec, region, machine_time_step, weight_scales, application_graph, machine_graph, app_vertex, post_slice, machine_vertex, graph_mapper, routing_info*)

Write the synapse parameters to the spec

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`

get_delay_maximum(*connector, delays*)

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, weights*)

Get the maximum weight for the synapses

get_weight_mean(*connector, weights*)

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

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** (*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

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.

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_value (*key*, *value*)

Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign

write_parameters (*spec*, *region*, *machine_time_step*, *weight_scales*)

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, pad_to_length=None*)

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

changes_during_run
Determine if the synapses change during a run
Return type bool

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, weights*)

Get the maximum weight for the synapses

get_weight_mean (*connector, weights*)

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.

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_value (*key, value*)

Set a property

Parameters

- **key** – the name of the parameter to change
- **value** – the new value of the parameter to assign Set a property
- **key** – the name of the parameter to change

- **value** – the new value of the parameter to assign

timing_dependence

weight_dependence

write_parameters (*spec, region, machine_time_step, weight_scales*)

Write the synapse parameters to the spec

class spynnaker.pyNN.models.neuron.synapse_dynamics.**AbstractSynapseDynamicsStructural**

Bases: object

class spynnaker.pyNN.models.neuron.synapse_dynamics.**StructuralDynamics** (*slow=None, fast=None, structure=None*)

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.pyNN_synapse_dynamics.PyNNSynapseDynamics*

structure

class spynnaker.pyNN.models.neuron.synapse_dynamics.**SynapseDynamicsStructuralCommon** (*stdp_mod*

f_rew=10

weight=0

de-

lay=1,

s_max=3

sigma_for

sigma_for

p_form_f

p_form_l

p_elim_d

p_elim_p

grid=arra

16]),

lat-

eral_inhi

ran-

dom_part

seed=Non

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural*

Common class that enables synaptic rewiring. It acts as a wrapper around SynapseDynamicsStatic or SynapseDynamicsSTDP. This means rewiring can operate in parallel with these types of synapses.

Written by Petrut Bogdan.

Parameters

- **f_rew** (*int*) – Frequency of rewiring (Hz). How many rewiring attempts will be done per second.
- **weight** (*float*) – Initial weight assigned to a newly formed connection
- **delay** (*int*) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **sigma_form_forward** (*float*) – Spread of feed-forward formation receptive field
- **sigma_form_lateral** (*float*) – Spread of lateral formation receptive field

- **p_form_forward** (*float*) – Peak probability for feed-forward formation
- **p_form_lateral** (*float*) – Peak probability for lateral formation
- **p_elim_pot** (*float*) – Probability of elimination of a potentiated synapse
- **p_elim_dep** (*float*) – Probability of elimination of a depressed synapse
- **grid** (*2d int array*) – Grid shape
- **lateral_inhibition** (*bool*) – Flag whether to mark synapses formed within a layer as inhibitory or excitatory
- **random_partner** (*bool*) – Flag whether to randomly select pre-synaptic partner for formation
- **seed** (*int*) – seed the random number generators

actual_sdram_usage

Actual SDRAM usage (based on what is written to spec).

Returns actual SDRAM usage

Return type int

default_parameters = {'delay': 1, 'f_rew': 10000, 'grid': array([16, 16]), 'lateral.

distance (*x0, x1, grid=array([16, 16]), type='euclidian'*)

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
- **type** (*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_extra_sdram_usage_in_bytes (*machine_in_edges*)

Better approximation of SDRAM usage based on incoming machine edges

Parameters **machine_in_edges** (*machine edges*) – incoming machine edges

Returns SDRAM usage

Return type int

get_n_synapses_in_rows (*pp_size, fp_size=None*)

Get number of synapses in a row.

get_parameter_names ()

get_parameters_sdram_usage_in_bytes (*n_neurons*, *n_synapse_types*, *in_edges*)

Approximate SDRAM usage

Parameters

- **n_neurons** (*int*) – number of neurons
- **n_synapse_types** (*int*) – number of synapse types (i.e. excitatory and inhibitory)
- **in_edges** (*edges*) – incoming edges

Returns SDRAM usage

Return type int

get_vertex_executable_suffix ()

is_same_as (*synapse_dynamics*)

n_words_for_plastic_connections (*value*)

Get size of plastic connections in words

n_words_for_static_connections (*value*)

Get size of static connections in words

p_rew

The period of rewiring.

Returns The period of rewiring

Return type int

synaptic_data_update (*connections*, *post_vertex_slice*, *app_edge*, *machine_edge*)

Get static synaptic data

weight_dynamics

write_parameters (*spec*, *region*, *machine_time_step*, *weight_scales*, *application_graph*, *machine_graph*, *app_vertex*, *post_slice*, *machine_vertex*, *graph_mapper*, *routing_info*)

Write the synapse parameters to the spec.

Parameters

- **spec** (*spec*) – the data spec
- **region** (*int*) – memory region
- **machine_time_step** (*int*) – the duration of a machine time step (ms)
- **weight_scales** (*list(float)*) – scaling the weights
- **application_graph** (*ApplicationGraph*) – the entire, highest level, graph of the network to be simulated
- **machine_graph** (*MachineGraph*) – the entire, lowest level, graph of the network to be simulated
- **app_vertex** (*ApplicationVertex*) – the highest level object of the post-synaptic population
- **post_slice** (*Slice*) – the slice of the app vertex corresponding to this machine vertex
- **machine_vertex** (*MachineVertex*) – the lowest level object of the post-synaptic population

- **graph_mapper** (GraphMapper) – for looking up application vertices
- **routing_info** (RoutingInfo) – All of the routing information on the network

Returns None

Return type None

class spynnaker.pyNN.models.neuron.synapse_dynamics.**SynapseDynamicsStructuralStatic** (*stdp_model*, *f_rew=10*, *weight=0*, *de-*, *lay=1*, *s_max=3*, *sigma_for*, *sigma_for*, *p_form_f*, *p_form_l*, *p_elim_d*, *p_elim_p*, *grid=array*, *16*]), *lat-*, *eral_inhi*, *ran-*, *dom_part*, *seed=None*)

Bases: `spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural`, `spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic`

Class that enables synaptic rewiring. It acts as a wrapper around SynapseDynamicsStatic. This means rewiring can operate in parallel with these types of synapses.

Written by Petrut Bogdan.

Example usage to allow rewiring in parallel with STDP:

```
stdp_model = sim.STDPMechanism(...)

structure_model_with_stdp = sim.StructuralMechanismStatic(
    weight=0,
    s_max=32,
    grid=[np.sqrt(pop_size), np.sqrt(pop_size)],
    random_partner=True,
    f_rew=10 ** 4, # Hz
    sigma_form_forward=1.,
    delay=10
)
plastic_projection = sim.Projection(
    ...,
    synapse_dynamics=sim.SynapseDynamics(
        slow=structure_model_with_stdp
    )
)
```

Parameters

- **f_rew** (*int*) – Frequency of rewiring (Hz). How many rewiring attempts will be done per second.
- **weight** (*float*) – Initial weight assigned to a newly formed connection
- **delay** (*int*) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **sigma_form_forward** (*float*) – Spread of feed-forward formation receptive field
- **sigma_form_lateral** (*float*) – Spread of lateral formation receptive field
- **p_form_forward** (*float*) – Peak probability for feed-forward formation
- **p_form_lateral** (*float*) – Peak probability for lateral formation
- **p_elim_pot** (*float*) – Probability of elimination of a potentiated synapse
- **p_elim_dep** (*float*) – Probability of elimination of a depressed synapse
- **grid** (*2d int array*) – Grid shape
- **lateral_inhibition** (*bool*) – Flag whether to mark synapses formed within a layer as inhibitory or excitatory
- **random_partner** (*bool*) – Flag whether to randomly select pre-synaptic partner for formation
- **seed** (*int*) – seed the random number generators

changes_during_run

Determine if the synapses change during a run

Return type bool

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, in_edges*)

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, app_edge, machine_edge*)

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_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

write_parameters (*spec, region, machine_time_step, weight_scales, application_graph, machine_graph, app_vertex, post_slice, machine_vertex, graph_mapper, routing_info*)

Write the synapse parameters to the spec

class spynnaker.pyNN.models.neuron.synapse_dynamics.**SynapseDynamicsStructuralSTDP** (*stdp_model=*
f_rew=1000,
weight=0,
de-
lay=1,
s_max=32,
sigma_form_
sigma_form_
p_form_forw
p_form_late
p_elim_dep=
p_elim_pot=
grid=array([
16]),
lat-
eral_inhibiti
ran-
dom_partner
seed=None)

Bases: *spynnaker.pyNN.models.neuron.synapse_dynamics.*
abstract_synapse_dynamics_structural.AbstractSynapseDynamicsStructural,
spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.
SynapseDynamicsSTDP

Class that enables synaptic rewiring. It acts as a wrapper around `SynapseDynamicsSTDP`. This means rewiring can operate in parallel with these types of synapses.

Written by Petrut Bogdan.

Example usage to allow rewiring in parallel with STDP:

```
stdp_model = sim.STDPMechanism(...)

structure_model_with_stdp = sim.StructuralMechanismSTDP(
    stdp_model=stdp_model,
    weight=0,
    s_max=32,
    grid=[np.sqrt(pop_size), np.sqrt(pop_size)],
    random_partner=True,
    f_rew=10 ** 4, # Hz
    sigma_form_forward=1.,
    delay=10
)

plastic_projection = sim.Projection(
    ...,
    synapse_dynamics=sim.SynapseDynamics(
        slow=structure_model_with_stdp
    )
)
```

Parameters

- **f_rew** (*int*) – Frequency of rewiring (Hz). How many rewiring attempts will be done per

second.

- **weight** (*float*) – Initial weight assigned to a newly formed connection
- **delay** (*int*) – Delay assigned to a newly formed connection
- **s_max** (*int*) – Maximum fan-in per target layer neuron
- **sigma_form_forward** (*float*) – Spread of feed-forward formation receptive field
- **sigma_form_lateral** (*float*) – Spread of lateral formation receptive field
- **p_form_forward** (*float*) – Peak probability for feed-forward formation
- **p_form_lateral** (*float*) – Peak probability for lateral formation
- **p_elim_pot** (*float*) – Probability of elimination of a potentiated synapse
- **p_elim_dep** (*float*) – Probability of elimination of a depressed synapse
- **grid** (*2d int array*) – Grid shape
- **lateral_inhibition** (*bool*) – Flag whether to mark synapses formed within a layer as inhibitory or excitatory
- **random_partner** (*bool*) – Flag whether to randomly select pre-synaptic partner for formation
- **seed** (*int*) – seed the random number generators

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, in_edges*)

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, app_edge, machine_edge*)

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_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

write_parameters (*spec, region, machine_time_step, weight_scales, application_graph, machine_graph, app_vertex, post_slice, machine_vertex, graph_mapper, routing_info*)

Write the synapse parameters to the spec

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_stages, 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_stages, 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_response*, *exc_synapse*, *tau_synapse*, *inh_response*, *inh_synapse*, *tau_synapse*)

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

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

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.ThresholdTypeMaassStochastic`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_neurons*, *la-*
bel, *con-*
strain, *max_*
spike, *ring_*
in-
com-
ing_s
neu-
ron_l
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.AbstractProvidesOutgoingPartitionConstraints,
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_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 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_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 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

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': 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

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.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`
 An enumeration.

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*)

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_dtcn_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, in_edges, machine_time_step*)

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 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

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*)

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, in_edges, machine_time_step)

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

    An enumeration.

    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,
                                                                    neu-
                                                                    ron_model,
                                                                    in-
                                                                    put_type,
                                                                    synapse_type,
                                                                    thresh-
                                                                    old_type,
                                                                    addi-
                                                                    tional_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=[])
```

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

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 = 4

PROVENANCE_REGION = 3

SPIKE_HISTORY_REGION = 2

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 (*graph*)

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_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

convert_rate (*rate*)

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 ()

Returns a human-readable description of the cell or synapse type.

The output may be customised by specifying a different template together with an associated template engine (see `pyNN.descriptions`).

If template is None, then a dictionary containing the template context will be returned.

duration

generate_data_specification (*spec, placement, machine_time_step, time_scale_factor, graph_mapper, routing_info, data_n_time_steps, graph*)

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_dtcn_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*)

static get_params_bytes (*vertex_slice*)

Gets the size of the poisson parameters in bytes

Parameters *vertex_slice* –

get_recording_sdram_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

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*)

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()

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

Module contents

class spynnaker.pyNN.models.spike_source.**SpikeSourceArray** (*spike_times=[]*)

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='r')

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

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(n_delay_stages,
                                                                    de-
                                                                    lay_per_stage,
                                                                    ver-
                                                                    tex_slice)
```

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.DelayExtensionVert
```

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_dtcm_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.delays.delay_generator_data module

class spynnaker.pyNN.models.utility_models.delays.delay_generator_data.**DelayGeneratorData** (*...*)

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, delay_per_stage, vertex_slice*)

Bases: object

A block of delays for a vertex.

add_delay (*source_id, stage*)

delay_block

class spynnaker.pyNN.models.utility_models.delays.**DelayExtensionMachineVertex** (*resources_required, label, constraints=None*)

Bases: pacman.model.graphs.machine.machine_vertex.MachineVertex, spinn_front_end_common.interface.provenance.provides_provenance_data_from_machine_impl.ProvidesProvenanceDataFromMachineImpl

class **EXTRA_PROVENANCE_DATA_ENTRIES**

Bases: enum.Enum

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_dtcm_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.AbstractProvidesOutgoingPartitionConstraints
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** (*app*

gra
pla
men
tran
pro
nan
ex-
e-
cute

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)

all ()

Iterator over cell IDs on all nodes.

can_record (*variable*)

Determine whether *variable* can be recorded from this population.

Note: This is supported by sPyNNaker8

conductance_based

True if the population uses conductance inputs

first_id

get (*parameter_names*, *gather=False*)

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

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.

id_to_index (*id*)

Given the ID(s) of cell(s) in the Population, return its (their) index (order in the Population).

id_to_local_index (*cell_id*)

Given the ID(s) of cell(s) in the Population, return its (their) index (order in the Population), counting only cells on the local MPI node.

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.

label

The label of the population

last_id**local_size**

The number of local cells

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,
                                                                    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

static **pynn7_format** (*data, ids, sampling_interval, data2=None*)

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.**GraphEdgeWeightUpdater**

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.**SpynnakerDataSpecificationWriter**

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.MunichIoSpiNNakerLinkProto

```

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

An enumeration.

BALL_BALANCER = 3

FREE = 5

MY_ORO_BOTICS = 4

PUSH_BOT = 1

RESET_TO_DEFAULT = 0

SPOMNIBOT = 2

add_payload_logic_to_current_output (payload, time=None)

add_payload_logic_to_current_output_key

bias_values (bias_id, bias_value, time=None)

bias_values_key

configure_master_key (new_key, time=None)

configure_master_key_key

disable_retina (time=None)

disable_retina_key

enable_disable_motor_key

generic_motor0_raw_output_leak_to_0 (pwm_signal, time=None)

generic_motor0_raw_output_leak_to_0_key

generic_motor0_raw_output_permanent (pwm_signal, time=None)

`generic_motor0_raw_output_permanent_key`
`generic_motor1_raw_output_leak_to_0` (*pwm_signal, time=None*)
`generic_motor1_raw_output_leak_to_0_key`
`generic_motor1_raw_output_permanent` (*pwm_signal, time=None*)
`generic_motor1_raw_output_permanent_key`
`generic_motor_disable` (*time=None*)
`generic_motor_enable` (*time=None*)
`generic_motor_total_period` (*time_in_ms, uart_id=0, time=None*)
`generic_motor_total_period_key`
`instance_key`
 The key of this instance of the protocol
`master_slave_key`
`master_slave_set_master_clock_active` (*time=None*)
`master_slave_set_master_clock_not_started` (*time=None*)
`master_slave_set_slave` (*time=None*)
`master_slave_use_internal_counter` (*time=None*)
`mode`
`poll_individual_sensor_continuously` (*sensor_id, time_in_ms, time=None*)
`poll_individual_sensor_continuously_key`
`poll_sensors_once` (*sensor_id, time=None*)
`poll_sensors_once_key`
`protocol_instance = 0`
`push_bot_laser_config_active_time` (*active_time, time=None*)
`push_bot_laser_config_active_time_key`
`push_bot_laser_config_total_period` (*total_period, time=None*)
`push_bot_laser_config_total_period_key`
`push_bot_laser_set_frequency` (*frequency, time=None*)
`push_bot_laser_set_frequency_key`
`push_bot_led_back_active_time` (*active_time, time=None*)
`push_bot_led_back_active_time_key`
`push_bot_led_front_active_time` (*active_time, time=None*)
`push_bot_led_front_active_time_key`
`push_bot_led_set_frequency` (*frequency, time=None*)
`push_bot_led_set_frequency_key`
`push_bot_led_total_period` (*total_period, time=None*)
`push_bot_led_total_period_key`


```

push_bot_motor_0_leaking_towards_zero(velocity, time=None)
push_bot_motor_0_leaking_towards_zero_key
push_bot_motor_0_permanent(velocity, time=None)
push_bot_motor_0_permanent_key
push_bot_motor_1_leaking_towards_zero(velocity, time=None)
push_bot_motor_1_leaking_towards_zero_key
push_bot_motor_1_permanent(velocity, time=None)
push_bot_motor_1_permanent_key
push_bot_speaker_config_active_time(active_time, time=None)
push_bot_speaker_config_active_time_key
push_bot_speaker_config_total_period(total_period, time=None)
push_bot_speaker_config_total_period_key
push_bot_speaker_set_melody(melody, time=None)
push_bot_speaker_set_melody_key
push_bot_speaker_set_tone(frequency, time=None)
push_bot_speaker_set_tone_key
pwm_pin_output_timer_a_channel_0_ratio(timer_period, time=None)
pwm_pin_output_timer_a_channel_0_ratio_key
pwm_pin_output_timer_a_channel_1_ratio(timer_period, time=None)
pwm_pin_output_timer_a_channel_1_ratio_key
pwm_pin_output_timer_a_duration(timer_period, time=None)
pwm_pin_output_timer_a_duration_key
pwm_pin_output_timer_b_channel_0_ratio(timer_period, time=None)
pwm_pin_output_timer_b_channel_0_ratio_key()
pwm_pin_output_timer_b_channel_1_ratio(timer_period, time=None)
pwm_pin_output_timer_b_channel_1_ratio_key
pwm_pin_output_timer_b_duration(timer_period, time=None)
pwm_pin_output_timer_b_duration_key
pwm_pin_output_timer_c_channel_0_ratio(timer_period, time=None)
pwm_pin_output_timer_c_channel_0_ratio_key
pwm_pin_output_timer_c_channel_1_ratio(timer_period, time=None)
pwm_pin_output_timer_c_channel_1_ratio_key()
pwm_pin_output_timer_c_duration(timer_period, time=None)
pwm_pin_output_timer_c_duration_key
query_state_of_io_lines(time=None)
query_state_of_io_lines_key

```

```
remove_payload_logic_to_current_output (payload, time=None)
remove_payload_logic_to_current_output_key
reset_retina (time=None)
reset_retina_key
sensor_transmission_key (sensor_id)
static sent_mode_command()
    True if the mode command has ever been requested by any instance
set_mode (time=None)
set_mode_key
set_output_pattern_for_payload (payload, time=None)
set_output_pattern_for_payload_key
set_payload_pins_to_high_impedance (payload, time=None)
set_payload_pins_to_high_impedance_key
set_retina_key (new_key, time=None)
set_retina_key_key
set_retina_transmission (retina_key=<RetinaKey.NATIVE_128_X_128: 67108864>,
                        retina_payload=None, time=None)
    Set the retina transmission key
```

Parameters

- **retina_key** – the new key for the retina
- **retina_payload** (*enum or None*) – the new payload for the set retina key command packet
- **time** – when to transmit this packet

Returns the command to send

Return type `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_
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

An enumeration.

BALL_BALANCER = 3

FREE = 5

MY_ORO_BOTICS = 4

PUSH_BOT = 1

RESET_TO_DEFAULT = 0

SPOMNIBOT = 2

add_payload_logic_to_current_output (payload, time=None)

add_payload_logic_to_current_output_key

bias_values (bias_id, bias_value, time=None)

bias_values_key

configure_master_key (new_key, time=None)

configure_master_key_key

disable_retina (time=None)

disable_retina_key

```

enable_disable_motor_key
generic_motor0_raw_output_leak_to_0 (pwm_signal, time=None)
generic_motor0_raw_output_leak_to_0_key
generic_motor0_raw_output_permanent (pwm_signal, time=None)
generic_motor0_raw_output_permanent_key
generic_motor1_raw_output_leak_to_0 (pwm_signal, time=None)
generic_motor1_raw_output_leak_to_0_key
generic_motor1_raw_output_permanent (pwm_signal, time=None)
generic_motor1_raw_output_permanent_key
generic_motor_disable (time=None)
generic_motor_enable (time=None)
generic_motor_total_period (time_in_ms, uart_id=0, time=None)
generic_motor_total_period_key
instance_key
    The key of this instance of the protocol
master_slave_key
master_slave_set_master_clock_active (time=None)
master_slave_set_master_clock_not_started (time=None)
master_slave_set_slave (time=None)
master_slave_use_internal_counter (time=None)
mode
poll_individual_sensor_continuously (sensor_id, time_in_ms, time=None)
poll_individual_sensor_continuously_key
poll_sensors_once (sensor_id, time=None)
poll_sensors_once_key
protocol_instance = 0
push_bot_laser_config_active_time (active_time, time=None)
push_bot_laser_config_active_time_key
push_bot_laser_config_total_period (total_period, time=None)
push_bot_laser_config_total_period_key
push_bot_laser_set_frequency (frequency, time=None)
push_bot_laser_set_frequency_key
push_bot_led_back_active_time (active_time, time=None)
push_bot_led_back_active_time_key
push_bot_led_front_active_time (active_time, time=None)
push_bot_led_front_active_time_key

```

```
push_bot_led_set_frequency(frequency, time=None)
push_bot_led_set_frequency_key
push_bot_led_total_period(total_period, time=None)
push_bot_led_total_period_key
push_bot_motor_0_leaking_towards_zero(velocity, time=None)
push_bot_motor_0_leaking_towards_zero_key
push_bot_motor_0_permanent(velocity, time=None)
push_bot_motor_0_permanent_key
push_bot_motor_1_leaking_towards_zero(velocity, time=None)
push_bot_motor_1_leaking_towards_zero_key
push_bot_motor_1_permanent(velocity, time=None)
push_bot_motor_1_permanent_key
push_bot_speaker_config_active_time(active_time, time=None)
push_bot_speaker_config_active_time_key
push_bot_speaker_config_total_period(total_period, time=None)
push_bot_speaker_config_total_period_key
push_bot_speaker_set_melody(melody, time=None)
push_bot_speaker_set_melody_key
push_bot_speaker_set_tone(frequency, time=None)
push_bot_speaker_set_tone_key
pwm_pin_output_timer_a_channel_0_ratio(timer_period, time=None)
pwm_pin_output_timer_a_channel_0_ratio_key
pwm_pin_output_timer_a_channel_1_ratio(timer_period, time=None)
pwm_pin_output_timer_a_channel_1_ratio_key
pwm_pin_output_timer_a_duration(timer_period, time=None)
pwm_pin_output_timer_a_duration_key
pwm_pin_output_timer_b_channel_0_ratio(timer_period, time=None)
pwm_pin_output_timer_b_channel_0_ratio_key()
pwm_pin_output_timer_b_channel_1_ratio(timer_period, time=None)
pwm_pin_output_timer_b_channel_1_ratio_key
pwm_pin_output_timer_b_duration(timer_period, time=None)
pwm_pin_output_timer_b_duration_key
pwm_pin_output_timer_c_channel_0_ratio(timer_period, time=None)
pwm_pin_output_timer_c_channel_0_ratio_key
pwm_pin_output_timer_c_channel_1_ratio(timer_period, time=None)
pwm_pin_output_timer_c_channel_1_ratio_key()
```

```

pwm_pin_output_timer_c_duration (timer_period, time=None)
pwm_pin_output_timer_c_duration_key
query_state_of_io_lines (time=None)
query_state_of_io_lines_key
remove_payload_logic_to_current_output (payload, time=None)
remove_payload_logic_to_current_output_key
reset_retina (time=None)
reset_retina_key
sensor_transmission_key (sensor_id)
static sent_mode_command()
    True if the mode command has ever been requested by any instance
set_mode (time=None)
set_mode_key
set_output_pattern_for_payload (payload, time=None)
set_output_pattern_for_payload_key
set_payload_pins_to_high_impedance (payload, time=None)
set_payload_pins_to_high_impedance_key
set_retina_key (new_key, time=None)
set_retina_key_key
set_retina_transmission (retina_key=<RetinaKey.NATIVE_128_X_128:
                        retina_payload=None, time=None)      67108864>,
    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 `spynaker.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*, *defaults=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 of 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

    An enumeration.

    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.**FakeHBPPortalMachineProvider**

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, extra=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='\\')
```

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,
                                                                           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_to_add*)

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**time_scale_factor**

The multiplicative scaling from application time to real execution time.

Returns the time scale factor**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

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 an 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*, *port*, *hostname*, *tag=None*, *board_address=None*, *strip_sdp=True*, *use_prefix=False*, *key_prefix=None*, *prefix_type=None*, *message_type=<EIEIOTType.KEY_32_BIT: 2>*, *right_shift=0*, *payload_as_time_stamps=True*, *use_payload_prefix=True*, *payload_prefix=None*, *payload_right_shift=0*, *number_of_packets_sent_per_time_step=0*, *partition_id=None*)

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_gsyn (path, n_neurons, runtime, gsyn)  
spynnaker.gsyn_tools.check_sister_gsyn (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

263

35	60
spynnaker.pyNN.models.abstract_models.abstract_neuron_model,	spynnaker.pyNN.models.neural_projections.connectors,
35	61
spynnaker.pyNN.models.abstract_models.abstract_neuron_model,	spynnaker.pyNN.models.neural_projections.connectors,
36	63
spynnaker.pyNN.models.abstract_models.abstract_neuron_model,	spynnaker.pyNN.models.neural_projections.connectors,
36	64
spynnaker.pyNN.models.abstract_models.abstract_neuron_model,	spynnaker.pyNN.models.neural_projections.connectors,
37	65
spynnaker.pyNN.models.abstract_models.abstract_neuron_model,	spynnaker.pyNN.models.neural_projections.delay_aff,
37	79
spynnaker.pyNN.models.abstract_pynn_models,	spynnaker.pyNN.models.neural_projections.delay_aff,
229	79
spynnaker.pyNN.models.common,	spynnaker.pyNN.models.neural_projections.delayed_ap,
spynnaker.pyNN.models.common.abstract_neuron_recorder,	80
40	spynnaker.pyNN.models.neural_projections.delayed_ma,
spynnaker.pyNN.models.common.abstract_spike_recorder,	80
41	spynnaker.pyNN.models.neural_projections.projection,
spynnaker.pyNN.models.common.eieio_spike_recorder,	81
42	spynnaker.pyNN.models.neural_projections.projection,
spynnaker.pyNN.models.common.multi_spike_recorder,	81
42	spynnaker.pyNN.models.neural_projections.synapse_in,
spynnaker.pyNN.models.common.neuron_recorder,	82
42	spynnaker.pyNN.models.neural_properties,
spynnaker.pyNN.models.common.recording_utils,	85
44	spynnaker.pyNN.models.neural_properties.neural_para,
spynnaker.pyNN.models.common.simple_population,	85
45	spynnaker.pyNN.models.neuron,
spynnaker.pyNN.models.defaults,	230
spynnaker.pyNN.models.neural_projections,	190
82	spynnaker.pyNN.models.neuron.abstract_pynn_neuron_r,
spynnaker.pyNN.models.neural_projections.connectors,	196
65	spynnaker.pyNN.models.neuron.abstract_pynn_neuron_r,
spynnaker.pyNN.models.neural_projections.connectors,	196
50	spynnaker.pyNN.models.neuron.abstract_connector,
spynnaker.pyNN.models.neural_projections.connectors,	197
51	spynnaker.pyNN.models.neuron.additional_inputs,
spynnaker.pyNN.models.neural_projections.connectors,	197
52	spynnaker.pyNN.models.neuron.additional_inputs.abst,
spynnaker.pyNN.models.neural_projections.connectors,	197
53	spynnaker.pyNN.models.neuron.additional_inputs.add,
spynnaker.pyNN.models.neural_projections.connectors,	197
54	spynnaker.pyNN.models.neuron.builds,
spynnaker.pyNN.models.neural_projections.connectors,	197
55	spynnaker.pyNN.models.neuron.builds.eif_cond_alpha,
spynnaker.pyNN.models.neural_projections.connectors,	197
56	spynnaker.pyNN.models.neuron.builds.eif_cond_alpha,
spynnaker.pyNN.models.neural_projections.connectors,	197
57	spynnaker.pyNN.models.neuron.builds.eif_cond_alpha,
spynnaker.pyNN.models.neural_projections.connectors,	197
58	spynnaker.pyNN.models.neuron.builds.eif_cond_alpha,
spynnaker.pyNN.models.neural_projections.connectors,	197
59	spynnaker.pyNN.models.neuron.builds.eif_cond_alpha,
spynnaker.pyNN.models.neural_projections.connectors,	197

90	122
spynnaker.pyNN.models.neuron.builds.if_cspynnaker.pyNN.models.neuron.plasticity,	
90	142
spynnaker.pyNN.models.neuron.builds.if_cspynnaker.pyNN.models.neuron.plasticity.stdp,	
90	142
spynnaker.pyNN.models.neuron.builds.if_cspynnaker.pyNN.models.neuron.plasticity.stdp.common	
90	127
spynnaker.pyNN.models.neuron.builds.if_cspynnaker.pyNN.models.neuron.plasticity.stdp.common	
90	127
spynnaker.pyNN.models.neuron.builds.if_fspynnaker.pyNN.models.neuron.plasticity.stdp.synaps	
91	128
spynnaker.pyNN.models.neuron.builds.izk_cspynnaker.pyNN.models.neuron.plasticity.stdp.synaps	
91	128
spynnaker.pyNN.models.neuron.builds.izk_cspynnaker.pyNN.models.neuron.plasticity.stdp.synaps	
91	128
spynnaker.pyNN.models.neuron.connection_hspynnaker.pyNN.models.neuron.plasticity.stdp.synaps	
197	128
spynnaker.pyNN.models.neuron.generator_dspynnaker.pyNN.models.neuron.plasticity.stdp.timing	
198	133
spynnaker.pyNN.models.neuron.implementatspynnaker.pyNN.models.neuron.plasticity.stdp.timing	
100	129
spynnaker.pyNN.models.neuron.implementatspynnaker.pyNN.models.neuron.plasticity.stdp.timing	
93	130
spynnaker.pyNN.models.neuron.implementatspynnaker.pyNN.models.neuron.plasticity.stdp.timing	
95	131
spynnaker.pyNN.models.neuron.implementatspynnaker.pyNN.models.neuron.plasticity.stdp.timing	
97	131
spynnaker.pyNN.models.neuron.implementatspynnaker.pyNN.models.neuron.plasticity.stdp.timing	
99	132
spynnaker.pyNN.models.neuron.implementatspynnaker.pyNN.models.neuron.plasticity.stdp.timing	
99	133
spynnaker.pyNN.models.neuron.input_tspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
111	140
spynnaker.pyNN.models.neuron.input_tspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
107	137
spynnaker.pyNN.models.neuron.input_tspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
107	137
spynnaker.pyNN.models.neuron.input_tspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
109	138
spynnaker.pyNN.models.neuron.input_tspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
110	139
spynnaker.pyNN.models.neuron.master_pop_sspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
117	139
spynnaker.pyNN.models.neuron.master_pop_sspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
115	198
spynnaker.pyNN.models.neuron.master_pop_sspynnaker.pyNN.models.neuron.plasticity.stdp.weight	
116	157
spynnaker.pyNN.models.neuron.neuron_modelspynnaker.pyNN.models.neuron.synapse_dynamics.abstr	
123	142
spynnaker.pyNN.models.neuron.neuron_modelspynnaker.pyNN.models.neuron.synapse_dynamics.abstr	
119	143
spynnaker.pyNN.models.neuron.neuron_modelspynnaker.pyNN.models.neuron.synapse_dynamics.abstr	
120	144
spynnaker.pyNN.models.neuron.neuron_modelspynnaker.pyNN.models.neuron.synapse_dynamics.abstr	

[144](#) spynnaker.pyNN.models.spike_source.spike_source_ar
 spynnaker.pyNN.models.neuron.synapse_dynamics.[219](#) abstract_synapse_dynamics_structural,
[145](#) spynnaker.pyNN.models.spike_source.spike_source_ar
 spynnaker.pyNN.models.neuron.synapse_dynamics.[211](#) in_synapse_dynamics,
[145](#) spynnaker.pyNN.models.spike_source.spike_source_fr
 spynnaker.pyNN.models.neuron.synapse_dynamics.[212](#) structural_dynamics,
[145](#) spynnaker.pyNN.models.spike_source.spike_source_po
 spynnaker.pyNN.models.neuron.synapse_dynamics.[212a](#) synapse_dynamics_static,
[146](#) spynnaker.pyNN.models.spike_source.spike_source_po
 spynnaker.pyNN.models.neuron.synapse_dynamics.[213](#) synapse_dynamics_stdp,
[147](#) spynnaker.pyNN.models.spike_source.spike_source_po
 spynnaker.pyNN.models.neuron.synapse_dynamics.[214](#) synapse_dynamics_structural_common,
[150](#) spynnaker.pyNN.models.utility_models,
 spynnaker.pyNN.models.neuron.synapse_dynamics.[229a](#) synapse_dynamics_structural_static,
[153](#) spynnaker.pyNN.models.utility_models.delays,
 spynnaker.pyNN.models.neuron.synapse_dynamics.[223](#) synapse_dynamics_structural_stdp,
[155](#) spynnaker.pyNN.models.utility_models.delays.delay_b
 spynnaker.pyNN.models.neuron.synapse_io, [219](#)
[171](#) spynnaker.pyNN.models.utility_models.delays.delay_e
 spynnaker.pyNN.models.neuron.synapse_io.abstract[219](#) synapse_io,
[170](#) spynnaker.pyNN.models.utility_models.delays.delay_e
 spynnaker.pyNN.models.neuron.synapse_io.max_row[220](#) info,
[170](#) spynnaker.pyNN.models.utility_models.delays.delay_e
 spynnaker.pyNN.models.neuron.synapse_io.synapse[222](#) o_row_based,
[171](#) spynnaker.pyNN.models.utility_models.spike_injector
 spynnaker.pyNN.models.neuron.synapse_types, [228](#)
[179](#) spynnaker.pyNN.models.utility_models.spike_injector
 spynnaker.pyNN.models.neuron.synapse_types.abst[226](#) ct_synapse_type,
[172](#) spynnaker.pyNN.models.utility_models.spike_injector
 spynnaker.pyNN.models.neuron.synapse_types.synap[226e](#) se_type_alpha,
[173](#) spynnaker.pyNN.models.utility_models.synapse_expan
 spynnaker.pyNN.models.neuron.synapse_types.synap[229e](#) se_type_delta,
[174](#) spynnaker.pyNN.models.utility_models.synapse_expan
 spynnaker.pyNN.models.neuron.synapse_types.synap[228e](#) se_type_dual_exponential,
[176](#) spynnaker.pyNN.overridden_pacman_functions,
 spynnaker.pyNN.models.neuron.synapse_types.synap[234e](#) se_type_exponential,
[177](#) spynnaker.pyNN.overridden_pacman_functions.graph_e
 spynnaker.pyNN.models.neuron.synaptic_manager, [233](#)
[200](#) spynnaker.pyNN.overridden_pacman_functions.graph_e
 spynnaker.pyNN.models.neuron.threshold_types, [234](#)
[187](#) spynnaker.pyNN.overridden_pacman_functions.spynnake
 spynnaker.pyNN.models.neuron.threshold_types.abst[234](#) ract_threshold_type,
[185](#) spynnaker.pyNN.protocols, [239](#)
 spynnaker.pyNN.models.neuron.threshold_types.spynnake[239](#) s.pyNN.type_masks.stochastic_ethernet_protocol
[185](#) spynnaker.pyNN, [234](#)
 spynnaker.pyNN.models.neuron.threshold_types.spynnake[239](#) s.pyNN.type_masks
[186](#) spynnaker.pyNN, [235](#)
 spynnaker.pyNN.models.pyNN_population_compy, [255](#)
[230](#) spynnaker.pyNN.spynnaker_external_device_plugin_mar
 spynnaker.pyNN.models.pyNN_projection_compy, [257](#)
[233](#) spynnaker.pyNN.spynnaker_simulator_interface,
 spynnaker.pyNN.models.recording_common, spynnaker.pyNN.utilities, [252](#)
[233](#) spynnaker.pyNN.utilities.constants, [248](#)
 spynnaker.pyNN.models.spike_source, [218](#) spynnaker.pyNN.utilities.extracted_data,

248
spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider,
249
spynnaker.pyNN.utilities.random_stats,
245
spynnaker.pyNN.utilities.random_stats.abstract_random_stats,
244
spynnaker.pyNN.utilities.ranged, 246
spynnaker.pyNN.utilities.ranged.spynnaker_ranged_dict,
245
spynnaker.pyNN.utilities.ranged.spynnaker_ranged_list,
246
spynnaker.pyNN.utilities.reports, 249
spynnaker.pyNN.utilities.running_stats,
249
spynnaker.pyNN.utilities.spynnaker_connection_holder_generations,
249
spynnaker.pyNN.utilities.spynnaker_failed_state,
249
spynnaker.pyNN.utilities.spynnaker_neuron_network_specification_report,
250
spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report,
250
spynnaker.pyNN.utilities.utility_calls,
250
spynnaker.spike_checker, 258

A

- `a` (`spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh_neuron_model` attribute), 120
- `a` (`spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh` attribute), 124
- `A3_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceAdditiveTriplet` attribute), 139
- `A3_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet` attribute), 142
- `A3_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceAdditiveTriplet` attribute), 139
- `A3_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.WeightDependenceAdditiveTriplet` attribute), 142
- `A_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus` attribute), 137
- `A_minus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractHasAPlusAMinus` attribute), 140
- `A_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus.AbstractHasAPlusAMinus` attribute), 137
- `A_plus` (`spynnaker.pyNN.models.neuron.plasticity.stdp.weight_dependence.AbstractHasAPlusAMinus` attribute), 140
- `ABSOLUTE_2_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload` attribute), 239
- `ABSOLUTE_2_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.RetinaPayload` attribute), 244
- `ABSOLUTE_3_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload` attribute), 239
- `ABSOLUTE_3_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.RetinaPayload` attribute), 244
- `ABSOLUTE_4_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaPayload` attribute), 239
- `ABSOLUTE_4_BYTE_TIMESTAMPS` (`spynnaker.pyNN.protocols.RetinaPayload` attribute), 244
- `AbstractAcceptsIncomingSynapses` (class in `spynnaker.pyNN.models.abstract_models`), 37
- `AbstractAcceptsIncomingSynapses` (class in `spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses`), 34
- `AbstractAdditionalInput` (class in `spynnaker.pyNN.models.neuron.additional_inputs`), 87
- `AbstractAdditionalInput` (class in `spynnaker.pyNN.models.neuron.additional_inputs.abstract_additional_input`), 86
- `AbstractConnector` (class in `spynnaker.pyNN.models.neural_projections.connectors`), 65
- `AbstractConnector` (class in `spynnaker.pyNN.models.neural_projections.connectors.abstract_connector`), 59
- `AbstractContainsUnits` (class in `spynnaker.pyNN.models.abstract_models`), 38
- `AbstractContainsUnits` (class in `spynnaker.pyNN.models.abstract_models.abstract_contains_units`), 24
- `AbstractEthernetController` (class in `spynnaker.pyNN.external_devices_models`), 26
- `AbstractEthernetController` (class in `spynnaker.pyNN.external_devices_models.abstract_ethernet_controller`), 16
- `AbstractEthernetSensor` (class in `spynnaker.pyNN.external_devices_models`), 26
- `AbstractEthernetSensor` (class in `spynnaker.pyNN.external_devices_models.abstract_ethernet_sensor`), 17
- `AbstractEthernetTranslator` (class in `spynnaker.pyNN.external_devices_models`), 27
- `AbstractEthernetTranslator` (class in `spynnaker.pyNN.external_devices_models.abstract_ethernet_translator`), 17
- `AbstractFilterableEdge` (class in `spynnaker.pyNN.models.abstract_models`), 38

AbstractFilterableEdge (class in spyn- naker.pyNN.models.abstract_models.abstract_filterable_edge), 35	AbstractNeuronRecordable (class in spyn- naker.pyNN.models.common.abstract_neuron_recordable), 40
AbstractGenerateConnectorOnMachine (class in spyn- naker.pyNN.models.neural_projections.connectors), 66	AbstractPlasticSynapseDynamics (class in spyn- naker.pyNN.models.neuron.synapse_dynamics), 158
AbstractGenerateConnectorOnMachine (class in spyn- naker.pyNN.models.neural_projections.connectors.abstract_generate_connector_on_machine), 51	AbstractPlasticSynapseDynamics (class in spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_plastic_synapse_dynamics), 143
AbstractGenerateOnMachine (class in spyn- naker.pyNN.models.neuron.synapse_dynamics), 158	AbstractPopulationInitializable (class in spynnaker.pyNN.models.abstract_models), 38
AbstractGenerateOnMachine (class in spyn- naker.pyNN.models.neuron.synapse_dynamics.abstract_generate_on_machine), 142	AbstractPopulationInitializable (class in spyn- naker.pyNN.models.abstract_models.abstract_population_initializable), 35
AbstractHasAPlusAMinus (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 140	AbstractPopulationSettable (class in spyn- naker.pyNN.models.abstract_models), 39
AbstractHasAPlusAMinus (class in spyn- naker.pyNN.models.neuron.plasticity.stdp.weight_dependence), 137	AbstractPopulationSettable (class in spyn- naker.pyNN.models.abstract_models.abstract_population_settable), 36
AbstractInputType (class in spyn- naker.pyNN.models.neuron.input_types), 111	AbstractPopulationVertex (class in spyn- naker.pyNN.models.neuron), 201
AbstractInputType (class in spyn- naker.pyNN.models.neuron.input_types.abstract_input_type), 107	AbstractPopulationVertex (class in spyn- naker.pyNN.models.neuron.abstract_population_vertex), 190
AbstractMasterPopTableFactory (class in spyn- naker.pyNN.models.neuron.master_pop_table_generators), 115	AbstractPushBotOutputDevice (class in spyn- naker.pyNN.external_devices_models.push_bot), 15
AbstractMulticastControllableDevice (class in spyn- naker.pyNN.external_devices_models), 27	AbstractPushBotOutputDevice (class in spyn- naker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device), 15
AbstractMulticastControllableDevice (class in spyn- naker.pyNN.external_devices_models.abstract_multicast_controllable_device), 17	AbstractPushBotRetinaDevice (class in spyn- naker.pyNN.external_devices_models.push_bot), 16
AbstractNeuronImpl (class in spyn- naker.pyNN.models.neuron.implementations), 105	AbstractPushBotRetinaDevice (class in spyn- naker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device), 17
AbstractNeuronImpl (class in spyn- naker.pyNN.models.neuron.implementations.abstract_neuron_impl), 93	AbstractPyNNModel (class in spyn- naker.pyNN.models.abstract_pynn_model), 229
AbstractNeuronModel (class in spyn- naker.pyNN.models.neuron.neuron_models), 123	AbstractPyNNNeuronModel (class in spyn- naker.pyNN.models.neuron), 209
AbstractNeuronModel (class in spyn- naker.pyNN.models.neuron.neuron_models.abstract_neuron_model), 119	AbstractPyNNNeuronModel (class in spyn- naker.pyNN.models.neuron.abstract_pynn_neuron_model), 196
AbstractNeuronRecordable (class in spyn- naker.pyNN.models.common), 45	AbstractPyNNNeuronModelStandard (class in spynnaker.pyNN.models.neuron), 210
	AbstractPyNNNeuronModelStandard (class in spyn- naker.pyNN.models.neuron.abstract_pynn_neuron_model_standard), 196
	AbstractRandomStats (class in spyn-

<code>naker.pyNN.utilities.random_stats</code>), 245	<code>naker.pyNN.models.neuron.synapse_io</code>),
<code>AbstractRandomStats</code> (class in <code>spyn-</code>	171
<code>naker.pyNN.utilities.random_stats.abstract_random_stats</code>),	<code>AbstractSynapseIO</code> (class in <code>spyn-</code>
244	<code>naker.pyNN.models.neuron.synapse_io.abstract_synapse_io</code>),
<code>AbstractReadParametersBeforeSet</code> (class in	170
<code>spynnaker.pyNN.models.abstract_models</code>), 39	<code>AbstractSynapseStructure</code> (class in <code>spyn-</code>
<code>AbstractReadParametersBeforeSet</code>	<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure</code>),
(class in <code>spyn-</code>	128
<code>naker.pyNN.models.abstract_models.abstract_read_parameters_before_set</code>),	<code>AbstractSynapseStructure</code> (class in <code>spyn-</code>
36	<code>naker.pyNN.models.neuron.plasticity.stdp.synapse_structure.abstract_synapse_structure</code>),
<code>AbstractSettable</code> (class in <code>spyn-</code>	128
<code>naker.pyNN.models.abstract_models</code>), 39	<code>AbstractSynapseType</code> (class in <code>spyn-</code>
<code>AbstractSettable</code> (class in <code>spyn-</code>	<code>naker.pyNN.models.neuron.synapse_types</code>),
<code>naker.pyNN.models.abstract_models.abstract_settable</code>),	179
37	<code>AbstractSynapseType</code> (class in <code>spyn-</code>
<code>AbstractSpikeRecordable</code> (class in <code>spyn-</code>	<code>naker.pyNN.models.neuron.synapse_types.abstract_synapse_type</code>),
<code>naker.pyNN.models.common</code>), 46	172
<code>AbstractSpikeRecordable</code> (class in <code>spyn-</code>	<code>AbstractThresholdType</code> (class in <code>spyn-</code>
<code>naker.pyNN.models.common.abstract_spike_recordable</code>),	<code>naker.pyNN.models.neuron.threshold_types</code>),
41	187
<code>AbstractSpiNNakerCommon</code> (class in <code>spyn-</code>	<code>AbstractThresholdType</code> (class in <code>spyn-</code>
<code>naker.pyNN.abstract_spinnaker_common</code>),	<code>naker.pyNN.models.neuron.threshold_types.abstract_threshold_type</code>),
252	185
<code>AbstractStandardNeuronComponent</code>	<code>AbstractTimingDependence</code> (class in <code>spyn-</code>
(class in <code>spyn-</code>	<code>naker.pyNN.models.neuron.plasticity.stdp.timing_dependence</code>),
<code>naker.pyNN.models.neuron.implementations</code>),	133
100	<code>AbstractTimingDependence</code> (class in <code>spyn-</code>
<code>AbstractStandardNeuronComponent</code>	<code>naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_timing_dependence</code>),
(class in <code>spyn-</code>	129
<code>naker.pyNN.models.neuron.implementations.abstract_standard_neuron_component</code>),	<code>AbstractWeightDependence</code> (class in <code>spyn-</code>
95	<code>naker.pyNN.models.neuron.plasticity.stdp.weight_dependence</code>),
<code>AbstractStaticSynapseDynamics</code>	140
(class in <code>spyn-</code>	<code>AbstractWeightDependence</code> (class in <code>spyn-</code>
<code>naker.pyNN.models.neuron.synapse_dynamics</code>),	<code>naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence</code>),
158	137
<code>AbstractStaticSynapseDynamics</code>	<code>AbstractWeightUpdatable</code> (class in <code>spyn-</code>
(class in <code>spyn-</code>	<code>naker.pyNN.models.abstract_models</code>), 40
<code>naker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics</code>),	<code>AbstractWeightUpdatable</code> (class in <code>spyn-</code>
144	<code>naker.pyNN.models.abstract_models.abstract_weight_updatable</code>),
<code>AbstractSynapseDynamics</code> (class in <code>spyn-</code>	37
<code>naker.pyNN.models.neuron.synapse_dynamics</code>),	<code>activate_live_output_for()</code> (<code>spyn-</code>
157	<code>naker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager</code>
<code>AbstractSynapseDynamics</code> (class in <code>spyn-</code>	static method), 255
<code>naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics</code>),	<code>activate_synapse_output_to()</code> (<code>spyn-</code>
144	<code>naker.pyNN.spynnaker_external_device_plugin_manager.SpynnakerExternalDevicePluginManager</code>
<code>AbstractSynapseDynamicsStructural</code>	static method), 256
(class in <code>spyn-</code>	<code>actual_sdram_usage</code> (<code>spyn-</code>
<code>naker.pyNN.models.neuron.synapse_dynamics</code>),	<code>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural</code>
163	attribute), 151
<code>AbstractSynapseDynamicsStructural</code>	<code>actual_sdram_usage</code> (<code>spyn-</code>
(class in <code>spyn-</code>	<code>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructural</code>
<code>naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics_structural</code>),	attribute), 151
145	<code>add_application_vertex()</code> (<code>spyn-</code>
<code>AbstractSynapseIO</code> (class in <code>spyn-</code>	<code>naker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon</code>

`naker.pyNN.models.neuron.input_types.input_type_current` `naker.pyNN.models.neuron.input_types.input_type_current` `naker.pyNN.models.neuron.threshold_types.ThresholdTypeMaass`
`method), 110` `method), 189`
`add_state_variables()` `(spyn- add_state_variables()` `(spyn-`
`naker.pyNN.models.neuron.input_types.InputTypeConductance` `naker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic`
`method), 111` `method), 188`
`add_state_variables()` `(spyn- add_synapse_information()` `(spyn-`
`naker.pyNN.models.neuron.input_types.InputTypeCurrent` `naker.pyNN.models.neural_projections.delayed_application_edges`
`method), 112` `method), 80`
`add_state_variables()` `(spyn- add_synapse_information()` `(spyn-`
`naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD` `naker.pyNN.models.neural_projections.DelayedApplicationEdge`
`method), 114` `method), 83`
`add_state_variables()` `(spyn- add_synapse_information()` `(spyn-`
`naker.pyNN.models.neuron.neuron_models.neuron_model_input_type` `naker.pyNN.models.neural_projections.projection_application_edges`
`method), 121` `method), 81`
`add_state_variables()` `(spyn- add_synapse_information()` `(spyn-`
`naker.pyNN.models.neuron.neuron_models.neuron_model_input_type` `naker.pyNN.models.neural_projections.projection_application_edges`
`method), 122` `method), 84`
`add_state_variables()` `(spyn- AdditionalInputCa2Adaptive (class in spyn-`
`naker.pyNN.models.neuron.neuron_models.NeuronModelLz` `naker.pyNN.models.neuron.additional_inputs,`
`method), 124` `87`
`add_state_variables()` `(spyn- AdditionalInputCa2Adaptive (class in spyn-`
`naker.pyNN.models.neuron.neuron_models.NeuronModelLz` `naker.pyNN.models.neuron.additional_inputs.additional_input_ca`
`method), 126` `86`
`add_state_variables()` `(spyn- ADDRESS_LIST_DTYPE` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.synapse_type_alpha` `naker.pyNN.models.neuron.master_pop_table_generators.master`
`method), 173` `attribute), 116`
`add_state_variables()` `(spyn- ADDRESS_LIST_DTYPE` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.synapse_type_delta` `naker.pyNN.models.neuron.master_pop_table_generators.Master`
`method), 174` `attribute), 117`
`add_state_variables()` `(spyn- ADDRESS_MASK` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.synapse_type_dual` `naker.pyNN.models.neuron.master_pop_table_generators.master`
`method), 176` `attribute), 116`
`add_state_variables()` `(spyn- ADDRESS_MASK` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.synapse_type_exponential` `naker.pyNN.models.neuron.master_pop_table_generators.Master`
`method), 177` `attribute), 117`
`add_state_variables()` `(spyn- ADDRESS_SCALE` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha` `naker.pyNN.models.neuron.master_pop_table_generators.master`
`method), 184` `attribute), 116`
`add_state_variables()` `(spyn- ADDRESS_SCALE` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta` `naker.pyNN.models.neuron.master_pop_table_generators.Master`
`method), 182` `attribute), 118`
`add_state_variables()` `(spyn- ADDRESS_SCALED_SHIFT` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDual` `naker.pyNN.models.neuron.master_pop_table_generators.master`
`method), 179` `attribute), 116`
`add_state_variables()` `(spyn- ADDRESS_SCALED_SHIFT` `(spyn-`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential` `naker.pyNN.models.neuron.master_pop_table_generators.Master`
`method), 181` `attribute), 118`
`add_state_variables()` `(spyn- all()` `(spynnaker.pyNN.models.pynn_population_common.PyNNPopulation`
`naker.pyNN.models.neuron.threshold_types.threshold_type_maass` `method), 185` `method), 185`
`method), 185` `ALL_TO_ALL_CONNECTOR` `(spyn-`
`add_state_variables()` `(spyn- naker.pyNN.models.neural_projections.connectors.abstract_generators`
`naker.pyNN.models.neuron.threshold_types.threshold_type_maass` `method), 186` `method), 186`
`method), 186` `allow_self_connections` `(spyn-`
`add_state_variables()` `(spyn- naker.pyNN.models.neural_projections.connectors.all_to_all_com`

attribute), 52
 allow_self_connections (spyn- are_weights_signed() (spyn-
 naker.pyNN.models.neural_projections.connectors.AllToAllConnector), 67
 allow_self_connections (spyn- are_weights_signed() (spyn-
 naker.pyNN.models.neural_projections.connectors.distance_dependent_synapse_distance_dependent_probabilistic_connector), 55
 allow_self_connections (spyn- ArrayConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors.DistanceDependentProbabilisticConnector), 69
 allow_self_connections (spyn- ArrayConnector (class in spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connectors.array_connectors), 56
 allow_self_connections (spyn- as_list() (spynnaker.pyNN.utilities.ranged.spynnaker_ranged_list.SpynnakerRangedList), 57
 allow_self_connections (spyn- as_list() (spynnaker.pyNN.utilities.ranged.SpynnakerRangedList), 247
 allow_self_connections (spyn- static method), 247
 naker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector
 attribute), 70
B
 allow_self_connections (spyn- b (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh), 71
 naker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector
 attribute), 71
 allow_self_connections (spyn- b (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh), 124
 naker.pyNN.models.neural_projections.connectors.index_based_probability_connector.IndexBasedProbabilityConnector
 attribute), 60
 allow_self_connections (spyn- b (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh), 235
 naker.pyNN.models.neural_projections.connectors.IndexBasedProbabilityConnector
 attribute), 74
 AllToAllConnector (class in spyn- b (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh), 240
 naker.pyNN.models.neural_projections.connectors), 67
 AllToAllConnector (class in spyn- BASE_SIZE (spynnaker.pyNN.models.neuron.generator_data.GeneratorData), 198
 naker.pyNN.models.neural_projections.connectors.all_to_all_connector), 52
 alpha (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011.TimingDependenceVogels2011), 133
 alpha (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011.TimingDependenceVogels2011), 190
 alpha (spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_vogels_2011.TimingDependenceVogels2011), 136
 ArbitraryFPGADevice (class in spyn- b (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh), 201
 naker.pyNN.external_devices_models), 27
 ArbitraryFPGADevice (class in spyn- bias_values() (spyn-
 naker.pyNN.external_devices_models.arbitrary_fpga_device), 18
 naker.pyNN.models.neuron.neuron_models.NeuronModelIzh), 235
 are_weights_signed() (spyn- bias_values() (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics), 144
 method), 144
 are_weights_signed() (spyn- bias_values_key (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics), 157
 method), 157
 are_weights_signed() (spyn- bias_values_key (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic), 146
 method), 146
 are_weights_signed() (spyn- binary_name (spyn-
 naker.pyNN.models.neuron.implementations.abstract_neuron_impl.AbstractNeuronImpl), 155
 method), 155

binary_name (spyn- attribute), 154
 naker.pyNN.models.neuron.implementations.AbstractNeuronImpl
 attribute), 105
 binary_name (spyn- attribute), 159
 naker.pyNN.models.neuron.implementations.neuron_impl_standard.NeuronImplStandard (spyn-
 attribute), 97
 naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics
 binary_name (spyn- attribute), 161
 naker.pyNN.models.neuron.implementations.NeuronImplStandard (spyn-
 attribute), 103
 naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics
 bits_per_coordinate (spyn- attribute), 167
 naker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey
 attribute), 239
 RetinaKey.exists_and_create_if_not ()
 (in module spyn-
 bits_per_coordinate (spyn- naker.pyNN.utilities.utility_calls), 250
 naker.pyNN.protocols.RetinaKey attribute), check_gsyn () (in module spynnaker.gsyn_tools), 258
 244 check_indexes () (spyn-
 BUFFER_OVERFLOW_COUNT (spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder
 naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES
 attribute), 199 check_indexes () (spyn-
 BUFFER_OVERFLOW_COUNT (spyn- naker.pyNN.models.common.NeuronRecorder
 naker.pyNN.models.neuron.PopulationMachineVertex.EXTRA_PROVENANCE_DATA_ENTRIES
 attribute), 208 check_path_gsyn () (in module spyn-
 BYTES_TILL_START_OF_GLOBAL_PARAMETERS naker.gsyn_tools), 258
 (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex) (in module spyn-
 attribute), 190 naker.pyNN.utilities.utility_calls), 250
 BYTES_TILL_START_OF_GLOBAL_PARAMETERS check_sister_gsyn () (in module spyn-
 (spynnaker.pyNN.models.neuron.AbstractPopulationVertex naker.gsyn_tools), 258
 attribute), 201 clear_connection_cache () (spyn-
 naker.pyNN.models.abstract_models.abstract_accepts_incoming_
 method), 34
C
 c (spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh clear_connection_cache () (spyn-
 attribute), 121 naker.pyNN.models.abstract_models.AbstractAcceptsIncomingSyn
 method), 37
 c (spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh clear_connection_cache () (spyn-
 attribute), 124 naker.pyNN.models.neuron.abstract_population_vertex.AbstractP
 can_record () (spyn- naker.pyNN.models.pynn_population_common.PyNNPopulationCommon
 method), 199
 method), 230 clear_connection_cache () (spyn-
 cdf () (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.NeuronAbstractPopulationVertex
 method), 244 method), 201
 cdf () (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats clear_connection_cache () (spyn-
 method), 245 naker.pyNN.models.neuron.synaptic_manager.SynapticManager
 method), 200
 changes_during_run (spyn- clear_connection_cache () (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics (spyn-
 attribute), 144 naker.pyNN.models.neuron.SynapticManager
 method), 207
 changes_during_run (spyn- clear_recording () (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics (spyn-
 attribute), 157 naker.pyNN.models.common.abstract_neuron_recordable.Abstract
 method), 40
 changes_during_run (spyn- clear_recording () (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic (spyn-
 attribute), 146 naker.pyNN.models.common.AbstractNeuronRecordable
 method), 45
 changes_during_run (spyn- clear_recording () (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP (spyn-
 attribute), 147 naker.pyNN.models.neuron.abstract_population_vertex.AbstractP
 method), 191
 changes_during_run (spyn- clear_recording () (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStatic
 method), 191

`naker.pyNN.models.neuron.AbstractPopulationVertex` attribute), 252
`method`), 201
`configure_master_key()` (spyn- `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` attribute), 235
`clear_spike_recording()` (spyn- `naker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable` method), 41
`configure_master_key()` (spyn- `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` attribute), 240
`clear_spike_recording()` (spyn- `naker.pyNN.models.common.AbstractSpikeRecordable` method), 46
`configure_master_key_key` (spyn- `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` attribute), 235
`clear_spike_recording()` (spyn- `naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex` attribute), 191
`configure_master_key_key` (spyn- `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` attribute), 240
`clear_spike_recording()` (spyn- `naker.pyNN.models.neuron.AbstractPopulationVertex` attribute), 202
`conn_list` (`spynnaker.pyNN.models.neural_projections.connectors.from_array_vertex.SpikeSourceArrayVertex` attribute), 59
`clear_spike_recording()` (spyn- `naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex` attribute), 73
`clear_spike_recording()` (spyn- `ConnectionHolder` (class in `spynnaker.pyNN.models.neural_projections.connectors.FromArrayVertex.SpikeSourceArrayVertex` attribute), 207
`clear_spike_recording()` (spyn- `ConnectionHolder` (class in `spynnaker.pyNN.models.neuron.connection_holder`), 197
`clear_spike_recording()` (spyn- `naker.pyNN.models.utility_models.spike_injector.spike_injector_vertex.SpikeInjectorVertex` attribute), 227
`connections` (spyn- `spynnaker.pyNN.connections.ethernet_control_connection_holder.EthernetControlConnectionHolder` attribute), 197
`close()` (`spynnaker.pyNN.external_devices_models.push_bot_push_bot_ethernet.push_bot_wifi_connection.PushBotWiFiConnection` attribute), 9
`cm` (`spynnaker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire.NeuronModelLeakyIntegrateAndFire` attribute), 122
`cm` (`spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire` attribute), 126
`connector` (`spynnaker.pyNN.models.neural_projections.synapse_information_from_list_connection_from_list_connector.ConnectionFromListConnector` attribute), 59
`connector` (`spynnaker.pyNN.models.neural_projections.SynapseInformationFromListConnector` attribute), 248
`column_names` (spyn- `naker.pyNN.models.neural_projections.connectors.ConnectionFromListConnector` attribute), 73
`compute_statistics()` (spyn- `naker.pyNN.models.neural_projections.connectors.kernel_connector.KernelConnector` attribute), 62
`compute_statistics()` (spyn- `convert_per_connection_data_to_rows()` (in module `spynnaker.pyNN.utilities.utility_calls`), 250
`compute_statistics()` (spyn- `naker.pyNN.models.neural_projections.connectors.KernelConnector` attribute), 77
`conductance_based` (spyn- `convert_per_connection_data_to_rows()` (in module `spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics` attribute), 157
`conductance_based` (spyn- `convert_rate()` (spyn- `naker.pyNN.models.neuron.AbstractPopulationVertex` attribute), 202
`conductance_based` (spyn- `convert_to()` (in module `spynnaker.pyNN.models.pynn_population_common.PyNNPopulationCommon` attribute), 230
`CONFIG_FILE_NAME` (spyn- `naker.pyNN.models.neural_projections.connectors.kernel_connector.KernelConnector` attribute), 250
`naker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon`

`create()` (`spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.FakeHBPPortalMachineProvider.connectors.array_connect`
`method`), 249

`create_machine_edge()` (`spynnaker.pyNN.models.neural_projections.delay_afferent_application.DelayAfferentApplicationEdge.connectors.ArrayConnect`
`method`), 79

`create_machine_edge()` (`spynnaker.pyNN.models.neural_projections.DelayAfferentApplicationEdge.connectors.csa_connector`
`method`), 82

`create_machine_edge()` (`spynnaker.pyNN.models.neural_projections.delayed_application.DelayedApplicationEdge.connectors.CSAConnector`
`method`), 80

`create_machine_edge()` (`spynnaker.pyNN.models.neural_projections.DelayedApplicationEdge.connectors.distance_depe`
`method`), 83

`create_machine_edge()` (`spynnaker.pyNN.models.neural_projections.projection_application.ProjectionApplicationEdge.connectors.DistanceDepe`
`method`), 81

`create_machine_edge()` (`spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge.connectors.fixed_number`
`method`), 84

`create_machine_vertex()` (`spynnaker.pyNN.external_devices_models.munich_spinnaker_link.MunichMotorDevice.connectors.fixed_number`
`method`), 22

`create_machine_vertex()` (`spynnaker.pyNN.external_devices_models.MunichMotorDevice.connectors.fixed_probabil`
`method`), 30

`create_machine_vertex()` (`spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex.connectors.FixedNumberH`
`method`), 191

`create_machine_vertex()` (`spynnaker.pyNN.models.neuron.AbstractPopulationVertex.connectors.FixedNumberH`
`method`), 202

`create_machine_vertex()` (`spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex.connectors.FixedProbabil`
`method`), 215

`create_machine_vertex()` (`spynnaker.pyNN.models.utility_models.delays.delay_extension.DelayExtensionVertex.connectors.from_list_conr`
`method`), 221

`create_machine_vertex()` (`spynnaker.pyNN.models.utility_models.delays.DelayExtensionVertex.connectors.FromListConn`
`method`), 224

`create_synaptic_block()` (`spynnaker.pyNN.models.neural_projections.connectors.abstract_neural_projections.connectors.index_based_p`
`method`), 50

`create_synaptic_block()` (`spynnaker.pyNN.models.neural_projections.connectors.AbstractNeuralProjections.connectors.IndexBasedPro`
`method`), 65

`create_synaptic_block()` (`spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connectors.all_to_all_kernel_connec`
`method`), 52

`create_synaptic_block()` (`spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnectors.connectors.KernelConnec`
`method`), 67

`create_synaptic_block()` (`spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnectors.connectors.KernelConnec`
`method`), 77

`naker.pyNN.models.neural_projections.connectors.multipopulationmachinevertex.PopulationMachineVertex`
`method`), 63

`create_synaptic_block()` (spyn- `CURRENT_TIMER_TIC` (spyn-
`naker.pyNN.models.neural_projections.connectors.MultipopulationMachineVertex`
`method`), 75

`create_synaptic_block()` (spyn- `CURRENT_TIMER_TIC` (spyn-
`naker.pyNN.models.neural_projections.connectors.one_to_one_connector.OneToOneConnector`
`method`), 64

`create_synaptic_block()` (spyn- `OneToOneConnector`
`method`), 76

`create_synaptic_block()` (spyn- `attribute`), 121
`naker.pyNN.models.neural_projections.connectors.small_world_connector.SmallWorldConnector`
`method`), 65

`create_synaptic_block()` (spyn- `attribute`), 124
`naker.pyNN.models.neural_projections.connectors.distance_dependent_distance_dependent`
`method`), 76

`create_vertex()` (spyn- `d_expression` (spyn-
`naker.pyNN.external_devices_models.external_device_lif_control.ExternalDeviceLifControl`
`method`), 18

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.external_devices_models.ExternalDeviceLifControl`
`method`), 28

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.abstract_pynn_model.AbstractPyNNModel`
`method`), 229

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.neuron.abstract_pynn_neuron_model.AbstractPyNNNeuronModel`
`method`), 196

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.neuron.AbstractPyNNNeuronModel`
`method`), 209

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.spike_source.spike_source_array.SpikeSourceArray`
`method`), 210

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson`
`method`), 212

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.spike_source.SpikeSourceArray`
`method`), 218

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.spike_source.SpikeSourcePoisson`
`method`), 218

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.utility_models.spike_injector.spike_injector.SpikeInjector`
`method`), 226

`create_vertex()` (spyn- `default_initial_values` (spyn-
`naker.pyNN.models.utility_models.spike_injector.SpikeInjector`
`method`), 228

`CSAConnector` (class in spyn- `default_initial_values` (spyn-
`naker.pyNN.models.neural_projections.connectors`),
68

`CSAConnector` (class in spyn-
`naker.pyNN.models.neural_projections.connectors`),
68

[attribute](#)), 85
[delay_block](#) (spyn- [DelayBlock](#) (class in spyn- [naker.pyNN.models.utility_models.delays.delay_block.DelayBlock](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 219
[delay_block](#) (spyn- [DelayExtensionMachineVertex.EXTRA_PROVENANCE_DATA_I](#) (class in spyn- [naker.pyNN.models.utility_models.delays.DelayBlock](#) (class in spyn- [naker.pyNN.models.utility_models.delays.delay_extension_machi](#) [attribute](#)), 223
[delay_edge](#) (spynnaker.pyNN.models.neural_projections.projection_application_edge.ProjectionApplicationEdge [attribute](#)), 81
[delay_edge](#) (spynnaker.pyNN.models.neural_projections.ProjectionApplicationEdge (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 84
[DelayAfferentApplicationEdge](#) (class in spyn- [naker.pyNN.models.neural_projections](#)), 82
[DelayAfferentApplicationEdge](#) (class in spyn- [naker.pyNN.models.neural_projections.delay_afferent_application_edge](#) (class in spyn- [naker.pyNN.models.utility_models.delays.delay_generator_data](#)), 79
[DelayAfferentMachineEdge](#) (class in spyn- [naker.pyNN.models.neural_projections](#)), 82
[DelayAfferentMachineEdge](#) (class in spyn- [naker.pyNN.models.neural_projections.delay_afferent_machine_edge](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), 79
[DelayBlock](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), 223
[DelayBlock](#) (class in spyn- [naker.pyNN.models.utility_models.delays.delay_block](#)), 219
[delayed_max_bytes](#) (spyn- [naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 170
[delayed_max_n_synapses](#) (spyn- [naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 170
[delayed_max_words](#) (spyn- [naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 170
[DelayedApplicationEdge](#) (class in spyn- [naker.pyNN.models.neural_projections](#)), 83
[DelayedApplicationEdge](#) (class in spyn- [naker.pyNN.models.neural_projections.delayed_application_edge](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 80
[DelayedMachineEdge](#) (class in spyn- [naker.pyNN.models.neural_projections](#)), 83
[DelayedMachineEdge](#) (class in spyn- [naker.pyNN.models.neural_projections.delayed_machine_edge](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 80
[DelayExtensionException](#), 254
[DelayExtensionMachineVertex](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), 223
[DelayExtensionMachineVertex](#) (class in spyn- [naker.pyNN.models.utility_models.delays.delay_extension_machine_vertex](#) (class in spyn- [naker.pyNN.models.utility_models.delays](#)), [attribute](#)), 219

<code>naker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>attribute), 27</code>	<code>attribute), 6</code>
<code>device_control_key</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>device_control_uses_payload</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>
<code>naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>attribute), 5</code>	<code>attribute), 18</code>	
<code>device_control_max_value</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>device_control_uses_payload</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>attribute), 17</code>	<code>attribute), 28</code>	
<code>device_control_max_value</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>device_control_uses_payload</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>
<code>naker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>attribute), 27</code>	<code>attribute), 6</code>	
<code>device_control_max_value</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>DIRECT_MATRIX</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>
<code>naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>attribute), 5</code>	<code>attribute), 248</code>	
<code>device_control_min_value</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>disable_motor()</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>attribute), 17</code>	<code>static method), 234</code>	
<code>device_control_min_value</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>disable_motor()</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>attribute), 27</code>	<code>static method), 239</code>	
<code>device_control_min_value</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>disable_retina()</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>
<code>naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>attribute), 5</code>	<code>static method), 234</code>	
<code>device_control_partition_id</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>disable_retina()</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>attribute), 17</code>	<code>method), 235</code>	
<code>device_control_partition_id</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>disable_retina()</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>attribute), 27</code>	<code>static method), 239</code>	
<code>device_control_partition_id</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>disable_retina()</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>
<code>naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>attribute), 6</code>	<code>method), 240</code>	
<code>device_control_scaling_factor</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>disable_retina_key</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>attribute), 17</code>	<code>attribute), 235</code>	
<code>device_control_scaling_factor</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>disable_retina_key</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>attribute), 27</code>	<code>attribute), 240</code>	
<code>device_control_send_type</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>distance()</code>	<code>(spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics</code>
<code>naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>attribute), 18</code>	<code>distance()</code>	<code>(spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics</code>
<code>device_control_send_type</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>method), 164</code>	
<code>naker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>attribute), 27</code>	<code>(class in spynnaker.pyNN.models.neural_projections.connectors),</code>	
<code>device_control_send_type</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>DistanceDependentProbabilityConnector</code>	<code>(spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>
<code>naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_device.PushBotEthernetDevice</code>	<code>attribute), 6</code>		
<code>device_control_timesteps_between_sending</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>(class in spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>(spynnaker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>
<code>naker.pyNN.external_devices_models.abstract_multicast_controllable_device.AbstractMulticastControllableDevice</code>	<code>attribute), 18</code>	<code>55</code>	
<code>device_control_timesteps_between_sending</code>	<code>(spynnaker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>DOWN_POLARITY</code>	<code>(spynnaker.pyNN.external_devices_models.external_spinnaker_link_factory.ExternalSpinnakerLinkFactory</code>
<code>naker.pyNN.external_devices_models.AbstractMulticastControllableDevice</code>	<code>attribute), 28</code>	<code>attribute), 21</code>	
<code>device_control_timesteps_between_sending</code>	<code>DOWN_POLARITY</code>	<code>(spynnaker.pyNN.external_devices_models.external_spinnaker_link_factory.ExternalSpinnakerLinkFactory</code>	

`naker.pyNN.external_devices_models.ExternalFPGAReinDevice` `naker.pyNN.models.neuron.input_types.InputTypeConductance`
 attribute), 29 attribute), 111
 DOWN_POLARITY (spyn- e_rev_I (spynnaker.pyNN.models.neuron.input_types.input_type_conduc
`naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.MunichRetinaDevice`
 attribute), 24 e_rev_I (spynnaker.pyNN.models.neuron.input_types.InputTypeConductance
 DOWN_POLARITY (spyn- attribute), 111
`naker.pyNN.external_devices_models.MunichRetinaDevice` `partition_identifiers_for_dependent_vertex()`
 attribute), 32 (spynnaker.pyNN.external_devices_models.external_device_lif_co
 DOWNSAMPLE_16_X_16 (spyn- method), 20
`naker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_resolution.PushBotRetinaResolution`
 attribute), 12 (spynnaker.pyNN.external_devices_models.munich_spinnaker_lin
 DOWNSAMPLE_16_X_16 (spyn- method), 23
`naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotRetinaResolution` `dependent_vertex()`
 attribute), 14 (spynnaker.pyNN.external_devices_models.MunichMotorDevice
 DOWNSAMPLE_16_X_16 (spyn- method), 30
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey` `order` (class in spyn-
 attribute), 238 `naker.pyNN.models.common`), 47
 DOWNSAMPLE_16_X_16 (spyn- EIEIOSpikeRecorder (class in spyn-
`naker.pyNN.protocols.RetinaKey` attribute), `naker.pyNN.models.common.eieio_spike_recorder`),
 243 42
 DOWNSAMPLE_32_X_32 (spyn- EIFConductanceAlphaPopulation (class in
`naker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_resolution.PushBotRetinaResolution`
 attribute), 12 EIFConductanceAlphaPopulation
 DOWNSAMPLE_32_X_32 (spyn- (class in spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotRetinaResolution` `build_cond_alpha_isfa_ista`),
 attribute), 14 89
 DOWNSAMPLE_32_X_32 (spyn- `enable_disable_motor_key` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey` `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich`
 attribute), 238 attribute), 235
 DOWNSAMPLE_32_X_32 (spyn- `enable_disable_motor_key` (spyn-
`naker.pyNN.protocols.RetinaKey` attribute), `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
 243 attribute), 240
 DOWNSAMPLE_64_X_64 (spyn- `enable_motor()` (spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_parameters.push_bot_retina_resolution.PushBotRetinaResolution`
 attribute), 12 static method), 234
 DOWNSAMPLE_64_X_64 (spyn- `enable_motor()` (spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotRetinaResolution` `EthernetProtocol`
 attribute), 14 static method), 239
 DOWNSAMPLE_64_X_64 (spyn- `enable_retina()` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.RetinaKey` `naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEth`
 attribute), 238 static method), 234
 DOWNSAMPLE_64_X_64 (spyn- `enable_retina()` (spyn-
`naker.pyNN.protocols.RetinaKey` attribute), `naker.pyNN.protocols.MunichIoEthernetProtocol`
 244 static method), 239
 du_th (spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic.ThresholdTypeMaassStochastic
 attribute), 185 `naker.pyNN.connections.ethernet_control_connection`),
 du_th (spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic
 attribute), 189 EVENTS_IN_PAYLOAD (spyn-
 duration (spynnaker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson) `naker.pyNN.spinnaker_link_protocol.Retina`
 attribute), 215 attribute), 239
 EVENTS_IN_PAYLOAD (spyn-
 E `naker.pyNN.protocols.RetinaPayload` attribute),
 e_rev_E (spynnaker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance
 attribute), 108 exc_response (spyn-

`naker.pyNN.models.neuron.synapse_types.synapse_type_alpha` `spynaker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha` `filter_edge()` `(spynaker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha attribute), 173`

`exc_response` `(spynaker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha attribute), 184` `filter_edge()` `(spynaker.pyNN.models.neural_projections.DelayAfferentMachineEdge method), 83`

`ExternalCochleaDevice` `(class in spynaker.pyNN.external_devices_models), 28` `filter_edge()` `(spynaker.pyNN.models.neural_projections.delayed_machine_edge.D method), 80`

`ExternalCochleaDevice` `(class in spynaker.pyNN.external_devices_models.external_spinnaker_linker.pyNN.models), 20` `filter_edge()` `(spynaker.pyNN.models.neural_projections.DelayedMachineEdge method), 83`

`ExternalDeviceLifControl` `(class in spynaker.pyNN.external_devices_models), 28` `filter_edge()` `(spynaker.pyNN.models.neural_projections.projection_machine_edge method), 81`

`ExternalDeviceLifControl` `(class in spynaker.pyNN.external_devices_models.external_device_lif_control), 18` `filter_edge()` `(spynaker.pyNN.models.neural_projections.ProjectionMachineEdge method), 84`

`ExternalDeviceLifControlVertex` `(class in spynaker.pyNN.external_devices_models.external_device_lif_control), 19` `FilterableException`, 254 `filter_edge()` `(spynaker.pyNN.models.neuron.connection_holder.Connection method), 197`

`ExternalFPGARetinaDevice` `(class in spynaker.pyNN.external_devices_models), 28` `finish()` `(spynaker.pyNN.models.neuron.ConnectionHolder method), 207`

`ExternalFPGARetinaDevice` `(class in spynaker.pyNN.external_devices_models.external_spinnaker_linker.pyNN.models), 21` `finish_master_pop_table()` `(spynaker.pyNN.models.neuron.master_pop_table_generators.abstract method), 115`

`extract_synaptic_matrix_data_location()` `finish_master_pop_table()` `(spynaker.pyNN.models.neuron.master_pop_table_generators.abstract method), 115` `finish_master_pop_table()` `(spynaker.pyNN.models.neuron.master_pop_table_generators.abstract method), 116`

`extract_synaptic_matrix_data_location()` `finish_master_pop_table()` `(spynaker.pyNN.models.neuron.master_pop_table_generators.abstract method), 116` `finish_master_pop_table()` `(spynaker.pyNN.models.neuron.master_pop_table_generators.abstract method), 118`

`extract_synaptic_matrix_data_location()` `first_id()` `(spynaker.pyNN.models.pyNN_population_common.PyNNPop method), 118` `FIXED_KEY` `(spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol attribute), 238`

`ExtractedData` `(class in spynaker.pyNN.utilities.extracted_data), 248` `FIXED_KEY` `(spynaker.pyNN.protocols.RetinaKey attribute), 244`

F `FIXED_NUMBER_POST_CONNECTOR` `(spynaker.pyNN.models.neural_projections.connectors.abstract_gener attribute), 52`

`FakeHBPPortalMachineProvider` `(class in spynaker.pyNN.utilities.fake_HBP_Portal_machine_provider), 249` `FIXED_NUMBER_PRE_CONNECTOR` `(spynaker.pyNN.models.neural_projections.connectors.abstract_gener attribute), 52`

`field_types` `(spynaker.pyNN.models.neuron.implementations.Struct attribute), 102` `FIXED_PROBABILITY_CONNECTOR` `(spynaker.pyNN.models.neural_projections.connectors.abstract_gener attribute), 52`

`field_types` `(spynaker.pyNN.models.neuron.implementations.struct.Struct attribute), 99` `FIXED_TOTAL_NUMBER_CONNECTOR` `(spynaker.pyNN.models.neural_projections.connectors.abstract_gener attribute), 52`

`filter_edge()` `(spynaker.pyNN.models.abstract_models.abstract_filterable_edge.AbstractFilterableEdge method), 35` `FixedNumberPostConnector` `(class in spynaker.pyNN.models.neural_projections.connectors), 70`

`filter_edge()` `(spynaker.pyNN.models.abstract_models.AbstractFilterableEdge method), 38` `FixedNumberPostConnector` `(class in spynaker.pyNN.models.neural_projections.connectors.fixed_number method), 56`

`filter_edge()` `(spynaker.pyNN.models.neural_projections.delay_afferent_machine_edge.DelayAfferentMachineEdge method), 56`

FixedNumberPreConnector (class in spyn- gen_connector_id (spyn-
 naker.pyNN.models.neural_projections.connectors), naker.pyNN.models.neural_projections.connectors.FixedProbabil
 71 attribute), 72
 FixedNumberPreConnector (class in spyn- gen_connector_id (spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector), naker.pyNN.models.neural_projections.connectors.kernel_connec
 57 attribute), 62
 FixedProbabilityConnector (class in spyn- gen_connector_id (spyn-
 naker.pyNN.models.neural_projections.connectors), naker.pyNN.models.neural_projections.connectors.KernelConnec
 72 attribute), 78
 FixedProbabilityConnector (class in spyn- gen_connector_id (spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_probability_connector), naker.pyNN.models.neural_projections.connectors.multipse_conn
 58 attribute), 63
 float_to_fixed() (in module spyn- gen_connector_id (spyn-
 naker.pyNN.models.neuron.plasticity.stdp.common.plasticity_helpers), naker.pyNN.models.neural_projections.connectors.MultipseConn
 127 attribute), 75
 FREE (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES, spyn-
 attribute), 235 naker.pyNN.models.neural_projections.connectors.one_to_one_connector
 FREE (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.MODES, spyn-
 attribute), 240 gen_connector_id (spyn-
 FromListConnector (class in spyn- naker.pyNN.models.neural_projections.connectors.OneToOneConn
 naker.pyNN.models.neural_projections.connectors), attribute), 76
 73 gen_connector_params () (spyn-
 FromListConnector (class in spyn- naker.pyNN.models.neural_projections.connectors.abstract_gener
 naker.pyNN.models.neural_projections.connectors.from_list_generator), method), 66
 59 gen_connector_params () (spyn-
 G naker.pyNN.models.neural_projections.connectors.AbstractGener
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.abstract_generator), method), 53
 attribute), 51 naker.pyNN.models.neural_projections.connectors.AllToAllConn
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.AbstractGenerator), method), 67
 attribute), 66 naker.pyNN.models.neural_projections.connectors.fixed_number_
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.all_to_all_connector), method), 56
 attribute), 53 naker.pyNN.models.neural_projections.connectors.fixed_number_
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.AllToAllConnector), method), 57
 attribute), 67 naker.pyNN.models.neural_projections.connectors.fixed_probabil
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector), method), 58
 attribute), 56 naker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector), method), 70
 attribute), 57 naker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.fixed_probability_connector), method), 72
 attribute), 58 naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector), method), 72
 attribute), 70 naker.pyNN.models.neural_projections.connectors.FixedProbability
 gen_connector_id (spyn- gen_connector_params () (spyn-
 naker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector), method), 62
 attribute), 71 naker.pyNN.models.neural_projections.connectors.kernel_connec

`gen_connector_params()` (spyn- attribute), 223
`naker.pyNN.models.neural_projections.connectors.kernel_connector_params()` (spyn-
method), 78
`gen_connector_params()` (spyn- method), 51
`naker.pyNN.models.neural_projections.connectors.multipse_connector.MultipseConnector` (spyn-
method), 63
`gen_connector_params()` (spyn- method), 66
`naker.pyNN.models.neural_projections.connectors.multipse_connector.Params()` (spyn-
method), 75
`gen_connector_params_size_in_bytes` (spyn- method), 62
`naker.pyNN.models.neural_projections.connectors.abstract_generate(connector_on_machine.AbstractGenerateConnector`
attribute), 51
`gen_connector_params_size_in_bytes` (spyn- method), 78
`naker.pyNN.models.neural_projections.connectors.AbstractGenerateConnector.OnMachine()` (spyn-
attribute), 66
`gen_connector_params_size_in_bytes` (spyn- method), 51
`naker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector()` (spyn-
attribute), 53
`gen_connector_params_size_in_bytes` (spyn- method), 67
`naker.pyNN.models.neural_projections.connectors.AllToAllConnector.size_in_bytes()` (spyn-
attribute), 67
`gen_connector_params_size_in_bytes` (spyn- method), 62
`naker.pyNN.models.neural_projections.connectors.fixed_number_posts_connector.FixedNumberPostConnector`
attribute), 56
`gen_connector_params_size_in_bytes` (spyn- method), 78
`naker.pyNN.models.neural_projections.connectors.fixed_number_pre_connector.FixedNumberPreConnector`
attribute), 57
`gen_connector_params_size_in_bytes` (spyn- method), 52
`naker.pyNN.models.neural_projections.connectors.fixed_probability_connector.FixedProbabilityConnector`
attribute), 58
`gen_connector_params_size_in_bytes` (spyn- method), 67
`naker.pyNN.models.neural_projections.connectors.FixedNumberPostConnector` (spyn-
attribute), 71
`gen_connector_params_size_in_bytes` (spyn- method), 62
`naker.pyNN.models.neural_projections.connectors.FixedNumberPreConnector` (spyn-
attribute), 72
`gen_connector_params_size_in_bytes` (spyn- method), 78
`naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector` (spyn-
attribute), 73
`gen_connector_params_size_in_bytes` (spyn- attribute), 142
`naker.pyNN.models.neural_projections.connectors.kernel_connector.KernelConnector` (spyn-
attribute), 62
`gen_connector_params_size_in_bytes` (spyn- attribute), 158
`naker.pyNN.models.neural_projections.connectors.KernelConnector` (spyn-
attribute), 78
`gen_connector_params_size_in_bytes` (spyn- attribute), 146
`naker.pyNN.models.neural_projections.connectors.multipse_connector.MultipseConnector` (spyn-
attribute), 64
`gen_connector_params_size_in_bytes` (spyn- attribute), 148
`naker.pyNN.models.neural_projections.connectors.MultipseConnector` (spyn-
attribute), 75
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics`
`gen_data(spynnaker.pyNN.models.neuron.generator_data.GeneratorData)` (spyn- attribute), 159
`gen_data(spynnaker.pyNN.models.utility_models.delays.delay_generator.DelayGeneratorData)` (spyn-
attribute), 198
`gen_matrix_id` (spyn- attribute), 198
`naker.pyNN.models.utility_models.delays.delay_generator.DelayGeneratorData` (spyn- attribute), 198

`attribute`), 161
`gen_matrix_params` (`spyn-` `gen_weights_id()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.abstract_generator`
`attribute`), 142
`gen_matrix_params` (`spyn-` `gen_weights_id()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.AbstractGenerator`
`attribute`), 158
`gen_matrix_params` (`spyn-` `gen_weights_id()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics`
`attribute`), 148
`gen_matrix_params` (`spyn-` `gen_weights_id()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics`
`attribute`), 161
`gen_matrix_params_size_in_bytes` (`spyn-` `gen_weights_params()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.abstract_generator`
`attribute`), 143
`gen_matrix_params_size_in_bytes` (`spyn-` `gen_weights_params()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.AbstractGenerator`
`attribute`), 158
`gen_matrix_params_size_in_bytes` (`spyn-` `gen_weights_params()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics`
`attribute`), 148
`gen_matrix_params_size_in_bytes` (`spyn-` `gen_weights_params()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics`
`attribute`), 161
`gen_on_machine()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method`), 191
`gen_on_machine()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.neuron.AbstractPopulationVertex` `naker.pyNN.external_devices_models.MunichMotorDevice`
`method`), 202
`gen_on_machine()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.neuron.synaptic_manager.SynapticManager` `naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method`), 200
`gen_on_machine()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.neuron.SynapticManager` `naker.pyNN.models.neuron.AbstractPopulationVertex`
`method`), 207
`gen_on_machine()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.utility_models.delays.delay_extension_vertex` `naker.pyNN.models.utility_models.delay_extension_vertex`
`method`), 221
`gen_on_machine()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.utility_models.delays.DelayExtensionVertex` `naker.pyNN.models.utility_models.delays.delay_extension_vertex`
`method`), 225
`gen_weight_params_size_in_bytes()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.abstract_generator` `naker.pyNN.models.utility_models.delays.delay_extension_vertex`
`method`), 52
`gen_weight_params_size_in_bytes()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.AbstractGenerator` `naker.pyNN.models.utility_models.delays.delay_extension_vertex`
`method`), 67
`gen_weight_params_size_in_bytes()` (`spyn-` `generate_data_specification()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.kernel_connection` `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics`
`method`), 62
`gen_weight_params_size_in_bytes()` (`spyn-` `generate_on_machine` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.KernelConnection` `naker.pyNN.models.neural_projections.connectors.kernel_connection`
`method`), 78

`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method), 191` `get_buffered_sdram()` (spyn-
`naker.pyNN.models.common.NeuronRecorder`
`naker.pyNN.models.neuron.AbstractPopulationVertex` `method), 47`
`method), 202` `get_buffered_sdram_per_record()` (spyn-
`naker.pyNN.models.common.neuron_recorder.NeuronRecorder`
`naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex`
`method), 215` `get_buffered_sdram_per_record()` (spyn-
`naker.pyNN.models.common.NeuronRecorder`
`naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex`
`method), 221` `get_buffered_sdram_per_timestep()` (spyn-
`naker.pyNN.models.common.neuron_recorder.NeuronRecorder`
`naker.pyNN.models.utility_models.delays.DelayExtensionVertex` `method), 43`
`method), 225` `get_buffered_sdram_per_timestep()` (spyn-
`naker.pyNN.models.common.NeuronRecorder`
`naker.pyNN.external_devices_models.munich_spinnaker_linker_device.MunichMotorDevice`
`method), 23` `get_by_selector()` (spyn-
`naker.pyNN.models.pyNN_population_common.PyNNPopulationCommon`
`naker.pyNN.external_devices_models.MunichMotorDevice` `method), 231`
`method), 31` `get_colour()` (in module `spynnaker.plot_utils`), 258
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method), 192` `get_connection_holders()` (spyn-
`naker.pyNN.models.neuron.AbstractPopulationVertex` `method), 203`
`method), 202` `get_connection_holders()` (spyn-
`naker.pyNN.models.neuron.AbstractPopulationVertex` `method), 203`
`naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex`
`method), 216` `get_connection_holders()` (spyn-
`naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex`
`method), 221` `get_connections_from_machine()` (spyn-
`naker.pyNN.models.utility_models.delays.DelayExtensionVertex` `method), 225`
`method), 225` `get_connections_from_machine()` (spyn-
`naker.pyNN.models.abstact_models.abstact_accepts_incoming_synaptic_manager.SynapticManager`
`method), 34`
`naker.pyNN.models.neuron.synapse_io.abstract_synapse_io_row_based_synapse_io_row_based`
`method), 170` `get_connections_from_machine()` (spyn-
`naker.pyNN.models.neuron.synapse_io.AbstractSynapseIO` `naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method), 171` `method), 192`
`naker.pyNN.models.neuron.synapse_io.synapse_io_row_based_synapse_io_row_based`
`method), 171` `method), 203`
`naker.pyNN.models.neuron.synapse_io.SynapseIORowBased` `naker.pyNN.models.neuron.synaptic_manager.SynapticManager`
`method), 172` `method), 200`
`naker.pyNN.models.common`, 49 `get_connections_from_machine()` (spyn-
`naker.pyNN.models.neuron.SynapticManager`
`method), 207`
`naker.pyNN.models.common.recording_utils`, 44 `get_cpu_usage_for_atoms()` (spyn-
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method), 192`
`naker.pyNN.models.common.neuron_recorder.NeuronRecorder` `usage_for_atoms()` (spyn-

<code>naker.pyNN.models.neuron.AbstractPopulationVertex</code>	<code>get_database_connection()</code>	(spyn-
<code>method</code>), 203	<code>naker.pyNN.external_devices_models.abstract_ethernet_sensor.A</code>	
<code>get_cpu_usage_for_atoms()</code>	(spyn-	<code>method</code>), 17
<code>naker.pyNN.models.spike_source.spike_source_poisson_vertex</code>	<code>SpikeSourcePoissonVertex</code>	(spyn-
<code>static method</code>), 216	<code>naker.pyNN.external_devices_models.AbstractEthernetSensor</code>	
<code>get_cpu_usage_for_atoms()</code>	(spyn-	<code>method</code>), 26
<code>naker.pyNN.models.utility_models.delays.delay_extension_vertex</code>	<code>DelayExtensionVertex</code>	(spyn-
<code>method</code>), 221	<code>naker.pyNN.models.neural_properties.neural_parameter.NeuronP</code>	
<code>get_cpu_usage_for_atoms()</code>	(spyn-	<code>method</code>), 85
<code>naker.pyNN.models.utility_models.delays.DelayExtensionVertex</code>	<code>spec_datatype()</code>	(spyn-
<code>method</code>), 225	<code>naker.pyNN.models.neural_properties.NeuronParameter</code>	
<code>get_current_time()</code>	(spyn-	<code>method</code>), 85
<code>naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface</code>	<code>get_maximum()</code>	(spyn-
<code>method</code>), 257	<code>naker.pyNN.models.neural_projections.connectors.abstract_conn</code>	
<code>get_current_time()</code>	(spyn-	<code>method</code>), 50
<code>naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState</code>	<code>maximum()</code>	(spyn-
<code>method</code>), 249	<code>naker.pyNN.models.neural_projections.connectors.AbstractConne</code>	
<code>get_data()</code>	(in module spyn-	<code>method</code>), 65
<code>naker.pyNN.models.common</code>), 49	<code>get_delay_maximum()</code>	(spyn-
<code>get_data()</code>	(in module spyn-	<code>naker.pyNN.models.neural_projections.connectors.all_to_all_con</code>
<code>naker.pyNN.models.common.recording_utils</code>),	<code>method</code>), 53	
44	<code>get_delay_maximum()</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.common.abstract_neuron_recorder.NeuronRecorder)	<code>naker.pyNN.models.neural_projections.connectors.AllToAllConne</code>
<code>method</code>), 40	<code>method</code>), 68	
<code>get_data()</code>	(spynnaker.pyNN.models.common.AbstractNeuronRecorder)	<code>maximum()</code>
<code>method</code>), 45	<code>naker.pyNN.models.neural_projections.connectors.array_connect</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder)	<code>method</code>), 54
<code>method</code>), 43	<code>get_delay_maximum()</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.common.NeuronRecorder	<code>naker.pyNN.models.neural_projections.connectors.ArrayConnect</code>
<code>method</code>), 48	<code>method</code>), 68	
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex	(spyn-
<code>method</code>), 192	<code>naker.pyNN.models.neural_projections.connectors.csa_connector</code>	
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.AbstractPopulationVertex	<code>method</code>), 54
<code>method</code>), 203	<code>get_delay_maximum()</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.abstract_neuron_impl.AbstractNeuronImpl	<code>naker.pyNN.models.neural_projections.connectors.CSAConnector</code>
<code>method</code>), 93	<code>method</code>), 69	
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComp	(spyn-
<code>method</code>), 95	<code>naker.pyNN.models.neural_projections.connectors.distance_depe</code>	
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.AbstractNeuronImpl	<code>method</code>), 70
<code>method</code>), 105	<code>get_delay_maximum()</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.AbstractNeuronImpl	<code>naker.pyNN.models.neural_projections.connectors.DistanceDepe</code>
<code>method</code>), 100	<code>method</code>), 70	
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.neuron_impl_standard.NeuronImplStandard	(spyn-
<code>method</code>), 97	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_</code>	
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.NeuronImplStandard	<code>method</code>), 103
<code>method</code>), 103	<code>get_delay_maximum()</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.StructuralNeuronImpl	<code>naker.pyNN.models.neural_projections.connectors.fixed_number_</code>
<code>method</code>), 102	<code>method</code>), 57	
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.implementations.StructuralNeuronImpl	<code>maximum()</code>
<code>method</code>), 99	<code>naker.pyNN.models.neural_projections.connectors.fixed_probabil</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel	<code>method</code>), 58
<code>method</code>), 119	<code>get_delay_maximum()</code>	(spyn-
<code>get_data()</code>	(spynnaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel	<code>naker.pyNN.models.neural_projections.connectors.FixedNumberF</code>
<code>method</code>), 123	<code>method</code>), 71	

get_delay_maximum()	(spyn-	get_delay_variance()	(spyn-
naker.pyNN.models.neural_projections.connectors.FixedNumberProbabilityConnector	method), 72	naker.pyNN.models.neural_projections.connectors.FixedNumberProbabilityConnector	method), 59
get_delay_maximum()	(spyn-	get_delay_variance()	(spyn-
naker.pyNN.models.neural_projections.connectors.FixedProbabilityConnector	method), 73	naker.pyNN.models.neural_projections.connectors.FromListConnector	method), 73
get_delay_maximum()	(spyn-	get_delay_variance()	(spyn-
naker.pyNN.models.neural_projections.connectors.from_list_connector	method), 59	naker.pyNN.models.neural_projections.connectors.from_list_connector	method), 144
get_delay_maximum()	(spyn-	get_delay_variance()	(spyn-
naker.pyNN.models.neural_projections.connectors.FromListConnector	method), 73	naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics	method), 157
get_delay_maximum()	(spyn-	get_dict_from_init()	(in module spyn-
naker.pyNN.models.neural_projections.connectors.index_based_probability_connector	method), 60	naker.pyNN.models.neural_projections.connectors.index_based_probability_connector	method), 123
get_delay_maximum()	(spyn-	get_distribution_to_stats()	(spyn-
naker.pyNN.models.neural_projections.connectors.IndexBasedProbabilityConnector	method), 74	naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface	method), 157
get_delay_maximum()	(spyn-	get_distribution_to_stats()	(spyn-
naker.pyNN.models.neural_projections.connectors.kernel_connector	method), 62	naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState	method), 157
get_delay_maximum()	(spyn-	get_dtcm_usage_for_atoms()	(spyn-
naker.pyNN.models.neural_projections.connectors.KernelConnector	method), 78	naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex	method), 192
get_delay_maximum()	(spyn-	get_dtcm_usage_for_atoms()	(spyn-
naker.pyNN.models.neural_projections.connectors.KernelConnector	method), 78	naker.pyNN.models.neuron.AbstractPopulationVertex	method), 192
get_delay_maximum()	(spyn-	get_dtcm_usage_for_atoms()	(spyn-
naker.pyNN.models.neural_projections.connectors.multipse_connector	method), 64	naker.pyNN.models.neuron.multipse_connector	method), 201
get_delay_maximum()	(spyn-	get_dtcm_usage_for_atoms()	(spyn-
naker.pyNN.models.neural_projections.connectors.MultipseConnector	method), 75	naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex	method), 216
get_delay_maximum()	(spyn-	get_dtcm_usage_for_atoms()	(spyn-
naker.pyNN.models.neural_projections.connectors.one_to_one_connector	method), 64	naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex	method), 221
get_delay_maximum()	(spyn-	get_dtcm_usage_for_atoms()	(spyn-
naker.pyNN.models.neural_projections.connectors.OneToOneConnector	method), 76	naker.pyNN.models.utility_models.delays.DelayExtensionVertex	method), 225
get_delay_maximum()	(spyn-	get_dtcm_usage_in_bytes()	(spyn-
naker.pyNN.models.neural_projections.connectors.small_world_connector	method), 65	naker.pyNN.models.common.eieio_spike_recorder.EIEIOSpikeRecorder	method), 227
get_delay_maximum()	(spyn-	get_dtcm_usage_in_bytes()	(spyn-
naker.pyNN.models.neural_projections.connectors.SmallWorldConnector	method), 76	naker.pyNN.models.common.EIEIOSpikeRecorder	method), 227
get_delay_maximum()	(spyn-	get_dtcm_usage_in_bytes()	(spyn-
naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics	method), 144	naker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder	method), 227
get_delay_maximum()	(spyn-	get_dtcm_usage_in_bytes()	(spyn-
naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics	method), 157	naker.pyNN.models.common.MultiSpikeRecorder	method), 227
get_delay_variance()	(spyn-	get_dtcm_usage_in_bytes()	(spyn-
naker.pyNN.models.neural_projections.connectors.abstract_connector	method), 50	naker.pyNN.models.common.neuron_recorder.NeuronRecorder	method), 227
get_delay_variance()	(spyn-	get_dtcm_usage_in_bytes()	(spyn-
naker.pyNN.models.neural_projections.connectors.AbstractConnector	method), 65	naker.pyNN.models.common.NeuronRecorder	method), 227

<code>naker.pyNN.models.neuron.input_types.input_type_conductance</code>	<code>naker.pyNN.models.neuron.input_types.input_type_conductance</code>	<code>naker.pyNN.models.neuron.input_types.input_type_conductance</code>
<code>method), 108</code>	<code>method), 17</code>	<code>method), 17</code>
<code>get_global_weight_scale()</code>	<code>(spyn- get_injector_label()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.neuron.input_types.input_type_current</code>	<code>naker.pyNN.models.neuron.input_types.input_type_current</code>	<code>naker.pyNN.models.neuron.input_types.input_type_current</code>
<code>method), 109</code>	<code>method), 26</code>	<code>method), 26</code>
<code>get_global_weight_scale()</code>	<code>(spyn- get_injector_parameters()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.neuron.input_types.input_type_current</code>	<code>naker.pyNN.models.neuron.input_types.input_type_current</code>	<code>naker.pyNN.models.neuron.input_types.input_type_current</code>
<code>method), 110</code>	<code>method), 17</code>	<code>method), 17</code>
<code>get_global_weight_scale()</code>	<code>(spyn- get_injector_parameters()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.neuron.input_types.InputTypeConductance</code>	<code>naker.pyNN.models.neuron.input_types.InputTypeConductance</code>	<code>naker.pyNN.models.neuron.input_types.InputTypeConductance</code>
<code>method), 111</code>	<code>method), 27</code>	<code>method), 27</code>
<code>get_global_weight_scale()</code>	<code>(spyn- get_kernel_vals()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code>	<code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code>	<code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code>
<code>method), 113</code>	<code>method), 62</code>	<code>method), 62</code>
<code>get_global_weight_scale()</code>	<code>(spyn- get_kernel_vals()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code>	<code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code>	<code>naker.pyNN.models.neuron.input_types.InputTypeCurrent</code>
<code>method), 114</code>	<code>method), 78</code>	<code>method), 78</code>
<code>get_incoming_partition_constraints()</code>	<code>get_local_provenance_data()</code>	<code>(spyn-</code>
<code>(spynnaker.pyNN.models.neuron.abstract_population_vertex</code>	<code>(spynnaker.pyNN.models.neuron.abstract_population_vertex</code>	<code>(spynnaker.pyNN.models.neuron.abstract_population_vertex</code>
<code>method), 192</code>	<code>method), 81</code>	<code>method), 81</code>
<code>get_incoming_partition_constraints()</code>	<code>get_local_provenance_data()</code>	<code>(spyn-</code>
<code>(spynnaker.pyNN.models.neuron.AbstractPopulationVertex</code>	<code>(spynnaker.pyNN.models.neuron.AbstractPopulationVertex</code>	<code>(spynnaker.pyNN.models.neuron.AbstractPopulationVertex</code>
<code>method), 203</code>	<code>method), 84</code>	<code>method), 84</code>
<code>get_incoming_partition_constraints()</code>	<code>get_lut_provenance()</code>	<code>(in module spyn-</code>
<code>(spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager</code>	<code>(spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager</code>	<code>(spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager</code>
<code>method), 200</code>	<code>method), 127</code>	<code>method), 127</code>
<code>get_incoming_partition_constraints()</code>	<code>get_machine_info()</code>	<code>(spyn-</code>
<code>(spynnaker.pyNN.models.neuron.SynapticManager</code>	<code>(spynnaker.pyNN.models.neuron.SynapticManager</code>	<code>(spynnaker.pyNN.models.neuron.SynapticManager</code>
<code>method), 208</code>	<code>method), 249</code>	<code>method), 249</code>
<code>get_index_parameters()</code>	<code>(spyn- get_master_population_table_size()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.common.neuron_recorder.NeuronRecorder</code>	<code>naker.pyNN.models.common.neuron_recorder.NeuronRecorder</code>	<code>naker.pyNN.models.common.neuron_recorder.NeuronRecorder</code>
<code>method), 43</code>	<code>method), 115</code>	<code>method), 115</code>
<code>get_index_parameters()</code>	<code>(spyn- get_master_population_table_size()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.common.NeuronRecorder</code>	<code>naker.pyNN.models.common.NeuronRecorder</code>	<code>naker.pyNN.models.common.NeuronRecorder</code>
<code>method), 48</code>	<code>method), 117</code>	<code>method), 117</code>
<code>get_initial_value()</code>	<code>(spyn- get_master_population_table_size()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.abstract_models.abstract_population_initalize</code>	<code>naker.pyNN.models.abstract_models.abstract_population_initalize</code>	<code>naker.pyNN.models.abstract_models.abstract_population_initalize</code>
<code>method), 35</code>	<code>method), 118</code>	<code>method), 118</code>
<code>get_initial_value()</code>	<code>(spyn- get_matrix_data()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.abstract_models.AbstractPopulationInitializable</code>	<code>naker.pyNN.models.abstract_models.AbstractPopulationInitializable</code>	<code>naker.pyNN.models.abstract_models.AbstractPopulationInitializable</code>
<code>method), 38</code>	<code>method), 43</code>	<code>method), 43</code>
<code>get_initial_value()</code>	<code>(spyn- get_matrix_data()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</code>
<code>method), 192</code>	<code>method), 48</code>	<code>method), 48</code>
<code>get_initial_value()</code>	<code>(spyn- get_max_atoms_per_core()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.neuron.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neuron.AbstractPopulationVertex</code>	<code>naker.pyNN.models.neuron.AbstractPopulationVertex</code>
<code>method), 203</code>	<code>class method), 229</code>	<code>class method), 229</code>
<code>get_initial_values()</code>	<code>(spyn- get_max_atoms_per_core()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.abstract_models.abstract_population_initialize</code>	<code>naker.pyNN.models.abstract_models.abstract_population_initialize</code>	<code>naker.pyNN.models.abstract_models.abstract_population_initialize</code>
<code>method), 35</code>	<code>class method), 196</code>	<code>class method), 196</code>
<code>get_initial_values()</code>	<code>(spyn- get_max_atoms_per_core()</code>	<code>(spyn-</code>
<code>naker.pyNN.models.abstract_models.AbstractPopulationInitializable</code>	<code>naker.pyNN.models.abstract_models.AbstractPopulationInitializable</code>	<code>naker.pyNN.models.abstract_models.AbstractPopulationInitializable</code>
<code>method), 38</code>	<code>class method), 210</code>	<code>class method), 210</code>
<code>get_injector_label()</code>	<code>(spyn- get_max_atoms_per_core()</code>	<code>(spyn-</code>

`naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson` (class method), 213

`naker.pyNN.models.neuron.synapse_io.synapse_io_row_base.SynapseIORowBased` (class method), 171

`get_max_atoms_per_core()` (spyn- `get_maximum_delay_supported_in_ms()` (`naker.pyNN.models.spike_source.SpikeSourcePoisson` (class method), 219

`get_max_row_info()` (spyn- `get_maximum_delay_supported_in_ms()` (`naker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO` (class method), 170

`get_max_row_info()` (spyn- `get_maximum_delay_supported_in_ms()` (`naker.pyNN.models.neuron.synapse_io.AbstractSynapseIO` (class method), 171

`get_max_row_info()` (spyn- `get_maximum_probable_value()` (in module `naker.pyNN.models.neuron.synapse_io.synapse_io_row_base.SynapseIORowBased` (class method), 171

`get_max_row_info()` (spyn- `get_mean()` (in module `spyn- naker.pyNN.utilities.utility_calls`), 251

`get_max_row_info()` (spyn- `message_translator()` (spyn- `naker.pyNN.external_devices_models.abstract_ethernet_controller.AbstractEthernetController` (class method), 172

`get_max_synapses()` (spyn- `method`), 16

`get_max_synapses()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics` (class method), 144

`get_max_synapses()` (spyn- `method`), 26

`get_max_synapses()` (spyn- `message_translator()` (spyn- `naker.pyNN.external_devices_models.external_device_lif_controller.ExternalDeviceLifController` (class method), 157

`get_max_synapses()` (spyn- `method`), 20

`get_max_synapses()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic` (class method), 146

`get_max_synapses()` (spyn- `get_munich_d()` (in module `spyn- naker.pyNN.utilities.utility_calls`), 251

`get_max_synapses()` (spyn- `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic` (class method), 148

`get_max_synapses()` (spyn- `get_munich_f()` (in module `spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic` (class method), 159

`get_max_synapses()` (spyn- `get_munich_i()` (in module `spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic` (class method), 161

`get_maximum_delay_supported_in_ms()` (`get_n_bits()` (in module `spyn- (spynnaker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses` (class method), 34

`get_maximum_delay_supported_in_ms()` (`get_n_connections()` (spyn- `naker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector` (class method), 37

`get_maximum_delay_supported_in_ms()` (`get_n_connections()` (spyn- `naker.pyNN.models.neural_projections.connectors.FromListConnector` (class method), 193

`get_maximum_delay_supported_in_ms()` (`get_n_connections_from_pre_vertex_maximum()` (spynnaker.pyNN.models.neural_projections.connectors.abstract_population_vertex.AbstractPopulationVertex (class method), 50

`get_maximum_delay_supported_in_ms()` (`get_n_connections_from_pre_vertex_maximum()` (spynnaker.pyNN.models.neuron.AbstractPopulationVertex (class method), 203

`get_maximum_delay_supported_in_ms()` (`get_n_connections_from_pre_vertex_maximum()` (spynnaker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO (class method), 170

`get_maximum_delay_supported_in_ms()` (`get_n_connections_from_pre_vertex_maximum()` (spynnaker.pyNN.models.neural_projections.connectors.all_to_all_connector.AllToAllConnector (class method), 53

`get_maximum_delay_supported_in_ms()` (`get_n_connections_from_pre_vertex_maximum()` (spynnaker.pyNN.models.neuron.synapse_io.AbstractSynapseIO (class method), 171

`get_maximum_delay_supported_in_ms()` (`get_n_connections_from_pre_vertex_maximum()` (spynnaker.pyNN.models.neural_projections.connectors.AllToAllConnector (class method), 53

Index 295

```

method), 58
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.fixed_probability_neuron.FixedProbabilityNeuronEIO SpikeRe
method), 59
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.FixedNeuronFixedConnectCommon.EIO SpikeRecorder
method), 71
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.FixedNeuronFixedConnectCommon.multi_spike_recorder.MultiSpikeRec
method), 72
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.FixedNeuronFixedConnectCommon.MultiSpikeRecorder
method), 73
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.from_list.pyNN.models.common.NeuronRecorder
method), 60
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.FrontList.pyNN.models.common.NeuronRecorder
method), 74
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.index_based_neuron.IndexBasedProbabilityConnect
method), 61
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.IndexBasedProbabilityConnectAdditionalInputs.AdditionalInputCa2
method), 75
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.kernel_copy.pyNN.models.neuron.implementations.abstract_neuron_impl
method), 63
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.KernelCopy.pyNN.models.neuron.implementations.abstract_standard_n
method), 79
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.multiplex.pyNN.models.neuron.implementations.AbstractNeuronImpl
method), 64
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.Multiplex.pyNN.models.neuron.implementations.AbstractStandardNeu
method), 76
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.open_take.pyNN.models.neuron.implementations.neuron_impl_standa
method), 64
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.OneToOne.pyNN.models.neuron.implementations.NeuronImplStandara
method), 76
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.small_copy.pyNN.models.neuron.input_types.input_type_conductance.I
method), 65
get_n_connections_to_post_vertex_maximum(get_n_cpu_cycles()
(spynaker.pyNN.models.neural_projections.connectors.SmallCopy.pyNN.models.neuron.input_types.input_type_current.InputT
method), 77
get_n_cpu_cycles()
(spynaker.pyNN.external_devices_models.threshold_type_multicast.pyNN.models.neuron.input_types.InputTypeConductance
method), 25
get_n_cpu_cycles()
(spynaker.pyNN.external_devices_models.ThresholdTypeMulticastDevice.pyNN.models.neuron.input_types.InputTypeConductance
method), 25

```


297

method), 27

get_n_neurons() (spyn- get_n_synapse_types() (spyn-
 naker.pyNN.external_devices_models.external_spinnaker_link_from_NN_model_device.ExternalFPGAResistorDevice.type_dual_exp
 static method), 21 method), 176

get_n_neurons() (spyn- get_n_synapse_types() (spyn-
 naker.pyNN.external_devices_models.ExternalFPGAResistorDevice.type_dual_exp
 static method), 29 method), 178

get_n_plastic_plastic_words_per_row() get_n_synapse_types() (spyn-
 (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractPlasticSynapseTypeDelta
 method), 143 method), 184

get_n_plastic_plastic_words_per_row() get_n_synapse_types() (spyn-
 (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractPlasticSynapseTypeDelta
 method), 159 method), 182

get_n_plastic_plastic_words_per_row() get_n_synapse_types() (spyn-
 (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseTypeDualExp
 method), 148 method), 179

get_n_plastic_plastic_words_per_row() get_n_synapse_types() (spyn-
 (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseTypeDualExp
 method), 161 method), 181

get_n_static_words_per_row() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractStaticSynapseTypeDualExp
 method), 144 method), 143

get_n_static_words_per_row() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseTypeDualExp
 method), 158 method), 144

get_n_static_words_per_row() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseTypeDualExp
 method), 146 method), 159

get_n_static_words_per_row() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.synapse_dynamics.SynapseTypeDualExp
 method), 159 method), 158

get_n_synapse_types() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.implementations.abstract_neuron_impl.StandardNeuronImpl.synapse_dynamics.synapse_dynamics
 method), 93 method), 146

get_n_synapse_types() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.implementations.AbstractNeuronImpl.synapse_dynamics.synapse_dynamics
 method), 106 method), 148

get_n_synapse_types() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.implementations.neuron_impl.StandardNeuronImpl.synapse_dynamics.synapse_dynamics
 method), 98 method), 151

get_n_synapse_types() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.implementations.NeuronImplStandardNeuronImpl.synapse_dynamics.SynapseDynamics
 method), 104 method), 159

get_n_synapse_types() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseTypeDualExp
 method), 172 method), 161

get_n_synapse_types() (spyn- get_n_synapses_in_rows() (spyn-
 naker.pyNN.models.neuron.synapse_types.AbstractSynapseTypeDualExp
 method), 179 method), 164

get_n_synapse_types() (spyn- get_n_words_for_plastic_connections() (spyn-
 naker.pyNN.models.neuron.synapse_types.synapse_type_abstract_synapse_type.AbstractSynapseTypeDualExp
 method), 173 method), 143

get_n_synapse_types() (spyn- get_n_words_for_plastic_connections() (spyn-
 naker.pyNN.models.neuron.synapse_types.synapse_type_delta_synapse_type_delta.SynapseTypeDelta
 method), 173 method), 143

method), 159	method), 118
get_n_words_for_plastic_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics. method), 148	get_outgoing_partition_constraints() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl. method), 20
get_n_words_for_plastic_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics. method), 156	get_outgoing_partition_constraints() (spynnaker.pyNN.external_devices_models.ExternalDeviceLifControl. method), 21
get_n_words_for_plastic_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 161	get_outgoing_partition_constraints() (spynnaker.pyNN.external_devices_models.ExternalFPGA. method), 29
get_n_words_for_plastic_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 169	get_outgoing_partition_constraints() (spynnaker.pyNN.external_devices_models.munich_spinnaker. method), 23
get_n_words_for_static_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics. method), 144	get_outgoing_partition_constraints() (spynnaker.pyNN.models.neuron.synapse_dynamics. method), 25
get_n_words_for_static_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics. method), 158	get_outgoing_partition_constraints() (spynnaker.pyNN.models.neuron.synapse_dynamics. method), 31
get_n_words_for_static_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics. method), 146	get_outgoing_partition_constraints() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 32
get_n_words_for_static_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamics. method), 154	get_outgoing_partition_constraints() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 193
get_n_words_for_static_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 159	get_outgoing_partition_constraints() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex. method), 204
get_n_words_for_static_connections() (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics. method), 167	get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.delay_extension. method), 216
get_neuron_sampling_interval() (spynnaker.pyNN.models.common.abstract_neuron_recordable. method), 40	get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.delays.delay_extension. method), 222
get_neuron_sampling_interval() (spynnaker.pyNN.models.common.AbstractNeuronRecordable. method), 46	get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.delays.DelayExtension. method), 225
get_neuron_sampling_interval() (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder. method), 44	get_outgoing_partition_constraints() (spynnaker.pyNN.models.utility_models.spike_injector.spike_injector. method), 227
get_neuron_sampling_interval() (spynnaker.pyNN.models.common.NeuronRecorder. method), 48	get_outgoing_partition_ids() (spynnaker.pyNN.external_devices_models.abstract_ethernet_controller. method), 16
get_neuron_sampling_interval() (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex. method), 193	get_outgoing_partition_ids() (spynnaker.pyNN.external_devices_models.AbstractEthernetController. method), 26
get_neuron_sampling_interval() (spynnaker.pyNN.models.neuron.AbstractPopulationVertex. method), 204	get_outgoing_partition_ids() (spynnaker.pyNN.external_devices_models.external_device_lif_control. method), 20
get_next_allowed_address() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators. method), 117	get_parameter_names() (spynnaker.pyNN.models.utility_models.delay_extension. class method), 229
get_next_allowed_address() (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators. method), 117	get_parameter_names() (spynnaker.pyNN.models.utility_models.delay_extension. class method), 229

`naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex`
`method), 199`

`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics`
`method), 148`

`get_profile_data()` (spyn- `get_provenance_data()` (spyn-
`naker.pyNN.models.neuron.PopulationMachineVertex` `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics`
`method), 209` `method), 162`

`get_profile_data()` (spyn- `get_provenance_data_from_machine()`
`naker.pyNN.models.spike_source.spike_source_poisson_machine.PopulationMachineVertex` (spyn-
`method), 213` `method), 199`

`get_projections_data()` (spyn- `get_provenance_data_from_machine()`
`naker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon` (spyn-
`method), 253` `method), 209`

`get_provenance_data()` (spyn- `get_provenance_data_from_machine()`
`naker.pyNN.models.neural_projections.connectors.abstract_synapse_dynamics.models.delays.delay_extension_m` (spyn-
`method), 51` `method), 220`

`get_provenance_data()` (spyn- `get_provenance_data_from_machine()`
`naker.pyNN.models.neural_projections.connectors.AbstractSynapseDynamics` (spyn-
`method), 66` `method), 223`

`get_provenance_data()` (spyn- `get_push_bot_laser_led_speaker_frequency_i()`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence.AbstractTimingDependence` (spyn-
`method), 129` `naker.pyNN.protocols.munich_io_spinnaker_link_protocol),`
`239`

`get_provenance_data()` (spyn- `239`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence.AbstractTimingDependence` (spyn-
`method), 133` `naker.pyNN.protocols.munich_io_spinnaker_link_protocol),`
`239`

`get_provenance_data()` (spyn- `239`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence.AbstractTimingDependence` (spyn-
`method), 130` `in module spyn-`
`(in module spyn-`

`get_provenance_data()` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair.TimingDependenceSpike`
`method), 132` `get_pynn_NumpyRNG()` (spyn-
`naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface`

`get_provenance_data()` (spyn- `naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair.TimingDependenceSpikePair`
`method), 132` `get_pynn_NumpyRNG()` (spyn-
`naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface`

`get_provenance_data()` (spyn- `naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_triplet.TimingDependenceSpikeTriplet`
`method), 135` `get_random_distribution()` (spyn-
`naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface`

`get_provenance_data()` (spyn- `naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_nearest_pair.TimingDependenceSpikeNearestPair`
`method), 136` `get_random_distribution()` (spyn-
`naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface`

`get_provenance_data()` (spyn- `naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState`
`naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing_dependence_spike_pair.TimingDependenceSpikePair`
`method), 134` `get_receive_method()` (spyn-
`naker.pyNN.external_devices_models.push_bot.push_bot_etherne`

`get_provenance_data()` (spyn- `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.timing_dependence.AbstractWeightDependence`
`method), 137` `get_recordable_units()` (spyn-
`naker.pyNN.models.neuron.implementations.abstract_neuron_impl`

`get_provenance_data()` (spyn- `naker.pyNN.models.neuron.implementations.abstract_neuron_impl`
`naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.timing_dependence.AbstractWeightDependence`
`method), 140` `get_recordable_units()` (spyn-
`naker.pyNN.models.neuron.implementations.AbstractNeuronImpl`

`get_provenance_data()` (spyn- `naker.pyNN.models.neuron.implementations.AbstractNeuronImpl`
`naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics`
`method), 145` `get_recordable_units()` (spyn-
`naker.pyNN.models.neuron.implementations.neuron_impl_stand`

`get_provenance_data()` (spyn- `naker.pyNN.models.neuron.implementations.neuron_impl_stand`
`naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics`
`method), 157` `get_recordable_units()` (spyn-
`naker.pyNN.models.neuron.implementations.NeuronImplStandard`

method), 104

get_recordable_variable_index() (spyn- naker.pyNN.models.neuron.implementations.abstract_neuron_impl_standard), 94

get_recordable_variable_index() (spyn- naker.pyNN.models.neuron.implementations.AbstractNeuronImplStandard), 106

get_recordable_variable_index() (spyn- naker.pyNN.models.neuron.implementations.neuron_impl_standard), 98

get_recordable_variable_index() (spyn- naker.pyNN.models.neuron.implementations.NeuronImplStandard), 104

get_recordable_variables() (spyn- naker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable), 41

get_recordable_variables() (spyn- naker.pyNN.models.common.AbstractNeuronRecordable), 46

get_recordable_variables() (spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder), 44

get_recordable_variables() (spyn- naker.pyNN.models.common.NeuronRecorder), 49

get_recordable_variables() (spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex), 193

get_recordable_variables() (spyn- naker.pyNN.models.neuron.AbstractPopulationVertex), 204

get_recordable_variables() (spyn- naker.pyNN.models.neuron.implementations.abstract_neuron_impl_standard), 94

get_recordable_variables() (spyn- naker.pyNN.models.neuron.implementations.AbstractNeuronImplStandard), 106

get_recordable_variables() (spyn- naker.pyNN.models.neuron.implementations.neuron_impl_standard), 98

get_recordable_variables() (spyn- naker.pyNN.models.neuron.implementations.NeuronImplStandard), 104

get_recorded_region_ids() (spyn- naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex), 199

get_recorded_region_ids() (spyn- naker.pyNN.models.neuron.PopulationMachineVertex), 209

get_recorded_region_ids() (spyn- naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex), 214

get_recording_region_base_address() (spyn- naker.pyNN.models.neuron.PopulationMachineVertex), 199

get_recording_region_base_address() (spyn- naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex), 214

get_recording_region_size_in_bytes() (in module naker.pyNN.models.common), 50

get_recording_region_size_in_bytes() (in module naker.pyNN.models.common.recording_utils), 44

get_recording_sdram_usage() (spyn- naker.pyNN.models.common.recording_utils), 216

get_resources_used_by_atoms() (spyn- naker.pyNN.external_devices_models.munich_spinnaker_link_protocol), 23

get_resources_used_by_atoms() (spyn- naker.pyNN.external_devices_models.MunichMotorDevice), 31

get_resources_used_by_atoms() (spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex), 193

get_resources_used_by_atoms() (spyn- naker.pyNN.models.neuron.AbstractPopulationVertex), 204

get_resources_used_by_atoms() (spyn- naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex), 216

get_resources_used_by_atoms() (spyn- naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex), 222

get_resources_used_by_atoms() (spyn- naker.pyNN.models.utility_models.delays.DelayExtensionVertex), 225

get_retina_i() (in module naker.pyNN.protocols.munich_io_spinnaker_link_protocol), 239

GET_RETINA_KEY_VALUE() (in module spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol), 235

GET_RETINA_PAYLOAD_VALUE() (in module spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol), 235

get_rng_next() (spyn- naker.pyNN.neural_projections.connectors.multipse_connector.MultapseConnector), 64

get_rng_next() (spyn- naker.pyNN.neural_projections.connectors.MultapseConnector), 76

get_sampling_overflow_sdram() (spyn- naker.pyNN.models.neuron_recorder.NeuronRecorder), 214

method), 44	method), 102
get_sampling_overflow_sdram() (spynaker.pyNN.models.common.NeuronRecorder method), 49	get_size_in_whole_words() (spynaker.pyNN.models.neuron.implementations.struct.Struct method), 99
get_sdram_usage_for_atoms() (spynaker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex method), 222	get_spike_counts() (spynaker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex method), 231
get_sdram_usage_for_atoms() (spynaker.pyNN.models.utility_models.delays.DelayExtensionVertex method), 225	get_spike_value_from_fpga_retina() (spynaker.pyNN.external_devices_models.external_spinnaker_link_fpga module), 22
get_sdram_usage_in_bytes() (spynaker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder method), 42	get_spike_value_from_robot_retina() (spynaker.pyNN.external_devices_models.munich_spinnaker_link_robot module), 25
get_sdram_usage_in_bytes() (spynaker.pyNN.models.common.MultiSpikeRecorder method), 49	get_spikes() (spynaker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable method), 41
get_sdram_usage_in_bytes() (spynaker.pyNN.models.common.neuron_recorder.NeuronRecorder method), 44	get_spikes() (spynaker.pyNN.models.common.AbstractSpikeRecordable method), 46
get_sdram_usage_in_bytes() (spynaker.pyNN.models.common.NeuronRecorder method), 49	get_spikes() (spynaker.pyNN.models.common.eieio_spike_recorder.EIEIOSpikeRecorder method), 43
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.implementations.abstract_neuron_model.AbstractNeuronModel method), 94	get_spikes() (spynaker.pyNN.models.common.EIEIOSpikeRecorder method), 47
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent method), 95	get_spikes() (spynaker.pyNN.models.common.multi_spike_recorder.MultiSpikeRecorder method), 42
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.implementations.AbstractNeuronImpl method), 106	get_spikes() (spynaker.pyNN.models.common.MultiSpikeRecorder method), 101
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.implementations.AbstractStandardNeuronComponent method), 101	get_spikes() (spynaker.pyNN.models.common.neuron_recorder.NeuronRecorder method), 98
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.implementations.neuron_impl_standard.NeuronImplStandard method), 98	get_spikes() (spynaker.pyNN.models.common.NeuronRecorder method), 104
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.implementations.NeuronImplStandard method), 104	get_spikes() (spynaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 120
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.neuron_models.abstract_neuron_model.AbstractNeuronModel method), 120	get_spikes() (spynaker.pyNN.models.neuron.AbstractPopulationVertex method), 204
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.neuron_models.AbstractNeuronModel method), 124	get_spikes() (spynaker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex method), 211
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.synaptic_manager.SynapticManager method), 200	get_spikes() (spynaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex method), 216
get_sdram_usage_in_bytes() (spynaker.pyNN.models.neuron.SynapticManager method), 208	get_spikes() (spynaker.pyNN.models.utility_models.spike_injector.spike_injector_vertex.SpikeInjectorVertex method), 227
get_size_in_whole_words() (spynaker.pyNN.models.neuron.implementations.Struct method), 102	

`get_spikes_sampling_interval()` (spyn- `naker.pyNN.models.neuron.implementations.abstract_neuron_im`
`naker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable`
`method`), 41 `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.implementations.AbstractNeuronImpl`
`method`), 106
`get_spikes_sampling_interval()` (spyn- `naker.pyNN.models.neuron.implementations.neuron_impl_standa`
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method`), 193 `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.implementations.NeuronImplStandard`
`method`), 104
`get_spikes_sampling_interval()` (spyn- `naker.pyNN.models.neuron.synapse_types.abstract_synapse_type`
`naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex`
`method`), 211 `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_types.AbstractSynapseType`
`naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex`
`method`), 216 `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.SynapseTypeAlpha`
`naker.pyNN.models.utility_models.spike_injector.spike_injector_vertex.SpikeInjectorVertex`
`method`), 227 `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta`
`get_standard_deviation()` (in module spyn- `naker.pyNN.utilities.utility_calls`), 251 `method`), 175
`get_static_synaptic_data()` (spyn- `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.abstract_static_synapse_dynamics.AbstractStaticSynapseDynamicsTypeDualExp`
`method`), 144 `method`), 176
`get_static_synaptic_data()` (spyn- `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.AbstractStaticSynapseDynamicsTypeExponential`
`method`), 158 `method`), 178
`get_static_synaptic_data()` (spyn- `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_synapse_types.SynapseTypeAlpha`
`method`), 146 `method`), 184
`get_static_synaptic_data()` (spyn- `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_synapse_types.SynapseTypeDelta`
`method`), 154 `method`), 182
`get_static_synaptic_data()` (spyn- `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsTypeDualExponential`
`method`), 160 `method`), 180
`get_static_synaptic_data()` (spyn- `get_synapse_id_by_target()` (spyn-
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsTypeExponential`
`method`), 167 `method`), 181
`get_synapse_id_by_target` (spyn- `get_synapse_targets()` (spyn-
`naker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses`
`attribute`), 34 `method`), 94
`get_synapse_id_by_target` (spyn- `get_synapse_targets()` (spyn-
`naker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses`
`attribute`), 37 `method`), 106
`get_synapse_id_by_target()` (spyn- `get_synapse_targets()` (spyn-
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex`
`method`), 193 `method`), 98
`get_synapse_id_by_target()` (spyn- `get_synapse_targets()` (spyn-
`naker.pyNN.models.neuron.AbstractPopulationVertex` `naker.pyNN.models.neuron.implementations.NeuronImplStandard`
`method`), 204 `method`), 104
`get_synapse_id_by_target()` (spyn- `get_synapse_targets()` (spyn-

[naker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType](#)
 method), 172

[naker.pyNN.models.neuron.synapse_types.AbstractSynapseType](#)
 method), 179

[naker.pyNN.models.neuron.synapse_types.synapse_type_alpha_synapse_type.AlphaSynapseType](#)
 method), 173

[naker.pyNN.models.neuron.synapse_types.synapse_type_delta_synapse_type.DeltaSynapseType](#)
 method), 175

[naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exp_synapse_type.DualExpSynapseType](#)
 method), 176

[naker.pyNN.models.neuron.synapse_types.synapse_type_exponential_synapse_type.ExponentialSynapseType](#)
 method), 178

[naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha](#)
 method), 184

[naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta](#)
 method), 183

[naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExp](#)
 method), 180

[naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential](#)
 method), 181

[naker.pyNN.models.neuron.synapse_io.abstract_synapse_io.AbstractSynapseIO](#)
 method), 170

[naker.pyNN.models.neuron.synapse_io.AbstractSynapseIO](#)
 method), 171

[naker.pyNN.models.neuron.synapse_io.synapse_io_row_based_synapse_io.RowBasedSynapseIO](#)
 method), 171

[naker.pyNN.models.neuron.synapse_io.SynapseIORowBased](#)
 method), 172

[naker.pyNN.external_devices_models.abstract_ethernet_sensor.AbstractEthernetSensor](#)
 method), 17

[naker.pyNN.external_devices_models.AbstractEthernetSensor](#)
 method), 27

[naker.pyNN.external_devices_models.threshold_type_multicausal_type.MulticausalThresholdType](#)
 method), 25

[naker.pyNN.external_devices_models.ThresholdTypeMulticausal](#)
 method), 33

[naker.pyNN.models.neuron.synapse_types.abstract_synapse_type.AbstractSynapseType](#)
 method), 34

[naker.pyNN.models.abstract_models.AbstractContainsUnits](#)
 method), 38

[naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex](#)
 method), 194

[naker.pyNN.models.neuron.AbstractPopulationVertex](#)
 method), 204

[naker.pyNN.models.neuron.synapse_types.DualExpSynapseType](#)
 method), 86

[naker.pyNN.models.neuron.synapse_types.ExponentialSynapseType](#)
 method), 88

[naker.pyNN.models.neuron.implementations.abstract_neuron_impl.AbstractNeuronImpl](#)
 method), 94

[naker.pyNN.models.neuron.implementations.abstract_standard_neuron_impl.AbstractStandardNeuronImpl](#)
 method), 96

[naker.pyNN.models.neuron.implementations.AbstractNeuronImpl](#)
 method), 106

[naker.pyNN.models.neuron.implementations.AbstractStandardNeuronImpl](#)
 method), 101

[naker.pyNN.models.neuron.implementations.neuron_impl_standard_neuron_impl.NeuronImplStandardNeuronImpl](#)
 method), 104

[naker.pyNN.models.neuron.input_types.input_type_conductance.InputTypeConductance](#)
 method), 108

[naker.pyNN.models.neuron.input_types.input_type_current.InputTypeCurrent](#)
 method), 109

[naker.pyNN.models.neuron.input_types.input_type_current_semaphore.InputTypeCurrentSemaphore](#)
 method), 110

[naker.pyNN.models.neuron.input_types.InputTypeConductance](#)
 method), 112

[naker.pyNN.models.neuron.input_types.InputTypeCurrent](#)
 method), 113

[naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD](#)
 method), 114

Index	307
--------------	------------

`method)`, 157
`get_weight_maximum()` (`spyn-` `get_words()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.STDP_dynamics.abstract_synapse_dynamics`
`method)`, 148 `method)`, 145
`get_weight_maximum()` (`spyn-` `get_words()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics.STDP_dynamics.AbstractSynapseDynamics`
`method)`, 162 `method)`, 157
`get_weight_mean()` (`spyn-` `get_x_from_fpga_retina()` (in module `spyn-`
`naker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector` `naker.pyNN.models.external_spinnaker_link_fpga`
`method)`, 51 22
`get_weight_mean()` (`spyn-` `get_x_from_robot_retina()` (in module `spyn-`
`naker.pyNN.models.neural_projections.connectors.AbstractConnector` `naker.pyNN.models.external_devices_models.munich_spinnaker_link_robot`
`method)`, 66 25
`get_weight_mean()` (`spyn-` `get_y_from_fpga_retina()` (in module `spyn-`
`naker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector` `naker.pyNN.models.external_spinnaker_link_fpga`
`method)`, 60 22
`get_weight_mean()` (`spyn-` `get_y_from_robot_retina()` (in module `spyn-`
`naker.pyNN.models.neural_projections.connectors.FromListConnector` `naker.pyNN.models.external_devices_models.munich_spinnaker_link_robot`
`method)`, 74 25
`get_weight_mean()` (`spyn-` `global_struct` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.STDP_dynamics.abstract_neuron_model`
`method)`, 145 `attribute)`, 120
`get_weight_mean()` (`spyn-` `global_struct` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics` `naker.pyNN.models.neuron.neuron_models.AbstractNeuronModel`
`method)`, 157 `attribute)`, 124
`get_weight_mean()` (`spyn-` `GraphEdgeFilter` (class in `spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.STDP_dynamics.functions.graph_edge_filter)`,
`method)`, 149 233
`get_weight_mean()` (`spyn-` `GraphEdgeWeightUpdater` (class in `spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics.STDP_dynamics.overridepacman_functions.graph_edge_weight_u`
`method)`, 162 234
`get_weight_variance()` (`spyn-` `grid()` (in module `spynaker.plot_utils`), 258
`naker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector`
`method)`, 51 **H**
`get_weight_variance()` (`spyn-` `has_parameter()` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.AbstractConnector` `naker.pyNN.models.abstract_pynn_model.AbstractPyNNModel`
`method)`, 66 `class method)`, 229
`get_weight_variance()` (`spyn-` `has_reset_last` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.from_list_connector.FromListConnector` `naker.pyNN.spinnaker_simulator_interface.SpinnakerSimulatorInterface`
`method)`, 60 `attribute)`, 257
`get_weight_variance()` (`spyn-` `has_reset_last` (`spyn-`
`naker.pyNN.models.neural_projections.connectors.FromListConnector` `naker.pyNN.utilities.spynaker_failed_state.SpinnakerFailedState`
`method)`, 74 `attribute)`, 249
`get_weight_variance()` (`spyn-` `has_variable()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics` `naker.pyNN.models.external_devices_models.threshold_type_multicast_dev`
`method)`, 145 `method)`, 26
`get_weight_variance()` (`spyn-` `has_variable()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.AbstractSynapseDynamics` `naker.pyNN.models.external_devices_models.ThresholdTypeMulticastDev`
`method)`, 157 `method)`, 33
`get_weight_variance()` (`spyn-` `has_variable()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.STDP_dynamics.functions.additional_input_callback`
`method)`, 149 `method)`, 87
`get_weight_variance()` (`spyn-` `has_variable()` (`spyn-`
`naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics.STDP_dynamics.functions.additional_input_callback`
`naker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2`

`method`), 88
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.implementations.abstract_standard_neuron_model`, 96
`method`), 96
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.implementations.AbstractStandardNeuronModel`, 101
`method`), 101
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.input_type_conductance`, 108
`method`), 108
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.input_type_current`, 109
`method`), 109
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.input_type_current`, 110
`method`), 110
`has_variable()` (spyn- `has_variable()` (spyn- `has_variable()` (spyn-
`naker.pyNN.models.neuron.input_types.InputTypeConductance`, 112
`method`), 112
`has_variable()` (spyn- `heat_plot()` (in module `spynaker.plot_utils`), 258
`naker.pyNN.models.neuron.input_types.InputTypeCurrent`, 113
`method`), 113
`has_variable()` (spyn- `HHCondExp` (class in spyn-
`naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD`, 114
`method`), 114
`has_variable()` (spyn- `high()` (in module spyn-
`naker.pyNN.models.neuron.neuron_models.neuron_model`, 121
`method`), 121
`has_variable()` (spyn- `high()` (spynaker.pyNN.utilities.random_stats.abstract_random_stats.Ab-
`naker.pyNN.models.neuron.neuron_models.neuron_model`, 123
`method`), 123
`has_variable()` (spyn- `high()` (spynaker.pyNN.utilities.random_stats.abstract_random_stats.Ab-
`naker.pyNN.models.neuron.neuron_models.NeuronModelIzh`, 125
`method`), 125
`has_variable()` (spyn- `i_alpha` (spynaker.pyNN.models.neuron.additional_inputs.additional_in-
`naker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegrateAndFire`, 126
`method`), 126
`has_variable()` (spyn- `i_ca2` (spynaker.pyNN.models.neuron.additional_inputs.additional_inpu-
`naker.pyNN.models.neuron.synapse_types.synapse_type_alpha`, 174
`method`), 174
`has_variable()` (spyn- `i_ca2` (spynaker.pyNN.models.neuron.additional_inputs.AdditionalInput-
`naker.pyNN.models.neuron.synapse_types.synapse_type_delta`, 175
`method`), 175
`has_variable()` (spyn- `i_offset` (spynaker.pyNN.models.neuron.neuron_models.neuron_model-
`naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential`, 177
`method`), 177
`has_variable()` (spyn- `i_offset` (spynaker.pyNN.models.neuron.neuron_models.NeuronModel-
`naker.pyNN.models.neuron.synapse_types.synapse_type_exponential`, 178
`method`), 178
`has_variable()` (spyn- `id_counter` (spynaker.pyNN.abstract_spinnaker_common.AbstractSpiN-
`naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha`, 184
`method`), 184
`has_variable()` (spyn- `id_to_index()` (spyn-
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta`, 231
`method`), 231

<code>id_to_local_index()</code>	(<i>spyn-</i> <i>index_expression</i> <i>(spyn-</i>	<i>naker.pyNN.models.pynn_population_common.PyNNPopulationCommon</i> <i>naker.pyNN.models.neural_projections.connectors.index_based_p</i>
<i>method</i>), 231	<i>attribute</i>), 61	
<code>IFCondAlpha</code> (class in <i>spyn-</i> <i>index_expression</i> <i>(spyn-</i>	<i>naker.pyNN.models.neuron.builds</i>), 91	<i>naker.pyNN.models.neural_projections.connectors.IndexBasedPr</i>
<code>IFCondAlpha</code> (class in <i>spyn-</i> <i>attribute</i>), 75	<i>naker.pyNN.models.neuron.builds.if_cond_alpha</i>), <code>index_to_id()</code>	<i>(spyn-</i>
89	<i>naker.pyNN.models.pynn_population_common.PyNNPopulationC</i>	
<code>IFCondExpBase</code> (class in <i>spyn-</i> <i>method</i>), 231	<i>naker.pyNN.models.neuron.builds</i>), 91	<code>IndexBasedProbabilityConnector</code>
<code>IFCondExpBase</code> (class in <i>spyn-</i> (class in <i>spyn-</i>	<i>naker.pyNN.models.neuron.builds.if_cond_exp_base</i>),	<i>naker.pyNN.models.neural_projections.connectors</i>),
89	74	
<code>IFCondExpStoc</code> (class in <i>spyn-</i> <code>IndexBasedProbabilityConnector</code>	<i>naker.pyNN.models.neuron.builds</i>), 92	(class in <i>spyn-</i>
<code>IFCondExpStoc</code> (class in <i>spyn-</i> <i>naker.pyNN.models.neural_projections.connectors.index_based_p</i>	<i>naker.pyNN.models.neuron.builds.if_cond_exp_stoc</i>),	60
89	<code>inh_input_previous</code> <i>(spyn-</i>	
<code>IFCurrAlpha</code> (class in <i>spyn-</i> <i>naker.pyNN.models.neuron.input_types.input_type_current_semd</i>	<i>naker.pyNN.models.neuron.builds</i>), 92	<i>attribute</i>), 111
<code>IFCurrAlpha</code> (class in <i>spyn-</i> <code>inh_input_previous</code> <i>(spyn-</i>	<i>naker.pyNN.models.neuron.builds.if_curr_alpha</i>),	<i>naker.pyNN.models.neuron.input_types.InputTypeCurrentSEMD</i>
90	<i>attribute</i>), 114	
<code>IFCurrDelta</code> (class in <i>spyn-</i> <code>inh_response</code> <i>(spyn-</i>	<i>naker.pyNN.models.neuron.builds</i>), 92	<i>naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.Sy</i>
<code>IFCurrDelta</code> (class in <i>spyn-</i> <i>attribute</i>), 174	<i>naker.pyNN.models.neuron.builds.if_curr_delta</i>), <code>inh_response</code>	<i>(spyn-</i>
90	<i>naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha</i>	
<code>IFCurrDualExpBase</code> (class in <i>spyn-</i> <i>attribute</i>), 184	<i>naker.pyNN.models.neuron.builds</i>), 92	<code>initial_values</code> <i>(spyn-</i>
<code>IFCurrDualExpBase</code> (class in <i>spyn-</i> <i>naker.pyNN.models.abstract_models.abstract_population_initiali</i>	<i>naker.pyNN.models.neuron.builds.if_curr_dual_exp_base</i>), <i>attribute</i>), 35	
90	<code>initial_values</code> <i>(spyn-</i>	
<code>IFCurrExpBase</code> (class in <i>spyn-</i> <i>naker.pyNN.models.abstract_models.AbstractPopulationInitializa</i>	<i>naker.pyNN.models.neuron.builds</i>), 92	<i>attribute</i>), 38
<code>IFCurrExpBase</code> (class in <i>spyn-</i> <code>initialise_table()</code> <i>(spyn-</i>	<i>naker.pyNN.models.neuron.builds.if_curr_exp_base</i>),	<i>naker.pyNN.models.neuron.master_pop_table_generators.master</i>
90	<i>method</i>), 117	
<code>IFCurrExpCa2Adaptive</code> (class in <i>spyn-</i> <code>initialise_table()</code> <i>(spyn-</i>	<i>naker.pyNN.models.neuron.builds</i>), 92	<i>naker.pyNN.models.neuron.master_pop_table_generators.Master</i>
<code>IFCurrExpCa2Adaptive</code> (class in <i>spyn-</i> <i>method</i>), 119	<i>naker.pyNN.models.neuron.builds.if_curr_exp_ca2_adaptive</i>), <code>initialize()</code>	<i>(spyn-</i>
90	<i>naker.pyNN.models.abstract_models.abstract_population_initiali</i>	
<code>IFCurrExpSEMDBase</code> (class in <i>spyn-</i> <i>method</i>), 35	<i>naker.pyNN.models.neuron.builds</i>), 92	<code>initialize()</code> <i>(spyn-</i>
<code>IFCurrExpSEMDBase</code> (class in <i>spyn-</i> <i>naker.pyNN.models.abstract_models.AbstractPopulationInitializa</i>	<i>naker.pyNN.models.neuron.builds.if_curr_exp_semd_base</i>), <i>method</i>), 38	
90	<code>initialize()</code> <i>(spyn-</i>	
<code>IFFacetsConductancePopulation</code> (class in <i>naker.pyNN.models.neuron.abstract_population_vertex.AbstractP</i>	<i>spynnaker.pyNN.models.neuron.builds</i>), 92	<i>method</i>), 194
<code>IFFacetsConductancePopulation</code> (class in <i>spyn-</i> <i>naker.pyNN.models.neuron.AbstractPopulationVertex</i>	<i>naker.pyNN.models.neuron.builds.if_facets_hardwareI</i>),	<i>method</i>), 205
91	<code>initialize_parameters</code> <i>(spyn-</i>	

[naker.pyNN.models.abstract_models.abstract_population_initialize_parameters](#) (spyn- [naker.pyNN.models.abstract_models.abstract_population_initialize_parameters](#) attribute), 35
[naker.pyNN.models.abstract_models.abstract_population_initialize_parameters](#) (spyn- [naker.pyNN.models.abstract_models.abstract_population_initialize_parameters](#) attribute), 38
[naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex](#) (spyn- [naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex](#) attribute), 194
[naker.pyNN.models.neuron.AbstractPopulationVertex](#) (spyn- [naker.pyNN.models.neuron.AbstractPopulationVertex](#) attribute), 205
[naker.pyNN.models.pynn_population_common.PyNNPopulationCommon](#) (spyn- [naker.pyNN.models.pynn_population_common.PyNNPopulationCommon](#) method), 231
[naker.pyNN.external_devices_models.push_bot.push_bot_etherne](#) (spyn- [naker.pyNN.external_devices_models.push_bot.push_bot_etherne](#) method), 10
[naker.pyNN.utilities.ranged.spynnaker_ranged_list.SpynnakerRangedList](#) (spyn- [naker.pyNN.utilities.ranged.spynnaker_ranged_list.SpynnakerRangedList](#) static method), 246
[naker.pyNN.utilities.ranged.SpynnakerRangedList](#) (spyn- [naker.pyNN.utilities.ranged.SpynnakerRangedList](#) is_list() method), 247
[naker.pyNN.external_devices_models.push_bot.push_bot_etherne](#) (spyn- [naker.pyNN.external_devices_models.push_bot.push_bot_etherne](#) method), 10
[naker.pyNN.models.neuron.implementations.abstract_neuron_imp](#) (spyn- [naker.pyNN.models.neuron.implementations.abstract_neuron_imp](#) method), 94
[naker.pyNN.models.neuron.implementations.AbstractNeuronImpl](#) (spyn- [naker.pyNN.models.neuron.implementations.AbstractNeuronImpl](#) method), 106
[naker.pyNN.models.neuron.implementations.neuron_impl_standa](#) (spyn- [naker.pyNN.models.neuron.implementations.neuron_impl_standa](#) method), 98
[naker.pyNN.models.neuron.implementations.NeuronImplStandard](#) (spyn- [naker.pyNN.models.neuron.implementations.NeuronImplStandard](#) method), 104
[naker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable](#) (spyn- [naker.pyNN.models.common.abstract_neuron_recordable.AbstractNeuronRecordable](#) method), 41
[naker.pyNN.models.common.AbstractNeuronRecordable](#) (spyn- [naker.pyNN.models.common.AbstractNeuronRecordable](#) method), 41
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol](#) (spyn- [naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol](#) attribute), 236
[naker.pyNN.models.common.neuron_recorder.NeuronRecorder](#) (spyn- [naker.pyNN.models.common.neuron_recorder.NeuronRecorder](#) method), 44
[naker.pyNN.models.common.NeuronRecorder](#) (spyn- [naker.pyNN.models.common.NeuronRecorder](#) method), 49
[naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface](#) (spyn- [naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface](#) method), 257
[naker.pyNN.models.neuron.abstract_population_vertex.AbstractP](#) (spyn- [naker.pyNN.models.neuron.abstract_population_vertex.AbstractP](#) method), 194
[naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState](#) (spyn- [naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState](#) method), 250
[naker.pyNN.models.neuron.AbstractPopulationVertex](#) (spyn- [naker.pyNN.models.neuron.AbstractPopulationVertex](#) method), 205
[naker.pyNN.models.neuron.implementations.abstract_neuron_imp](#) (spyn- [naker.pyNN.models.neuron.implementations.abstract_neuron_imp](#) attribute), 94
[naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex](#) (spyn- [naker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex](#) method), 199
[naker.pyNN.models.neuron.PopulationMachineVertex](#) (spyn- [naker.pyNN.models.neuron.PopulationMachineVertex](#) method), 209
[naker.pyNN.models.neuron.implementations.neuron_impl_standard.NeuronImplStandard](#) (spyn- [naker.pyNN.models.neuron.implementations.neuron_impl_standard.NeuronImplStandard](#) attribute), 98
[naker.pyNN.models.spike_source.spike_source_poisson_machine](#) (spyn- [naker.pyNN.models.spike_source.spike_source_poisson_machine](#) method), 214

is_recording_spikes()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.common.abstract_spike_recordable.AbstractSpikeRecordable	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.	TimingDependenceWe
method), 41	method), 137	
is_recording_spikes()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.common.AbstractSpikeRecordable	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abs	AbsWeightDependenceWe
method), 47	method), 138	
is_recording_spikes()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.Abs	AbsWeightDependenceWe
method), 194	method), 140	
is_recording_spikes()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.AbstractPopulationVertex	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.we	WeWeightDependenceWe
method), 205	method), 138	
is_recording_spikes()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.we	WeWeightDependenceWe
method), 211	method), 139	
is_recording_spikes()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.we	WeWeightDependenceWe
method), 216	method), 139	
is_recording_spikes()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.utility_models.spike_injector.spike_injector_model.SpikeInjectorModel	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.We	WeWeightDependenceWe
method), 227	method), 141	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbsoluteTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Abs	AbsoluteTimingDependenceWe
method), 129	method), 142	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbsoluteTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.We	WeWeightDependenceWe
method), 134	method), 141	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.ActiveTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Active	ActiveTimingDependenceWe
method), 130	method), 145	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 131	method), 157	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 132	method), 147	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 132	method), 149	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 133	method), 151	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 135	method), 154	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 135	method), 156	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 136	method), 160	
is_same_as()	(spyn- is_same_as())	(spyn-
naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AsyncTimingDependence	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Async	AsyncTimingDependenceWe
method), 134	method), 162	

is_same_as() (spyn- K
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon (spyn-
method), 165 KERNEL_CONNECTOR

is_same_as() (spyn- naker.pyNN.models.neural_projections.connectors.abstract_gener
attribute), 52
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic(class in spyn-
method), 167 KERNEL_CONNECTOR

is_same_as() (spyn- 77
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralSTDP(class in spyn-
method), 169 KERNEL_CONNECTOR

isyn_exc(spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta
attribute), 175

isyn_exc(spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential
attribute), 177

isyn_exc(spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential
attribute), 178 label(spynnaker.pyNN.models.pynn_population_common.PyNNPopulation
attribute), 231

isyn_exc(spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta LASER_ACTIVE_TIME (spyn-
attribute), 183 naker.pyNN.external_devices_models.push_bot.push_bot_paramete
attribute), 11

isyn_exc(spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential (spyn-
attribute), 180 LASER_ACTIVE_TIME

isyn_exc(spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential
attribute), 182 naker.pyNN.external_devices_models.push_bot.push_bot_paramete
attribute), 15

isyn_exc2(spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential
attribute), 177 laser_active_time() (spyn-
naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEth
static method), 234

isyn_exc2(spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential (spyn-
attribute), 180 laser_active_time()

isyn_inh(spynnaker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta
attribute), 175 static method), 234

isyn_inh(spynnaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential
attribute), 177 LASER_FREQUENCY (spyn-
naker.pyNN.external_devices_models.push_bot.push_bot_paramete
attribute), 11

isyn_inh(spynnaker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential (spyn-
attribute), 178 LASER_FREQUENCY

isyn_inh(spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDelta
attribute), 183 naker.pyNN.external_devices_models.push_bot.push_bot_paramete
attribute), 13

isyn_inh(spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential
attribute), 180 laser_frequency() (spyn-
naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEth
static method), 234

isyn_inh(spynnaker.pyNN.models.neuron.synapse_types.SynapseTypeExponential (spyn-
attribute), 182 laser_frequency()

iterator_by_slice() (spyn- static method), 239
naker.pyNN.models.neural_properties.neuron_parameter.NeuronParameter (spyn-
method), 85 LASER_TOTAL_PERIOD

iterator_by_slice() (spyn- naker.pyNN.external_devices_models.push_bot.push_bot_paramete
attribute), 11
naker.pyNN.models.neural_properties.NeuronParameter LASER_TOTAL_PERIOD (spyn-
method), 85

IzkCondExpBase (class in spyn- naker.pyNN.external_devices_models.push_bot.push_bot_paramete
naker.pyNN.models.neuron.builds), 92 attribute), 13

IzkCondExpBase (class in spyn- laser_total_period() (spyn-
naker.pyNN.models.neuron.builds.izk_cond_exp_base), naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEth
static method), 234
91

IzkCurrExpBase (class in spyn- laser_total_period() (spyn-
naker.pyNN.models.neuron.builds), 92 naker.pyNN.protocols.MunichIoEthernetProtocol
static method), 239

IzkCurrExpBase (class in spyn- last_id(spynnaker.pyNN.models.pynn_population_common.PyNNPopul
naker.pyNN.models.neuron.builds.izk_curr_exp_base), attribute), 231
91

LED_BACK_ACTIVE_TIME (spyn-
naker.pyNN.external_devices_models.push_bot.push_bot_paramete

<code>attribute)</code> , 11	<code>attribute)</code> , 24
<code>LED_BACK_ACTIVE_TIME</code> (<code>spyn-</code> <code>LEFT_RETINA_DISABLE</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.push_bot.push_bot_push_bot_led.PushBotLED</code> (<code>naker.pyNN.external_devices_models.munich_retina_device</code> (<code>attribute)</code> , 13	<code>attribute)</code> , 32
<code>led_back_active_time()</code> (<code>spyn-</code> <code>LEFT_RETINA_ENABLE</code> (<code>spyn-</code> <code>naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol</code> (<code>naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device</code> (<code>static method)</code> , 234	<code>attribute)</code> , 24
<code>led_back_active_time()</code> (<code>spyn-</code> <code>LEFT_RETINA_ENABLE</code> (<code>spyn-</code> <code>naker.pyNN.protocols.MunichIoEthernetProtocol</code> (<code>naker.pyNN.external_devices_models.MunichRetinaDevice</code> (<code>static method)</code> , 239	<code>attribute)</code> , 32
<code>LED_FREQUENCY</code> (<code>spyn-</code> <code>LEFT_RETINA_KEY_SET</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.push_bot.push_bot_push_bot_led.PushBotLED</code> (<code>naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device</code> (<code>attribute)</code> , 11	<code>attribute)</code> , 24
<code>LED_FREQUENCY</code> (<code>spyn-</code> <code>LEFT_RETINA_KEY_SET</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.push_bot.push_bot_push_bot_led.PushBotLED</code> (<code>naker.pyNN.external_devices_models.MunichRetinaDevice</code> (<code>attribute)</code> , 13	<code>attribute)</code> , 32
<code>led_frequency()</code> (<code>spyn-</code> <code>line_plot()</code> (in module <code>spynaker.plot_utils</code>), 258	
<code>naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol</code> (<code>spyn-</code> <code>static method)</code> , 234	<code>naker.pyNN.utilities.ranged.spynaker_ranged_dict.SpynakerRangeDictionary</code> (<code>method)</code> , 245
<code>led_frequency()</code> (<code>spyn-</code> <code>naker.pyNN.protocols.MunichIoEthernetProtocol</code> (<code>static method)</code> , 240	<code>list_factory()</code> (<code>spyn-</code> <code>naker.pyNN.utilities.ranged.SpynakerRangeDictionary</code> (<code>method)</code> , 247
<code>LED_FRONT_ACTIVE_TIME</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.push_bot.push_bot_push_bot_led.PushBotLED</code> (<code>naker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>attribute)</code> , 11	<code>attribute)</code> , 12
<code>LED_FRONT_ACTIVE_TIME</code> (<code>spyn-</code> <code>local_host</code> (<code>spynaker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>naker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>attribute)</code> , 13	<code>local_ip_address</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>static method)</code> , 234
<code>led_front_active_time()</code> (<code>spyn-</code> <code>naker.pyNN.protocols.munich_io_ethernet_protocol.MunichIoEthernetProtocol</code> (<code>static method)</code> , 234	<code>local_port</code> (<code>spynaker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>attribute)</code> , 10
<code>led_front_active_time()</code> (<code>spyn-</code> <code>naker.pyNN.protocols.MunichIoEthernetProtocol</code> (<code>static method)</code> , 240	<code>local_port</code> (<code>spynaker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>attribute)</code> , 12
<code>LED_TOTAL_PERIOD</code> (<code>spyn-</code> <code>local_port</code> (<code>spynaker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>naker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>attribute)</code> , 11	<code>local_size</code> (<code>spynaker.pyNN.models.pynn_population_common.PyNNPopulationCommon</code> (<code>attribute)</code> , 231
<code>LED_TOTAL_PERIOD</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.push_bot.push_bot_push_bot_led.PushBotLED</code> (<code>naker.pyNN.external_devices_models.push_bot.push_bot_led.PushBotLED</code> (<code>attribute)</code> , 13	<code>low()</code> (<code>spynaker.pyNN.utilities.utility_calls</code>), 251
<code>led_total_period()</code> (<code>spyn-</code> <code>low()</code> (<code>spynaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats</code> (<code>static method)</code> , 234	<code>low()</code> (<code>spynaker.pyNN.utilities.random_stats.AbstractRandomStats</code> (<code>method)</code> , 245
<code>led_total_period()</code> (<code>spyn-</code> <code>naker.pyNN.protocols.MunichIoEthernetProtocol</code> (<code>static method)</code> , 240	
M	
<code>LEFT_RETINA</code> (<code>spyn-</code> <code>machine_time_step()</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice</code> (<code>static method)</code> , 257	
<code>LEFT_RETINA</code> (<code>spyn-</code> <code>make_missing_string()</code> (in module <code>spynaker.pyNN.external_devices_models.MunichRetinaDevice</code> (<code>naker.pyNN.models.common.recording_utils</code>), 44	
<code>LEFT_RETINA_DISABLE</code> (<code>spyn-</code> <code>MANAGEMENT_BIT</code> (<code>spyn-</code> <code>naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice</code> (<code>naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device</code> (<code>static method)</code> , 234	

<i>attribute</i>), 24	<i>method</i>), 217
MANAGEMENT_BIT	(spyn- MASTER_POP_ENTRY_DTYPE (spyn-
<i>naker.pyNN.external_devices_models.MunichRetinaDevice</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators.master</i>
<i>attribute</i>), 32	<i>attribute</i>), 116
MANAGEMENT_MASK	(spyn- MASTER_POP_ENTRY_DTYPE (spyn-
<i>naker.pyNN.external_devices_models.munich_spinnaker_link_protocol.MunichRetinaDevice</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators.Master</i>
<i>attribute</i>), 24	<i>attribute</i>), 118
MANAGEMENT_MASK	(spyn- master_slave_key (spyn-
<i>naker.pyNN.external_devices_models.MunichRetinaDevice</i>	<i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich</i>
<i>attribute</i>), 32	<i>attribute</i>), 236
map_to_pre_coords()	(spyn- master_slave_key (spyn-
<i>naker.pyNN.models.neural_projections.connectors.kernel_connector.MunichIoSpiNNakerLinkProtocol</i>	<i>naker.pyNN.models.neural_projections.connectors.kernel_connector.MunichIoSpiNNakerLinkProtocol</i>
<i>method</i>), 63	<i>attribute</i>), 241
map_to_pre_coords()	(spyn- master_slave_set_master_clock_active()
<i>naker.pyNN.models.neural_projections.connectors.KernelConnector</i>	<i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.M</i>
<i>method</i>), 79	<i>method</i>), 236
mark_no_changes()	(spyn- master_slave_set_master_clock_active()
<i>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</i>	<i>naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol</i>
<i>method</i>), 194	<i>method</i>), 241
mark_no_changes()	(spyn- master_slave_set_master_clock_not_started()
<i>naker.pyNN.models.neuron.AbstractPopulationVertex</i>	(spyn- <i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.M</i>
<i>method</i>), 205	<i>method</i>), 236
mark_no_changes()	(spyn- master_slave_set_master_clock_not_started()
<i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamicsStable</i>	<i>naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol</i>
<i>method</i>), 147	<i>method</i>), 241
mark_no_changes()	(spyn- master_slave_set_slave()
<i>naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.SynapseDynamicsStable</i>	<i>naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol</i>
<i>method</i>), 149	<i>method</i>), 236
mark_no_changes()	(spyn- master_slave_set_slave()
<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStable</i>	<i>naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol</i>
<i>method</i>), 160	<i>method</i>), 241
mark_no_changes()	(spyn- master_slave_use_internal_counter()
<i>naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStable</i>	<i>naker.pyNN.protocols.munich_io_spinnaker_link_protocol.M</i>
<i>method</i>), 162	<i>method</i>), 236
mark_no_changes()	(spyn- master_slave_use_internal_counter()
<i>naker.pyNN.models.pynn_population_common.PyNNPopulationCommon</i>	<i>naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol</i>
<i>method</i>), 231	<i>method</i>), 241
mark_no_changes()	(spyn- MasterPopTableAsBinarySearch (class in spyn-
<i>naker.pyNN.models.pynn_projection_common.PyNNProjectionCommon</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators</i>
<i>method</i>), 233	117
mark_no_changes()	(spyn- MasterPopTableAsBinarySearch (class in spyn-
<i>naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex</i>	<i>naker.pyNN.models.neuron.master_pop_table_generators.master</i>
<i>method</i>), 212	116
mark_no_changes()	(spyn- MatrixGeneratorID (class in spyn-
<i>naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex</i>	<i>naker.pyNN.models.neuron.synapse_dynamics.abstract_generate</i>
<i>method</i>), 216	143
mark_regions_reloaded()	(spyn- max_delay (spyn- <i>naker.pyNN.abstract_spinnaker_common.AbstractSpiNN</i>
<i>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</i>	<i>naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex</i>
<i>method</i>), 194	max_delay (spyn- <i>naker.pyNN.spinnaker_simulator_interface.SpinnakerS</i>
mark_regions_reloaded()	(spyn- <i>attribute</i>), 258
<i>naker.pyNN.models.neuron.AbstractPopulationVertex</i>	max_delay (spyn- <i>naker.pyNN.utilities.spinnaker_failed_state.SpinnakerF</i>
<i>method</i>), 205	<i>attribute</i>), 250
mark_regions_reloaded()	(spyn- MAX_RATE (spyn- <i>naker.pyNN.models.common.neuron_recorder.NeuronRec</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>

MAX_RATE (spynnaker.pyNN.models.common.NeuronRecommendationException), 47	MODE_32 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 21
max_rate (spynnaker.pyNN.models.spike_source.spike_source_model), 217	MODE_32 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 29
max_size (spynnaker.pyNN.exceptions.SynapseRowTooBigException), 254	MODE_32 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 21
max_value (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device), 15	MODE_32 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 29
max_value (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device), 15	MODE_32 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 94
MaxRowInfo (class in spynnaker.pyNN.models.neuron.synapse_io.max_row_info), 170	model_name (spynnaker.pyNN.models.neuron.implementations.AbstractNeuron), 106
mean (spynnaker.pyNN.utilities.running_stats.RunningStats), 249	model_name (spynnaker.pyNN.models.neuron.implementations.neuron_implementations), 98
mean () (spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats), 244	model_name (spynnaker.pyNN.models.neuron.implementations.NeuronImplementation), 106
mean () (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats), 245	MOTOR_0_LEAKY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 12
MemReadException, 254	MOTOR_0_LEAKY (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 12
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 21	motor_0_leaky_velocity () (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 12
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.ExternalFPGA_RetinaDevice), 29	motor_0_leaky_velocity () (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 12
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.ExternalFPGA_RetinaDevice), 29	motor_0_leaky_velocity () (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 12
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_protocol), 24	MOTOR_0_PERMANENT (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 12
MERGED_POLARITY (spynnaker.pyNN.external_devices_models.ExternalFPGA_RetinaDevice), 29	MOTOR_0_PERMANENT (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 12
min_delay (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpinnakerCommon), 253	min_delay (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 13
min_delay (spynnaker.pyNN.spynnaker_simulator_interface.SpinnakerSimulatorInterface), 258	min_delay (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 13
min_delay (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState), 250	min_delay (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 13
min_value (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device), 15	min_value (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 15
min_value (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device), 16	min_value (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters), 16
mode (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpinnakerLinkProtocol), 236	mode (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpinnakerLinkProtocol), 236
mode (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol), 241	mode (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol), 241
MODE_128 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 21	MODE_128 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 21
MODE_128 (spynnaker.pyNN.external_devices_models.ExternalFPGA_RetinaDevice), 29	MODE_128 (spynnaker.pyNN.external_devices_models.ExternalFPGA_RetinaDevice), 29
MODE_16 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 21	MODE_16 (spynnaker.pyNN.external_devices_models.external_spinnaker_link_protocol), 21
MODE_16 (spynnaker.pyNN.external_devices_models.ExternalFPGA_RetinaDevice), 29	MODE_16 (spynnaker.pyNN.external_devices_models.ExternalFPGA_RetinaDevice), 29

N_BYTES_PER_INDEX	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 42	N_DELAYS (spynnaker.pyNN.models.utility_models.delays.delay_extension_machin attribute), 220	N_DELAYS (spynnaker.pyNN.models.utility_models.delays.DelayExtension attribute), 223
N_BYTES_PER_INDEX	(spyn- naker.pyNN.models.common.NeuronRecorder attribute), 47	N_EXTRA_PROVENANCE_DATA_ENTRIES (spyn- naker.pyNN.models.utility_models.delays.delay_extension_machin attribute), 220	N_EXTRA_PROVENANCE_DATA_ENTRIES (spyn- naker.pyNN.models.utility_models.delays.DelayExtensionMachin attribute), 223
N_BYTES_PER_POINTER	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 42	n_items (spynnaker.pyNN.utilities.running_stats.RunningStats attribute), 249	n_neurons (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protoc attribute), 239
N_BYTES_PER_POINTER	(spyn- naker.pyNN.models.common.NeuronRecorder attribute), 47	n_neurons (spynnaker.pyNN.protocols.RetinaKey at- tribute), 244	N_PACKETS_ADDED (spyn- naker.pyNN.models.utility_models.delays.delay_extension_machin attribute), 220
N_BYTES_PER_RATE	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 42	N_PACKETS_ADDED (spyn- naker.pyNN.models.utility_models.delays.DelayExtensionMachin attribute), 223	N_PACKETS_PROCESSED (spyn- naker.pyNN.models.utility_models.delays.delay_extension_machin attribute), 220
N_BYTES_PER_RATE	(spyn- naker.pyNN.models.common.NeuronRecorder attribute), 47	N_PACKETS_PROCESSED (spyn- naker.pyNN.models.utility_models.delays.DelayExtensionMachin attribute), 223	N_PACKETS_RECEIVED (spyn- naker.pyNN.models.utility_models.delays.delay_extension_machin attribute), 220
N_BYTES_PER_SIZE	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 42	N_PACKETS_RECEIVED (spyn- naker.pyNN.models.utility_models.delays.DelayExtensionMachin attribute), 223	N_PACKETS_RECEIVED (spyn- naker.pyNN.models.utility_models.delays.DelayExtensionMachin attribute), 223
N_BYTES_PER_SIZE	(spyn- naker.pyNN.models.common.NeuronRecorder attribute), 47	N_PACKETS_SENT (spyn- naker.pyNN.models.utility_models.delays.delay_extension_machin attribute), 220	N_PACKETS_SENT (spyn- naker.pyNN.models.utility_models.delays.DelayExtensionMachin attribute), 223
N_BYTES_PER_VALUE	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 43	N_PACKETS_SENT (spyn- naker.pyNN.models.utility_models.delays.delay_extension_machin attribute), 220	n_payload_bytes (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Retina attribute), 239
N_BYTES_PER_VALUE	(spyn- naker.pyNN.models.common.NeuronRecorder attribute), 47	n_payload_bytes (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Retina attribute), 239	n_projection_application_edges (spyn- naker.pyNN.protocols.RetinaPayload attribute), 244
N_BYTES_PER_WORD	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 43	n_projection_application_edges (spyn- naker.pyNN.protocols.RetinaPayload attribute), 244	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abs attribute), 129
N_BYTES_PER_WORD	(spyn- naker.pyNN.models.common.NeuronRecorder attribute), 47	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Abs attribute), 134	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134
N_CPU_CYCLES_PER_NEURON	(spyn- naker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 43	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134
N_CPU_CYCLES_PER_NEURON	(spyn- naker.pyNN.models.common.NeuronRecorder attribute), 47	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134
n_delay_stages	(spyn- naker.pyNN.models.neural_projections.projection_application_edge.ProjectionApplicationEdge attribute), 81	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134
n_delay_stages	(spyn- naker.pyNN.models.neural_projections.ProjectionApplicationEdge attribute), 84	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134
n_delay_stages	(spyn- naker.pyNN.models.utility_models.delays.delay_extension_vertex.DelayExtensionVertex attribute), 222	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134
n_delay_stages	(spyn- naker.pyNN.models.utility_models.delays.DelayExtensionVertex attribute), 226	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134	n_projection_application_edges (spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.tim attribute), 134

[attribute](#)), 130
[n_weight_terms](#) (spyn- [naker.pyNN.models.common.recording_utils](#)),
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 131
[n_weight_terms](#) (spyn- [naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS](#)
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 132
[n_weight_terms](#) (spyn- [naker.pyNN.models.neuron.implementations](#)),
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 132
[n_weight_terms](#) (spyn- [naker.pyNN.models.neuron.implementations.neuron_impl_standards](#)
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 133
[n_weight_terms](#) (spyn- [naker.pyNN.models.neuron.neuron_models](#)),
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 135
[n_weight_terms](#) (spyn- [naker.pyNN.models.neuron.neuron_models.neuron_model_izh](#)),
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 135
[n_weight_terms](#) (spyn- [naker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire](#)
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 136
[n_weight_terms](#) (spyn- [naker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire](#)
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 134
[n_weight_terms](#) (spyn- [naker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrate_and_fire](#)
[naker.pyNN.models.neuron.plasticity.stdp.timing_dependence](#)
[attribute](#)), 137
[n_words_for_plastic_connections\(\)](#) (spyn- [naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common](#)
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common](#)
[method](#)), 152
[n_words_for_plastic_connections\(\)](#) (spyn- [naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common](#)
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common](#)
[method](#)), 165
[n_words_for_static_connections\(\)](#) (spyn- [naker.pyNN.models.common.neuron_recorder](#)),
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common](#)
[method](#)), 152
[n_words_for_static_connections\(\)](#) (spyn- [naker.pyNN.models.common.neuron_recorder](#)
[naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common](#)
[method](#)), 165
[NATIVE_128_X_128](#) (spyn- [numpy_connectors_dtype](#) (spyn-
[naker.pyNN.external_devices_models.push_bot.push_bot_payload](#)
[attribute](#)), 12
[NATIVE_128_X_128](#) (spyn- [numpy_connectors_dtype](#) (spyn-
[naker.pyNN.external_devices_models.push_bot.push_bot_payload](#)
[attribute](#)), 14
[NATIVE_128_X_128](#) (spyn- [numpy_dtype](#) (spyn-
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol](#)
[attribute](#)), 239
[NATIVE_128_X_128](#) (spyn- [numpy_dtype](#) (spyn-
[naker.pyNN.protocols.RetinaKey](#) [attribute](#)),
[244](#)
[needs_buffering\(\)](#) (in module spyn- [numpy_synapses_dtype](#) (spyn-
[naker.pyNN.models.common](#)), 50

attribute), 50
 NUMPY_SYNAPSES_DTYPE (spyn- naker.pyNN.external_devices_models.push_bot.push_bot_etherne
 naker.pyNN.models.neural_projections.connectors.AbstractConnector), 8
 attribute), 65
 pause_stop_commands (spyn- naker.pyNN.external_devices_models.push_bot.push_bot_etherne
 attribute), 8
O
 ONE_TO_ONE_CONNECTOR (spyn- pixels (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.R
 naker.pyNN.models.neural_projections.connectors.abstract_connector_on_machine.ConnectorIDs
 attribute), 52
 pixels (spynnaker.pyNN.protocols.RetinaKey at-
 tribute), 244
 OneToOneConnector (class in spyn- PLASTIC_SYNAPTIC_WEIGHT_SATURATION_COUNT
 naker.pyNN.models.neural_projections.connectors), (spynnaker.pyNN.models.neuron.population_machine_vertex.Pop
 76
 OneToOneConnector (class in spyn- PLASTIC_SYNAPTIC_WEIGHT_SATURATION_COUNT
 naker.pyNN.models.neural_projections.connectors.one_to_one_connector), (spynnaker.pyNN.models.neuron.PopulationMachineVertex.EXT
 64
 attribute), 208
P
 plot_spikes() (in module spynnaker.plot_utils), 258
 p_rew (spynnaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics.StructuralCommon.SynapseDynamicsStructuralComm
 attribute), 152
 naker.pyNN.models.spike_source.spike_source_poisson_machine.
 attribute), 213
 p_rew (spynnaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon
 attribute), 165
 poll_individual_sensor_continuously() (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.M
 PARAMS_REGION (spyn- method), 236
 naker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.MunichMotorDevice
 attribute), 22
 poll_individual_sensor_continuously() (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 PARAMS_REGION (spyn- method), 241
 naker.pyNN.external_devices_models.MunichMotorDevice attribute), 30
 poll_individual_sensor_continuously_key
 PARAMS_SIZE (spyn- (spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.M
 attribute), 236
 naker.pyNN.external_devices_models.munich_spinnaker_link_motor_device.MunichMotorDevice
 attribute), 22
 poll_individual_sensor_continuously_key
 PARAMS_SIZE (spyn- (spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 attribute), 241
 naker.pyNN.external_devices_models.MunichMotorDevice attribute), 30
 poll_sensors_once() (spyn-
 pause_stop_commands (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich
 naker.pyNN.external_devices_models.external_spinnaker_link_fpga_device.ExternalFPGARetinaDevice
 attribute), 22
 poll_sensors_once() (spyn-
 pause_stop_commands (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.external_devices_models.ExternalFPGARetinaDevice method), 241
 attribute), 29
 poll_sensors_once_key (spyn-
 pause_stop_commands (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich
 attribute), 236
 naker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice
 attribute), 25
 poll_sensors_once_key (spyn-
 pause_stop_commands (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 attribute), 241
 naker.pyNN.external_devices_models.MunichRetinaDevice attribute), 32
 POPULATION_BASED_REGIONS (class in spyn-
 pause_stop_commands (spyn- naker.pyNN.utilities.constants), 248
 naker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice
 attribute), 15
 naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS
 attribute), 248
 pause_stop_commands (spyn- PopulationMachineVertex (class in spyn-
 naker.pyNN.external_devices_models.push_bot.AbstractPushBotRetinaDevice naker.pyNN.models.neuron), 208
 attribute), 16
 PopulationMachineVertex (class in spyn-
 pause_stop_commands (spyn- naker.pyNN.models.neuron.population_machine_vertex)
 naker.pyNN.external_devices_models.push_bot.push_bot_etherne_push_bot_etherne_laser_device.PushBotEtherneLaserDe
 attribute), 7
 198

[illegible]

`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`attribute`), 236 `method`), 241

`protocol_instance` (`spyn- push_bot_laser_set_frequency_key` (`spyn-`
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich`
`attribute`), 241 `attribute`), 236

`protocol_property` (`spyn- push_bot_laser_set_frequency_key` (`spyn-`
`naker.pyNN.external_devices_models.push_bot.abstract_push_bot_device.MunichIoSpiNNakerLinkProtocol`
`attribute`), 15 `attribute`), 241

`protocol_property` (`spyn- push_bot_led_back_active_time()` (`spyn-`
`naker.pyNN.external_devices_models.push_bot.AbstractPushBotDevice` `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich`
`attribute`), 16 `method`), 236

`PROVENANCE_DATA` (`spyn- push_bot_led_back_active_time()` (`spyn-`
`naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 248 `method`), 241

`PROVENANCE_REGION` (`spyn- push_bot_led_back_active_time_key` (`spyn-`
`naker.pyNN.models.spike_source.spike_source_poisson_mach` `naker.pyNN.models.spike_source_poisson_mach`
`attribute`), 213 `attribute`), 236

`pull_off_cached_lists()` (`in module spyn- push_bot_led_back_active_time_key` (`spyn-`
`naker.pyNN.models.common`), 50 `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 241

`pull_off_cached_lists()` (`in module spyn-`
`naker.pyNN.models.common.recording_utils`), `push_bot_led_front_active_time()` (`spyn-`
45 `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.Munich`
`attribute`), 236

`PUSH_BOT` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol.MODES`
`attribute`), 235 `push_bot_led_front_active_time()` (`spyn-`
`attribute`), 240 `method`), 241

`push_bot_laser_config_active_time()` `push_bot_led_front_active_time_key`
(`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`method`), 236 `attribute`), 236

`push_bot_laser_config_active_time()` `push_bot_led_front_active_time_key`
(`spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 241 `attribute`), 241

`push_bot_laser_config_active_time_key` `push_bot_led_set_frequency()` (`spyn-`
(`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`attribute`), 236 `method`), 236

`push_bot_laser_config_active_time_key` `push_bot_led_set_frequency()` (`spyn-`
(`spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 241 `method`), 241

`push_bot_laser_config_total_period()` `push_bot_led_set_frequency_key` (`spyn-`
(`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`method`), 236 `attribute`), 236

`push_bot_laser_config_total_period()` `push_bot_led_set_frequency_key` (`spyn-`
(`spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 241 `attribute`), 242

`push_bot_laser_config_total_period_key` `push_bot_led_total_period()` (`spyn-`
(`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`attribute`), 236 `method`), 236

`push_bot_laser_config_total_period_key` `push_bot_led_total_period()` (`spyn-`
(`spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 241 `method`), 242

`push_bot_laser_set_frequency()` (`spyn- push_bot_led_total_period_key` (`spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`method`), 236 `attribute`), 236

`push_bot_laser_set_frequency()` (`spyn- push_bot_led_total_period_key` (`spyn-`

`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 242 `method`), 242
`push_bot_motor_0_leaking_towards_zero()` `push_bot_speaker_config_active_time_key`
(`spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 236 `attribute`), 237
`push_bot_motor_0_leaking_towards_zero()` `push_bot_speaker_config_active_time_key`
(`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 242 `attribute`), 242
`push_bot_motor_0_leaking_towards_zero_key` `push_bot_speaker_config_total_period()`
(`spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 237 `method`), 237
`push_bot_motor_0_leaking_towards_zero_key` `push_bot_speaker_config_total_period()`
(`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 242 `method`), 242
`push_bot_motor_0_permanent()` (`spyn- push_bot_speaker_config_total_period_key`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 237 `attribute`), 237
`push_bot_motor_0_permanent()` (`spyn- push_bot_speaker_config_total_period_key`
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 242 `attribute`), 242
`push_bot_motor_0_permanent_key` (`spyn- push_bot_speaker_set_melody()` (`spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 237 `method`), 237
`push_bot_motor_0_permanent_key` (`spyn- push_bot_speaker_set_melody()` (`spyn-`
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 242 `method`), 242
`push_bot_motor_1_leaking_towards_zero()` `push_bot_speaker_set_melody_key` (`spyn-`
(`spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 237 `attribute`), 237
`push_bot_motor_1_leaking_towards_zero()` `push_bot_speaker_set_melody_key` (`spyn-`
(`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 242 `attribute`), 242
`push_bot_motor_1_leaking_towards_zero_key` `push_bot_speaker_set_tone()` (`spyn-`
(`spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 237 `method`), 237
`push_bot_motor_1_leaking_towards_zero_key` `push_bot_speaker_set_tone()` (`spyn-`
(`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 242 `method`), 242
`push_bot_motor_1_permanent()` (`spyn- push_bot_speaker_set_tone_key` (`spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 237 `attribute`), 237
`push_bot_motor_1_permanent()` (`spyn- push_bot_speaker_set_tone_key` (`spyn-`
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 242 `attribute`), 242
`push_bot_motor_1_permanent_key` (`spyn- PushBotEthernetDevice` (`class in spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 237 `5`
`push_bot_motor_1_permanent_key` (`spyn- PushBotEthernetLaserDevice` (`class in spyn-`
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.external_devices_models.push_bot.push_bot_etherne`
`attribute`), 242 `6`
`push_bot_speaker_config_active_time()` `PushBotEthernetLEDDevice` (`class in spyn-`
(`spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol` (`spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 237 `7`
`push_bot_speaker_config_active_time()` `PushBotEthernetMotorDevice` (`class in spyn-`

`naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (class in `spyn-` `pwm_pin_output_timer_a_channel_1_ratio()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotLaser` (class in `spyn-` `pwm_pin_output_timer_a_channel_1_ratio()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotLaser` (class in `spyn-` `pwm_pin_output_timer_a_channel_1_ratio()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 242

`PushBotLED` (class in `spyn-` `pwm_pin_output_timer_a_channel_1_ratio_key` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotLED` (class in `spyn-` `pwm_pin_output_timer_a_channel_1_ratio_key` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 242

`PushBotMotor` (class in `spyn-` `pwm_pin_output_timer_a_duration()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotMotor` (class in `spyn-` `pwm_pin_output_timer_a_duration()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 242

`PushBotRetinaResolution` (class in `spyn-` `pwm_pin_output_timer_a_duration_key` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotRetinaResolution` (class in `spyn-` `pwm_pin_output_timer_a_duration_key` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 242

`PushBotRetinaViewer` (class in `spyn-` `pwm_pin_output_timer_b_channel_0_ratio()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotRetinaViewer` (class in `spyn-` `pwm_pin_output_timer_b_channel_0_ratio()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 242

`PushBotSpeaker` (class in `spyn-` `pwm_pin_output_timer_b_channel_0_ratio_key()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotSpeaker` (class in `spyn-` `pwm_pin_output_timer_b_channel_0_ratio_key()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 242

`PushBotTranslator` (class in `spyn-` `pwm_pin_output_timer_b_channel_1_ratio()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 237

`PushBotWiFiConnection` (class in `spyn-` `pwm_pin_output_timer_b_channel_1_ratio()` `naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot` (method), 242

`pwm_pin_output_timer_a_channel_0_ratio()` `pwm_pin_output_timer_b_channel_1_ratio_key` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol` (method), 237

`pwm_pin_output_timer_a_channel_0_ratio()` `pwm_pin_output_timer_b_channel_1_ratio_key` (`spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (method), 242

`pwm_pin_output_timer_a_channel_0_ratio_key` `pwm_pin_output_timer_b_duration()` (`spyn-` (`spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol` (method), 237

`pwm_pin_output_timer_a_channel_0_ratio_key` `pwm_pin_output_timer_b_duration()` (`spyn-`

`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.synapse_dynamics`,
`method`), 242 159
`pwm_pin_output_timer_b_duration_key` `PyNNSynapseDynamics` (class in `spyn-`
`(spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`attribute`), 237 145
`pwm_pin_output_timer_b_duration_key` `(spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 242 **Q**
`pwm_pin_output_timer_c_channel_0_ratio()` `query_state_of_io_lines()` (`spyn-`
`(spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`method`), 237 `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`pwm_pin_output_timer_c_channel_0_ratio()` `query_state_of_io_lines()` (`spyn-`
`(spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`method`), 242 `method`), 243
`pwm_pin_output_timer_c_channel_0_ratio_key` `query_state_of_io_lines_key` (`spyn-`
`(spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`attribute`), 237 `naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`pwm_pin_output_timer_c_channel_0_ratio_key` `query_state_of_io_lines_key` (`spyn-`
`(spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol`
`attribute`), 243 `attribute`), 243
`pwm_pin_output_timer_c_channel_1_ratio()` **R**
`(spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`method`), 237 `range_dict_vertex_slice` (class in `spyn-`
`pwm_pin_output_timer_c_channel_1_ratio()` `naker.pyNN.models.neuron.implementations`,
`(spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` 107
`method`), 242 `RangeDictVertexSlice` (class in `spyn-`
`pwm_pin_output_timer_c_channel_1_ratio_key()` `naker.pyNN.models.neuron.implementations.range_dict_vertex_`
`(spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` 99
`method`), 237 `rate` (`spynaker.pyNN.models.neuron.implementations.spike_source_poisson_vertex_`
`attribute`), 217
`pwm_pin_output_timer_c_channel_1_ratio_key()` `read_data()` (`spyn-`
`(spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.implementations.abstract_neuron_imp`
`method`), 242 `method`), 94
`pwm_pin_output_timer_c_duration()` (`spyn- read_data()` (`spyn-`
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.implementations.abstract_standard_n`
`method`), 237 `method`), 96
`pwm_pin_output_timer_c_duration()` (`spyn- read_data()` (`spyn-`
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.implementations.AbstractNeuronImpl`
`method`), 242 `method`), 107
`pwm_pin_output_timer_c_duration_key` `read_data()` (`spyn-`
`(spynaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.implementations.AbstractStandardNe`
`attribute`), 237 `method`), 101
`pwm_pin_output_timer_c_duration_key` `read_data()` (`spyn-`
`(spynaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.implementations.neuron_impl standa`
`attribute`), 243 `method`), 99
`pynn7_format()` (`spyn- read_data()` (`spyn-`
`naker.pyNN.models.recording_common.RecordingCommon` `naker.pyNN.models.neuron.implementations.NeuronImplStandard`
`static method`), 233 `method`), 104
`PyNNPopulationCommon` (class in `spyn- read_data()` (`spyn-`
`naker.pyNN.models.pynn_population_common`), `naker.pyNN.models.neuron.implementations.Struct`
230 `method`), 102
`PyNNProjectionCommon` (class in `spyn- read_data()` (`spyn-`
`naker.pyNN.models.pynn_projection_common`), `naker.pyNN.models.neuron.implementations.struct.Struct`
233 `method`), 100
`PyNNSynapseDynamics` (class in `spyn-`

method), 237	method), 217
remove_payload_logic_to_current_output() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243	reserve_memory_regions() (spyn- naker.pyNN.external_devices_models.munich_spinnaker_link_mo- del method), 24
remove_payload_logic_to_current_output_key (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol attribute), 238	reserve_memory_regions() (spyn- naker.pyNN.external_devices_models.munich_spinnaker_link_mo- del method), 31
remove_payload_logic_to_current_output_key (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 243	reset() (spynnaker.pyNN.spynnaker_simulator_interface.SpynnakerSimu- lator method), 217
requires_data_generation (spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 195	reset() (spynnaker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState static method), 250
requires_data_generation (spyn- naker.pyNN.models.neuron.AbstractPopulationVertex attribute), 205	reset_number_of_neurons_per_core() (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon method), 253
requires_mapping (spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 195	reset_retina() (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 238
requires_mapping (spyn- naker.pyNN.models.neuron.AbstractPopulationVertex attribute), 206	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping (spyn- naker.pyNN.models.pynn_population_common.PyNNPopulationCommon attribute), 231	reset_retina() (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping (spyn- naker.pyNN.models.pynn_projection_common.PyNNProjectionCommon attribute), 233	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping (spyn- naker.pyNN.models.spike_source.spike_source_array_vertex.SpikeSourceArrayVertex attribute), 212	reset_retina() (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping (spyn- naker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex attribute), 217	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping() (spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic method), 147	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping() (spyn- naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP method), 149	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping() (spyn- naker.pyNN.models.neuron.AbstractPopulationVertex method), 206	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping() (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic method), 160	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_mapping() (spyn- naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP method), 162	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_memory_regions_to_be_reloaded() (spyn- naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex method), 195	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_memory_regions_to_be_reloaded() (spyn- naker.pyNN.models.neuron.AbstractPopulationVertex method), 206	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_memory_regions_to_be_reloaded() (spyn- naker.pyNN.models.spike_source.spike_source_poisson_machine_vertex.SpikeSourcePoissonMachineVertex method), 220	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243
requires_memory_regions_to_be_reloaded() (spyn- naker.pyNN.models.utility_models.delays.delay_extension_machine_vertex.DelayExtensionMachineVertex method), 220	reset_retina() (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol method), 243

`naker.pyNN.models.utility_models.delays.DelayExtensionMunichRetinaDevice` (class in `spynnaker.pyNN.models.utility_models.delays.DelayExtensionMunichRetinaDevice`), 223

`RetinaKey` (class in `spynnaker.pyNN.protocols`), 243

`RetinaKey` (class in `spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol`), 238

`RetinaPayload` (class in `spynnaker.pyNN.protocols`), 244

`RetinaPayload` (class in `spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol`), 239

`RIGHT_RETINA` (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 24

`RIGHT_RETINA` (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 32

`RIGHT_RETINA_DISABLE` (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 24

`RIGHT_RETINA_DISABLE` (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 32

`RIGHT_RETINA_ENABLE` (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 24

`RIGHT_RETINA_ENABLE` (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 32

`RIGHT_RETINA_KEY_SET` (spynnaker.pyNN.external_devices_models.munich_spinnaker_link_retina_device.MunichRetinaDevice attribute), 24

`RIGHT_RETINA_KEY_SET` (spynnaker.pyNN.external_devices_models.MunichRetinaDevice attribute), 32

`ring_buffer_sigma` (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 195

`ring_buffer_sigma` (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 206

`ring_buffer_sigma` (spynnaker.pyNN.models.neuron.synaptic_manager.SynapticManager attribute), 200

`ring_buffer_sigma` (spynnaker.pyNN.models.neuron.SynapticManager attribute), 208

`routing_key_partition_atom_mapping()` (spynnaker.pyNN.external_devices_models.external_device_lif_control_vertex.ExternalDeviceLifControlVertex method), 20

`ROW_LENGTH_MASK` (spynnaker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_as_binary_search.MasterPopTableAsBinarySearch attribute), 116

`ROW_LENGTH_MASK` (spynnaker.pyNN.external_devices_models.abstract_multicast_controller.MulticastController attribute), 118

`run()` (spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon method), 253

`run()` (spynnaker.pyNN.connections.ethernet_control_connection.EthernetControlConnection method), 4

`run()` (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters method), 12

`run()` (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters method), 14

`RunningStats` (class in `spynnaker.pyNN.utilities.running_stats`), 249

`run_spinnaker_link_retina_device` (spynnaker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex attribute), 190

`SDP_PORT_SIZE` (spynnaker.pyNN.models.neuron.AbstractPopulationVertex attribute), 201

`safe` (spynnaker.pyNN.models.neural_projections.connectors.abstract_connector.AbstractConnector attribute), 51

`safe` (spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector attribute), 66

`spinnaker_link_retina_device` (spynnaker.pyNN.models.common.neuron_recorder.NeuronRecorder attribute), 43

`SPINLOCK_SIZE` (spynnaker.pyNN.models.common.NeuronRecorder attribute), 47

`spinnaker_link_retina_device` (spynnaker.pyNN.models.neuron.population_machine_vertex.PopulationMachineVertex attribute), 199

`SPINNING_COUNT` (spynnaker.pyNN.models.neuron.PopulationMachineVertex.EXTRA_PARAMETERS attribute), 208

`spinnaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex` (class in `spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex.SpikeSourcePoissonVertex`), 217

`send()` (spynnaker.pyNN.external_devices_models.push_bot.push_bot_parameters.PushBotParameters method), 10

`send_spike()` (spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpyNNakerLiveSpikesConnection method), 4

`send_spikes()` (spynnaker.pyNN.connections.spynnaker_live_spikes_connection.SpyNNakerLiveSpikesConnection method), 4

`send_type` (spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot.PushBot attribute), 15

`send_type` (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBot attribute), 16

`SEND_TYPE_ACCUM` (spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBot attribute), 18

SEND_TYPE_FRACT (spyn- method), 8
 naker.pyNN.external_devices_models.abstract_multicast_controllable_device.SendType (spyn-
 attribute), 18
 SEND_TYPE_INT (spyn- method), 232
 naker.pyNN.external_devices_models.abstract_multicast_controllable_device.SendType (spyn-
 attribute), 18
 SEND_TYPE_UACCUM (spyn- method), 35
 naker.pyNN.external_devices_models.abstract_multicast_controllable_device.SendType (spyn-
 attribute), 18
 SEND_TYPE_UFRACT (spyn- method), 38
 naker.pyNN.external_devices_models.abstract_multicast_controllable_device.SendType (spyn-
 attribute), 18
 SEND_TYPE_UINT (spyn- method), 195
 naker.pyNN.external_devices_models.abstract_multicast_controllable_device.SendType (spyn-
 attribute), 18
 SendType (class in spyn- method), 206
 naker.pyNN.external_devices_models.abstract_multicast_controllable_device.int () (spyn-
 18
 naker.pyNN.models.pynn_population_common.PyNNPopulationCommon)

sensor_transmission_key () (spyn- method), 232
 naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol (spyn-
 method), 238
 naker.pyNN.models.pynn_population_common.PyNNPopulationCommon)

sensor_transmission_key () (spyn- method), 232
 naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol.mode () (spyn-
 method), 243
 naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol

sent_mode_command () (spyn- set_mode () (spyn- naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
 static method), 238
 set_mode_key (spyn-
 naker.pyNN.models.pynn_population_common.PyNNPopulationCommon)

sent_mode_command () (spyn- naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol
 naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 238
 static method), 243
 set_mode_key (spyn-
 naker.pyNN.models.pynn_population_common.PyNNPopulationCommon)

set () (spyn- naker.pyNN.models.pynn_population_common.PyNNPopulationCommon) (spyn-
 method), 231
 naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol attribute), 243

set () (spyn- naker.pyNN.utilities.extracted_data.ExtractedData_model_max_atoms_per_core () (spyn-
 method), 248
 naker.pyNN.models.abstract_pynn_model.AbstractPyNNModel)

set_a_plus_a_minus () (spyn- class method), 229
 naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus (AbstractHasAPlusAMinus
 method), 137
 naker.pyNN.models.neuron.abstract_pynn_neuron_model.AbstractPyNNNeuronModel)

set_a_plus_a_minus () (spyn- class method), 196
 naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_has_a_plus_a_minus (AbstractHasAPlusAMinus
 method), 140
 naker.pyNN.models.neuron.AbstractPyNNNeuronModel)

set_by_selector () (spyn- class method), 210
 naker.pyNN.models.pynn_population_common.PyNNPopulationCommon)

set_command_protocol () (spyn- class method), 213
 naker.pyNN.external_devices_models.push_bot_push_bot_ethernet_push_bot_ethernet_device.PushBotEthernetDevice
 method), 6
 naker.pyNN.models.spike_source.spike_source_poisson.SpikeSourcePoisson)

set_command_protocol () (spyn- class method), 219
 naker.pyNN.external_devices_models.push_bot_push_bot_ethernet_push_bot_ethernet_device.PushBotEthernetLaserDevice
 method), 7
 naker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon)

set_command_protocol () (spyn- method), 253
 naker.pyNN.external_devices_models.push_bot_push_bot_ethernet_push_bot_ethernet_device.PushBotEthernetLEDDevice
 method), 8
 naker.pyNN.spynnaker_simulator_interface.SpynnakerSimulatorInterface)

set_command_protocol () (spyn- method), 258
 naker.pyNN.external_devices_models.push_bot_push_bot_ethernet_push_bot_ethernet_device.PushBotEthernetMotorDevice

`naker.pyNN.utilities.spynnaker_failed_state.SpynnakerFailedState` (method), 250

`naker.pyNN.models.neural_projections.connectors.SmallWorldConnector` (method), 77

`set_output_pattern_for_payload()` (spyn- `set_recording()` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (method), 238
`naker.pyNN.models.neural_projections.connectors.SmallWorldConnector` (method), 41

`set_output_pattern_for_payload()` (spyn- `set_recording()` (spyn-
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (method), 243
`naker.pyNN.models.common.AbstractNeuronRecordable` (method), 46

`set_output_pattern_for_payload_key` `set_recording()` (spyn-
`(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (attribute), 238
`naker.pyNN.models.common.EIEIOSpikeRecorder` (method), 42

`set_output_pattern_for_payload_key` `set_recording()` (spyn-
`(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (attribute), 243
`naker.pyNN.models.common.EIEIOSpikeRecorder` (method), 47

`set_payload_pins_to_high_impedance()` `set_recording()` (spyn-
`(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (method), 238
`naker.pyNN.models.common.NeuronRecorder` (method), 44

`set_payload_pins_to_high_impedance()` `set_recording()` (spyn-
`(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (method), 243
`naker.pyNN.models.common.NeuronRecorder` (method), 49

`set_payload_pins_to_high_impedance_key` `set_recording()` (spyn-
`(spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (attribute), 238
`naker.pyNN.models.common.PopulationVertex` (method), 195

`set_payload_pins_to_high_impedance_key` `set_recording()` (spyn-
`(spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (attribute), 243
`naker.pyNN.models.neuron.AbstractPopulationVertex` (method), 206

`set_projection_information()` (spyn- `set_recording_spikes()` (spyn-
`naker.pyNN.models.neural_projections.connectors.abstract_naker.pyNN.models.common.AbstractSpikeRecordable` (method), 51
`naker.pyNN.models.common.AbstractSpikeRecordable` (method), 41

`set_projection_information()` (spyn- `set_recording_spikes()` (spyn-
`naker.pyNN.models.neural_projections.connectors.AbstractNakerPyNNConnector` (method), 66
`naker.pyNN.models.common.AbstractSpikeRecordable` (method), 47

`set_projection_information()` (spyn- `set_recording_spikes()` (spyn-
`naker.pyNN.models.neural_projections.connectors.distance_dependent_naker.pyNN.models.common.DistanceDependentPopulability` (method), 55
`naker.pyNN.models.common.DistanceDependentPopulability` (method), 195

`set_projection_information()` (spyn- `set_recording_spikes()` (spyn-
`naker.pyNN.models.neural_projections.connectors.DistanceDependentNakerPyNNConnector` (method), 70
`naker.pyNN.models.common.DistanceDependentPopulability` (method), 206

`set_projection_information()` (spyn- `set_recording_spikes()` (spyn-
`naker.pyNN.models.neural_projections.connectors.fixed_number_naker.pyNN.models.common.FixedNumberSpikeConnector` (method), 57
`naker.pyNN.models.common.FixedNumberSpikeConnector` (method), 212

`set_projection_information()` (spyn- `set_recording_spikes()` (spyn-
`naker.pyNN.models.neural_projections.connectors.fixed_number_naker.pyNN.models.common.FixedNumberSpikeConnector` (method), 58
`naker.pyNN.models.common.FixedNumberSpikeConnector` (method), 217

`set_projection_information()` (spyn- `set_recording_spikes()` (spyn-
`naker.pyNN.models.neural_projections.connectors.FixedNumberNakerPyNNConnector` (method), 71
`naker.pyNN.models.utility_models.spike_injector.spike_injector_v` (method), 228

`set_projection_information()` (spyn- `set_retina_key()` (spyn-
`naker.pyNN.models.neural_projections.connectors.FixedNumberNakerPyNNConnector` (method), 72
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` (method), 238

`set_projection_information()` (spyn- `set_retina_key()` (spyn-
`naker.pyNN.models.neural_projections.connectors.small_world_naker.pyNN.models.common.SmallWorldConnector` (method), 65
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` (method), 243

`set_projection_information()` (spyn- `set_retina_key_key` (spyn-

`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol`
`attribute`), 238

`set_retina_key_key` (spyn- `set_value()` (spyn-
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.AbstractPopulationVertex`
`attribute`), 243 `method`), 206

`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`), 234 `method`), 147

`set_retina_transmission()` (spyn- `set_value()` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics`
`method`), 238 `method`), 149

`set_retina_transmission()` (spyn- `set_value()` (spyn-
`naker.pyNN.protocols.MunichIoEthernetProtocol` `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics`
`static method`), 240 `method`), 160

`set_retina_transmission()` (spyn- `set_value()` (spyn-
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamics`
`method`), 243 `method`), 162

`set_retina_transmission_key` (spyn- `set_value()` (spyn-
`naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.neuron.spike_source_poisson_vertex.SpikeSourcePoissonVertex`
`attribute`), 238 `method`), 218

`set_retina_transmission_key` (spyn- `set_value_by_selector()` (spyn-
`naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol` `naker.pyNN.models.abstract_models.abstract_population_settable`
`attribute`), 243 `method`), 36

`set_space()` (spyn- `set_value_by_selector()` (spyn-
`naker.pyNN.models.neural_projections.connectors.abstract_neural_projections_connectors.AbstractNeuralProjectionsConnectors` `naker.pyNN.models.abstract_models.AbstractPopulationSettable`
`method`), 51 `method`), 39

`set_space()` (spyn- `shape2word()` (in module spyn-
`naker.pyNN.models.neural_projections.connectors.AbstractNeuralProjectionsConnectors` `naker.pyNN.models.neural_projections.connectors.kernel_connectors.KernelConnectors`
`method`), 66 `method`), 63

`set_synapse_dynamics()` (spyn- `show_connection_set()` (spyn-
`naker.pyNN.models.abstract_models.abstract_accepts_incoming_synapses.AbstractAcceptsIncomingSynapses` `naker.pyNN.models.neural_projections.connectors.csa_connectors.CSAConnectors`
`method`), 34 `method`), 54

`set_synapse_dynamics()` (spyn- `show_connection_set()` (spyn-
`naker.pyNN.models.abstract_models.AbstractAcceptsIncomingSynapses` `naker.pyNN.models.neural_projections.connectors.CSAConnectors`
`method`), 37 `method`), 69

`set_synapse_dynamics()` (spyn- `SimplePopulationSettable` (class in spyn-
`naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex` `naker.pyNN.models.common`), 49
`method`), 195 `SimplePopulationSettable` (class in spyn-
`naker.pyNN.models.neuron.AbstractPopulationVertex` `naker.pyNN.models.common.simple_population_settable`),
`method`), 206 45

`set_value()` (spyn- `SINGLE_BIT_FLAG_BIT` (spyn-
`naker.pyNN.models.abstract_models.abstract_settable.AbstractSettable` `naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generators`
`method`), 37 `method`), 16

`set_value()` (spyn- `SINGLE_BIT_FLAG_BIT` (spyn-
`naker.pyNN.models.abstract_models.AbstractSettable` `naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerators`
`method`), 40 `attribute`), 118

`set_value()` (spyn- `size` (spynnaker.pyNN.models.neuron.generator_data.GeneratorData
`naker.pyNN.models.common.simple_population_settable.SimplePopulationSettable` `naker.pyNN.models.common.PyNNPopulationSettable`
`method`), 45 `attribute`), 232

`set_value()` (spyn- `size` (spynnaker.pyNN.models.utility_models.delays.delay_generator_data.DelayGeneratorData
`naker.pyNN.models.common.SimplePopulationSettable` `attribute`), 223
`method`), 49

`set_value()` (spyn- `size()` (spynnaker.pyNN.models.pynn_projection_common.PyNNProjectionCommon)
`method`), 233

Index 335

[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 57](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 34](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 58](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 34](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 59](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 35](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 60](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 35](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 61](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 36](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 63](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 36](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 64](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 37](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 65](#)
[spynnaker.pyNN.models.abstract_models.abstract_neuron_model \(module\), 37](#)
[spynnaker.pyNN.models.abstract_pynn_model \(module\), 79](#)
[spynnaker.pyNN.models.abstract_pynn_model \(module\), 229](#)
[spynnaker.pyNN.models.common \(module\), 45](#)
[spynnaker.pyNN.models.common.abstract_neuron_model \(module\), 40](#)
[spynnaker.pyNN.models.common.abstract_neuron_model \(module\), 41](#)
[spynnaker.pyNN.models.common.eieio_spikes_pynn_model \(module\), 42](#)
[spynnaker.pyNN.models.common.eieio_spikes_pynn_model \(module\), 42](#)
[spynnaker.pyNN.models.common.multi_spikes_pynn_model \(module\), 42](#)
[spynnaker.pyNN.models.common.neuron_recorder_pynn_model \(module\), 42](#)
[spynnaker.pyNN.models.common.recording_utils_pynn_model \(module\), 44](#)
[spynnaker.pyNN.models.common.simple_population_pynn_model \(module\), 45](#)
[spynnaker.pyNN.models.defaults \(module\), 230](#)
[spynnaker.pyNN.models.neural_projections \(module\), 82](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 65](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 50](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 51](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 52](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 53](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 54](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 55](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 56](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 57](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 58](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 59](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 60](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 61](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 63](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 64](#)
[spynnaker.pyNN.models.neural_projections.connector \(module\), 65](#)
[spynnaker.pyNN.models.neural_projections.delay_affector \(module\), 79](#)
[spynnaker.pyNN.models.neural_projections.delay_affector \(module\), 79](#)
[spynnaker.pyNN.models.neural_projections.delayed_application \(module\), 80](#)
[spynnaker.pyNN.models.neural_projections.delayed_application \(module\), 80](#)
[spynnaker.pyNN.models.neural_projections.projection \(module\), 81](#)
[spynnaker.pyNN.models.neural_projections.projection \(module\), 81](#)
[spynnaker.pyNN.models.neural_projections.synapse_initializer \(module\), 82](#)
[spynnaker.pyNN.models.neural_properties \(module\), 85](#)
[spynnaker.pyNN.models.neural_properties \(module\), 85](#)
[spynnaker.pyNN.models.neuron \(module\), 201](#)
[spynnaker.pyNN.models.neuron.abstract_population_vector \(module\), 190](#)
[spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model \(module\), 196](#)
[spynnaker.pyNN.models.neuron.abstract_pynn_neuron_model \(module\), 196](#)
[spynnaker.pyNN.models.neuron.abstract_connector \(module\), 187](#)
[spynnaker.pyNN.models.neuron.additional_inputs \(module\), 187](#)
[spynnaker.pyNN.models.neuron.additional_inputs.abstract_generate_connector_on_machine \(module\), 186](#)
[spynnaker.pyNN.models.neuron.additional_inputs.abstract_to_all_connector \(module\), 186](#)
[spynnaker.pyNN.models.neuron.additional_inputs.additive_connector \(module\), 186](#)
[spynnaker.pyNN.models.neuron.builds \(module\), 189](#)
[spynnaker.pyNN.models.neuron.builds.eif_cond_alpha \(module\), 189](#)
[spynnaker.pyNN.models.neuron.builds.distance_dependent_probability_connector \(module\), 189](#)
[spynnaker.pyNN.models.neuron.builds.hh_cond_exp \(module\), 189](#)
[spynnaker.pyNN.models.neuron.builds.number_post_connector \(module\), 189](#)
[spynnaker.pyNN.models.neuron.builds.number_pre_connector \(module\), 189](#)

[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 89
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 123
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 89
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 119
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 90
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 120
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 90
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 122
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 90
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 142
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 90
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 142
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 90
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 127
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 90
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 127
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 91
[spynnaker.pyNN.models.neuron.builds.if_copy_synapse_base](#) (module), 128
[spynnaker.pyNN.models.neuron.builds.izk_synapse_base](#) (module), 91
[spynnaker.pyNN.models.neuron.builds.izk_synapse_base](#) (module), 128
[spynnaker.pyNN.models.neuron.builds.izk_synapse_base](#) (module), 91
[spynnaker.pyNN.models.neuron.builds.izk_synapse_base](#) (module), 128
[spynnaker.pyNN.models.neuron.connection_base](#) (module), 197
[spynnaker.pyNN.models.neuron.connection_base](#) (module), 128
[spynnaker.pyNN.models.neuron.generator_data](#) (module), 198
[spynnaker.pyNN.models.neuron.generator_data](#) (module), 133
[spynnaker.pyNN.models.neuron.implementation](#) (module), 100
[spynnaker.pyNN.models.neuron.implementation](#) (module), 129
[spynnaker.pyNN.models.neuron.implementation](#) (module), 93
[spynnaker.pyNN.models.neuron.implementation](#) (module), 130
[spynnaker.pyNN.models.neuron.implementation](#) (module), 95
[spynnaker.pyNN.models.neuron.implementation](#) (module), 131
[spynnaker.pyNN.models.neuron.implementation](#) (module), 97
[spynnaker.pyNN.models.neuron.implementation](#) (module), 131
[spynnaker.pyNN.models.neuron.implementation](#) (module), 99
[spynnaker.pyNN.models.neuron.implementation](#) (module), 132
[spynnaker.pyNN.models.neuron.implementation](#) (module), 99
[spynnaker.pyNN.models.neuron.implementation](#) (module), 133
[spynnaker.pyNN.models.neuron.input_types](#) (module), 111
[spynnaker.pyNN.models.neuron.input_types](#) (module), 140
[spynnaker.pyNN.models.neuron.input_types](#) (module), 107
[spynnaker.pyNN.models.neuron.input_types](#) (module), 137
[spynnaker.pyNN.models.neuron.input_types](#) (module), 107
[spynnaker.pyNN.models.neuron.input_types](#) (module), 137
[spynnaker.pyNN.models.neuron.input_types](#) (module), 109
[spynnaker.pyNN.models.neuron.input_types](#) (module), 138
[spynnaker.pyNN.models.neuron.input_types](#) (module), 110
[spynnaker.pyNN.models.neuron.input_types](#) (module), 139
[spynnaker.pyNN.models.neuron.master_population](#) (module), 117
[spynnaker.pyNN.models.neuron.master_population](#) (module), 139
[spynnaker.pyNN.models.neuron.master_population](#) (module), 115
[spynnaker.pyNN.models.neuron.master_population](#) (module), 198
[spynnaker.pyNN.models.neuron.master_population](#) (module), 116
[spynnaker.pyNN.models.neuron.master_population](#) (module), 157

(module), 235	(class in spyn-	
spynnaker.pyNN.spynnaker_external_device_plugin_manager (module), 255	naker.pyNN.external_devices_models.push_bot.push_bot_etherne	
spynnaker.pyNN.spynnaker_simulator_interface (module), 257	SpynNakerNeuronGraphNetworkSpecificationReport (class in spyn-	
spynnaker.pyNN.utilities (module), 252	naker.pyNN.utilities.spynnaker_neuron_network_specification_re	
spynnaker.pyNN.utilities.constants (module), 248	250	
spynnaker.pyNN.utilities.extracted_data (module), 248	SpynnakerRangeDictionary (class in spyn-	
spynnaker.pyNN.utilities.fake_HBP_Portal_machine_utilities (module), 249	naker.pyNN.utilities.ranged), 246	
spynnaker.pyNN.utilities.random_stats (module), 245	SpynnakerRangeDictionary (class in spyn-	
spynnaker.pyNN.utilities.random_stats.abSpynnakerRangedList (class in spyn-	naker.pyNN.utilities.ranged.spynnaker_ranged_dict),	
spynnaker.pyNN.utilities.ranged (module), 246	245	
spynnaker.pyNN.utilities.ranged.spynnaker_ranged_spynnaker_ranged_list (class in spyn-	naker.pyNN.utilities.ranged.spynnaker_ranged_list),	
spynnaker.pyNN.utilities.ranged.spynnaker_ranged_spynnaker_ranged_list (module), 246	246	
spynnaker.pyNN.utilities.ranged.spynnaker_ranged_spynnaker_ranged_list (module), 246	SpynnakerSimulatorInterface (class in spyn-	
spynnaker.pyNN.utilities.reports (module), 249	naker.pyNN.spynnaker_simulator_interface),	
spynnaker.pyNN.utilities.running_stats (module), 249	257	
spynnaker.pyNN.utilities.spynnaker_connection_holder_generations (module), 249	SpynNakerSynapticMatrixReport (class in spyn-	
spynnaker.pyNN.utilities.spynnaker_failed_state (module), 249	naker.pyNN.utilities.spynnaker_synaptic_matrix_report),	
spynnaker.pyNN.utilities.spynnaker_neuron_network_external_devices_models.external_spinnaker_link_fp	250	
spynnaker.pyNN.utilities.spynnaker_synaptic_matrix_report (module), 250	attribute), 22	
spynnaker.pyNN.utilities.utility_calls (module), 250	start_resume_commands (spyn-	
spynnaker.spike_checker (module), 258	naker.pyNN.external_devices_models.ExternalFPGA	
SpYNNakerConnectionHolderGenerator (class in spyn-	naker.pyNN.external_devices_models.munich_spinnaker_link_ret	
naker.pyNN.utilities.spynnaker_connection_holder_generations (module), 249	attribute), 25	
SpynnakerDataSpecificationWriter (class in spyn-	start_resume_commands (spyn-	
naker.pyNN.overridepacman_functions.spynnaker_data_specification_writer),	naker.pyNN.external_devices_models.push_bot.abstract_push_bo	
234	start_resume_commands (spyn-	
SpynnakerException, 254	naker.pyNN.external_devices_models.push_bot.AbstractPushBot	
SpynnakerExternalDevicePluginManager (class in spyn-	attribute), 16	
naker.pyNN.spynnaker_external_device_plugin_manager),	start_resume_commands (spyn-	
255	naker.pyNN.external_devices_models.push_bot.push_bot_etherne	
SpynnakerFailedState (class in spyn-	attribute), 7	
naker.pyNN.utilities.spynnaker_failed_state),	start_resume_commands (spyn-	
249	naker.pyNN.external_devices_models.push_bot.push_bot_etherne	
SpynnakerLiveSpikesConnection	attribute), 8	
	start_resume_commands (spyn-	

[naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_motor_device.PushBotEthernetMotorDevice](#)
[attribute](#)), 8
[naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS](#)
 STATIC_MATRIX (spyn- [attribute](#)), 248
[naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_generator.conductance.MatrixGeneratorHD](#) (spyn-
[attribute](#)), 143
[naker.pyNN.models.utility_models.synapse_expander.synapse_expander](#)
 std() (spynaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats
[method](#)), 244
 std() (spynaker.pyNN.utilities.random_stats.AbstractRandomStats [naker.pyNN.models.neural_projections.delayed_application_edges](#)
[method](#)), 245
 STDP_MATRIX (spyn- [synapse_information](#) (spyn-
[naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_generator.conductance.MatrixGeneratorHD](#).DelayedApplicationEdge
[attribute](#)), 143
[naker.pyNN.models.utility_models.synapse_expander.synapse_expander](#)
 stop() (spynaker.pyNN.abstract_spinnaker_common.AbstractSpinnakerCommon
[method](#)), 253
 Struct (class in spyn- [attribute](#)), 81
[naker.pyNN.models.neuron.implementations](#)), [synapse_information](#) (spyn-
102
[naker.pyNN.models.neural_projections.projection_machine_edges](#)
 Struct (class in spyn- [attribute](#)), 82
[naker.pyNN.models.neuron.implementations.struct](#) [synapse_information](#) (spyn-
99
[naker.pyNN.models.neural_projections.ProjectionApplicationEdge](#)
 struct (spynaker.pyNN.models.neuron.implementations.abstract_standard_neuron_component.AbstractStandardNeuronComponent
[attribute](#)), 96
[synapse_information](#) (spyn-
 struct (spynaker.pyNN.models.neuron.implementations.AbstractStandardNeuronComponent
[attribute](#)), 101
[naker.pyNN.models.neural_projections.ProjectionMachineEdge](#)
 StructuralDynamics (class in spyn- SYNAPSE_PARAMS (spyn-
[naker.pyNN.models.neuron.synapse_dynamics](#)), [naker.pyNN.utilities.constants.POPULATION_BASED_REGIONS](#)
163
[attribute](#)), 248
 StructuralDynamics (class in spyn- [synapse_type](#) (spyn-
[naker.pyNN.models.neuron.synapse_dynamics.structural_dynamics](#)), [naker.pyNN.models.neural_projections.synapse_information.SynapseInformation](#)
145
[attribute](#)), 82
 structure (spynaker.pyNN.models.neuron.synapse_dynamics.structural_dynamics.StructuralDynamics (spyn-
[attribute](#)), 145
[naker.pyNN.models.neural_projections.SynapseInformation](#)
 structure (spynaker.pyNN.models.neuron.synapse_dynamics.StructuralDynamics
[attribute](#)), 163
 SynapseDynamicsStatic (class in spyn-
 structure (spynaker.pyNN.models.pynn_population_common.PyNNPopulationCommon
[attribute](#)), 232
159
 synapse_dynamics (spyn- SynapseDynamicsStatic (class in spyn-
[naker.pyNN.models.neural_projections.synapse_information.SynapseInformation](#) [naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics](#)
[attribute](#)), 82
146
 synapse_dynamics (spyn- SynapseDynamicsSTDP (class in spyn-
[naker.pyNN.models.neural_projections.SynapseInformation](#) [naker.pyNN.models.neuron.synapse_dynamics](#)),
[attribute](#)), 85
160
 synapse_dynamics (spyn- SynapseDynamicsSTDP (class in spyn-
[naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex](#) [naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics](#)
[attribute](#)), 196
147
 synapse_dynamics (spyn- SynapseDynamicsStructuralCommon
[naker.pyNN.models.neuron.AbstractPopulationVertex](#) (class in spyn-
[attribute](#)), 206
[naker.pyNN.models.neuron.synapse_dynamics](#)),
 synapse_dynamics (spyn- [attribute](#)), 163
[naker.pyNN.models.neuron.synaptic_manager.SynapticManager](#) [SynapseDynamicsStructuralCommon](#)
[attribute](#)), 200
(class in spyn-
 synapse_dynamics (spyn- [naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics](#)
[naker.pyNN.models.neuron.SynapticManager](#) [attribute](#)), 150
 SynapseDynamicsStructuralStatic

<code>(class in spynaker.pyNN.models.neuron.synapse_dynamics),</code> 166	<code>SynapseTypeDualExponential (class in spynaker.pyNN.models.neuron.synapse_types),</code> 179
<code>SynapseDynamicsStructuralStatic (class in spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static),</code> 153	<code>SynapseTypeDualExponential (class in spynaker.pyNN.models.neuron.synapse_types.synapse_type_dual_exp),</code> 179
<code>SynapseDynamicsStructuralSTDP (class in spynaker.pyNN.models.neuron.synapse_dynamics),</code> 168	<code>SynapseTypeExponential (class in spynaker.pyNN.models.neuron.synapse_types),</code> 181
<code>SynapseDynamicsStructuralSTDP (class in spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp),</code> 155	<code>SynapseTypeExponential (class in spynaker.pyNN.models.neuron.synapse_types.synapse_type_exponential),</code> 177
<code>SynapseInformation (class in spynaker.pyNN.models.neural_projections),</code> 85	<code>synaptic_data_update () (spynaker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp method),</code> 152
<code>SynapseInformation (class in spynaker.pyNN.models.neural_projections.synapse_information),</code> 82	<code>synaptic_data_update () (spynaker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralSTDP method),</code> 165
<code>SynapseIORowBased (class in spynaker.pyNN.models.neuron.synapse_io),</code> 171	<code>SYNAPTIC_MATRIX (spynaker.pyNN.utilities.constants.POPULATION_BASED_REGIONS attribute),</code> 248
<code>SynapseIORowBased (class in spynaker.pyNN.models.neuron.synapse_io.synapse_io_row_based),</code> 171	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.absolute attribute),</code> 129
<code>SynapseRowTooBigException,</code> 254	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.absolute attribute),</code> 134
<code>SynapseStructureWeightAccumulator (class in spynaker.pyNN.models.neuron.plasticity.stdp.synapse_structure_weight_accumulator),</code> 129	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 130
<code>SynapseStructureWeightAccumulator (class in spynaker.pyNN.models.neuron.plasticity.stdp.synapse_structure_weight_accumulator),</code> 128	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 131
<code>SynapseStructureWeightOnly (class in spynaker.pyNN.models.neuron.plasticity.stdp.synapse_structure_weight_only),</code> 128	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 132
<code>SynapseStructureWeightOnly (class in spynaker.pyNN.models.neuron.plasticity.stdp.synapse_structure_weight_only),</code> 128	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 133
<code>SynapseTypeAlpha (class in spynaker.pyNN.models.neuron.synapse_types),</code> 183	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 135
<code>SynapseTypeAlpha (class in spynaker.pyNN.models.neuron.synapse_types.synapse_type_alpha),</code> 173	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 136
<code>SynapseTypeDelta (class in spynaker.pyNN.models.neuron.synapse_types),</code> 182	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 136
<code>SynapseTypeDelta (class in spynaker.pyNN.models.neuron.synapse_types.synapse_type_delta),</code> 174	<code>synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.relative attribute),</code> 134

synaptic_structure (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 137

SynapticBlockGenerationException, 254

SynapticBlockReadException, 254

SynapticConfigurationException, 254

SynapticManager (class in spynaker.pyNN.models.neuron), 207

SynapticManager (class in spynaker.pyNN.models.neuron.synaptic_manager), 200

SynapticMaxIncomingAtomsSupportException, 255

synfire_multiple_lines_spike_checker() (in module spynaker.spike_checker), 258

synfire_spike_checker() (in module spynaker.spike_checker), 258

SYSTEM (spynaker.pyNN.utilities.constants.POPULATION_BASED_REFERENCE), 248

SYSTEM_REGION (spynaker.pyNN.external_devices_models.munich_spiral_neuron_model.attribute), 22

SYSTEM_REGION (spynaker.pyNN.external_devices_models.MunichMotorDevice.attribute), 30

SYSTEM_REGION (spynaker.pyNN.models.spike_source.spike_source_poisson_machine.attribute), 213

T

tau (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 133

tau (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 137

tau_ca2 (spynaker.pyNN.models.neuron.additional_inputs_additional_ca2_adaptive_attribute), 87

tau_ca2 (spynaker.pyNN.models.neuron.additional_inputs_additional_ca2_adaptive_attribute), 88

tau_m (spynaker.pyNN.models.neuron.neuron_models.neuron_model_attribute), 123

tau_m (spynaker.pyNN.models.neuron.neuron_models.NeuronModel.attribute), 126

tau_minus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 130

tau_minus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 132

tau_minus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 132

tau_minus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 135

tau_minus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 136

tau_minus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 134

tau_plus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 132

tau_plus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 132

tau_plus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 135

tau_plus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 136

tau_plus (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 134

tau_refrac (spynaker.pyNN.models.neuron.neuron_models.neuron_model_attribute), 123

tau_refrac (spynaker.pyNN.models.neuron.neuron_models.NeuronModel.attribute), 126

tau_syn_E (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 174

tau_syn_E (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 177

tau_syn_E (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 178

tau_syn_E (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeE.attribute), 185

tau_syn_E (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeE.attribute), 180

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 177

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 180

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 174

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 177

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.synapse_type_attribute), 178

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeE.attribute), 185

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeE.attribute), 180

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeE.attribute), 182

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeE.attribute), 186

tau_syn_E2 (spynaker.pyNN.models.neuron.synapse_types.SynapseTypeE.attribute), 189

tau_syn_E2 (spynaker.pyNN.models.neuron.plasticity.stdp.timing_dependence_attribute), 130

`tau_y` (`spynnaker.pyNN.models.neuron.plasticity.stdp.timing_dependence.TimingDependencePfisterSpikeTriplet` attribute), 135
`ThresholdTypeMaassStochastic` (class in `spynnaker.pyNN.models.neuron.threshold_types`), 188
`ThresholdTypeMaassStochastic` (class in `spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic`), 185
`ThresholdTypeMulticastDeviceControl` (class in `spynnaker.pyNN.external_devices_models`), 32
`ThresholdTypeMulticastDeviceControl` (class in `spynnaker.pyNN.external_devices_models.threshold_type_multicast_device_control`), 25
`ThresholdTypeStatic` (class in `spynnaker.pyNN.models.neuron.threshold_types`), 187
`ThresholdTypeStatic` (class in `spynnaker.pyNN.models.neuron.threshold_types.threshold_type_static`), 186
`time_between_send` (`spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_output_device.AbstractPushBotOutputDevice` attribute), 15
`time_between_send` (`spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotOutputDevice` attribute), 16
`time_scale_factor` (`spynnaker.pyNN.abstract_spinnaker_common.AbstractSpiNNakerCommon` attribute), 254
`time_scale_factor()` (`spynnaker.pyNN.spynnaker_external_device_plugin_manager.SpiNNakerExternalDevicePluginManager` static method), 257
`timed_commands` (`spynnaker.pyNN.external_devices_models.external_spinnaker_link_fpga_device.ExternalFPGADevice` attribute), 22
`timed_commands` (`spynnaker.pyNN.external_devices_models.ExternalFPGADevice` attribute), 29
`timed_commands` (`spynnaker.pyNN.external_devices_models.munich_spinnaker_link_prime_device.MunichRetinaDevice` attribute), 25
`timed_commands` (`spynnaker.pyNN.external_devices_models.MunichRetinaDevice` attribute), 32
`timed_commands` (`spynnaker.pyNN.external_devices_models.push_bot.abstract_push_bot_retina_device.AbstractPushBotRetinaDevice` attribute), 15
`timed_commands` (`spynnaker.pyNN.external_devices_models.push_bot.AbstractPushBotRetinaDevice` attribute), 16
`timed_commands` (`spynnaker.pyNN.external_devices_models.push_bot.push_bot_ethernet_push_bot_ethernet_laser_device.PushBotEthernetLaserDevice` attribute), 7

[attribute](#)), 190
[TRAFFIC_IDENTIFIER](#) ([spyn-](#)
[naker.pyNN.models.neuron.AbstractPopulationVertex](#)
[attribute](#)), 201
[translate_control_packet\(\)](#) ([spyn-](#)
[naker.pyNN.external_devices_models.abstract_ethernet_translator](#)
[method](#)), 17
[translate_control_packet\(\)](#) ([spyn-](#)
[naker.pyNN.external_devices_models.AbstractEthernetTranslator](#)
[method](#)), 27
[translate_control_packet\(\)](#) ([spyn-](#)
[naker.pyNN.external_devices_models.push_bot.push_bot_ethernet_translator.PushBotTranslator](#)
[method](#)), 9
[turn_off_sensor_reporting\(\)](#) ([spyn-](#)
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol](#)
[method](#)), 238
[turn_off_sensor_reporting\(\)](#) ([spyn-](#)
[naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
[method](#)), 243
[turn_off_sensor_reporting_key](#) ([spyn-](#)
[naker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol](#)
[attribute](#)), 238
[turn_off_sensor_reporting_key](#) ([spyn-](#)
[naker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
[attribute](#)), 243
U
[u_init](#) ([spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh](#)
[attribute](#)), 121
[u_init](#) ([spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh](#)
[attribute](#)), 125
[uart_id](#) ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol](#)
[attribute](#)), 238
[uart_id](#) ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
[attribute](#)), 243
[undelayed_max_bytes](#) ([spyn-](#)
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#)
[attribute](#)), 170
[undelayed_max_n_synapses](#) ([spyn-](#)
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#)
[attribute](#)), 170
[undelayed_max_words](#) ([spyn-](#)
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#)
[attribute](#)), 170
[UP_POLARITY](#) ([spyn-](#)
[naker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARefractoryDevice](#)
[attribute](#)), 21
[UP_POLARITY](#) ([spyn-](#)
[naker.pyNN.external_devices_models.ExternalFPGARefractoryDevice](#)
[attribute](#)), 29
[UP_POLARITY](#) ([spyn-](#)
[naker.pyNN.external_devices_models.munich_spinnaker_link_fpga_retina_device.MunichRetinaDevice](#)
[attribute](#)), 24
[UP_POLARITY](#) ([spyn-](#)
[naker.pyNN.external_devices_models.MunichRetinaDevice](#)
[attribute](#)), 32
[update_live_packet_gather_tracker\(\)](#)
[update_master_population_table\(\)](#) ([spyn-](#)
[naker.pyNN.models.neuron.master_pop_table_generators.abstract_master_pop_table_generator](#)
[method](#)), 115
[update_master_population_table\(\)](#) ([spyn-](#)
[naker.pyNN.models.neuron.master_pop_table_generators.master_pop_table_generator](#)
[method](#)), 117
[update_master_population_table\(\)](#) ([spyn-](#)
[naker.pyNN.models.neuron.master_pop_table_generators.MasterPopTableGenerator](#)
[method](#)), 119
[update_values\(\)](#) ([spyn-](#)
[naker.pyNN.external_devices_models.threshold_type_multicast_device.ThresholdTypeMulticastDevice](#)
[method](#)), 26
[update_values\(\)](#) ([spyn-](#)
[naker.pyNN.external_devices_models.ThresholdTypeMulticastDevice](#)
[method](#)), 26
[update_values\(\)](#) ([spyn-](#)
[naker.pyNN.models.neuron.additional_inputs.additional_input_ca2.AdditionalInputCa2](#)
[method](#)), 88
[update_values\(\)](#) ([spyn-](#)
[naker.pyNN.models.neuron.additional_inputs.AdditionalInputCa2](#)
[method](#)), 88
[u_init](#) ([spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izh.NeuronModelIzh](#)
[attribute](#)), 121
[u_init](#) ([spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzh](#)
[attribute](#)), 125
[uart_id](#) ([spynnaker.pyNN.protocols.munich_io_spinnaker_link_protocol.MunichIoSpiNNakerLinkProtocol](#)
[attribute](#)), 238
[uart_id](#) ([spynnaker.pyNN.protocols.MunichIoSpiNNakerLinkProtocol](#)
[attribute](#)), 243
[undelayed_max_bytes](#) ([spyn-](#)
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#)
[attribute](#)), 170
[undelayed_max_n_synapses](#) ([spyn-](#)
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#)
[attribute](#)), 170
[undelayed_max_words](#) ([spyn-](#)
[naker.pyNN.models.neuron.synapse_io.max_row_info.MaxRowInfo](#)
[attribute](#)), 170
[UP_POLARITY](#) ([spyn-](#)
[naker.pyNN.external_devices_models.external_spinnaker_link_fpga_retina_device.ExternalFPGARefractoryDevice](#)
[attribute](#)), 21
[UP_POLARITY](#) ([spyn-](#)
[naker.pyNN.external_devices_models.ExternalFPGARefractoryDevice](#)
[attribute](#)), 29
[UP_POLARITY](#) ([spyn-](#)
[naker.pyNN.external_devices_models.munich_spinnaker_link_fpga_retina_device.MunichRetinaDevice](#)
[attribute](#)), 24

`update_values()` (spyn- `update_weight()` (spyn- `update_weight()` (spyn-
`naker.pyNN.models.neuron.neuron_models.neuron_model_leaky_integrator_neuron_model_delay_afferent_machine_edge`
`method), 123` `method), 83` `method), 83`
`update_values()` (spyn- `update_weight()` (spyn- `update_weight()` (spyn-
`naker.pyNN.models.neuron.neuron_models.NeuronModelIzhikevich_naker.pyNN.models.neural_projections.projection_machine_edge`
`method), 125` `method), 82` `method), 82`
`update_values()` (spyn- `update_weight()` (spyn- `update_weight()` (spyn-
`naker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegratorAndElson_naker.pyNN.models.neural_projections.ProjectionMachineEdge`
`method), 126` `method), 85` `method), 85`
`update_values()` (spyn- `v_init(spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izhikevich`
`naker.pyNN.models.neuron.synapse_types.synapse_type_alpha.SynapseTypeAlpha`
`method), 174` `attribute), 122`
`update_values()` (spyn- `attribute), 122` `attribute), 122`
`naker.pyNN.models.neuron.synapse_types.synapse_type_delta.SynapseTypeDelta`
`method), 175` `attribute), 123`
`update_values()` (spyn- `v_init(spynnaker.pyNN.models.neuron.neuron_models.NeuronModelIzhikevich`
`naker.pyNN.models.neuron.synapse_types.synapse_type_dual_exponential.SynapseTypeDualExponential`
`method), 177` `attribute), 127`
`update_values()` (spyn- `attribute), 127` `attribute), 127`
`naker.pyNN.models.neuron.synapse_types.synapse_type_exponential.SynapseTypeExponential`
`method), 178` `attribute), 123`
`update_values()` (spyn- `v_reset(spynnaker.pyNN.models.neuron.neuron_models.NeuronModelLeakyIntegratorAndElson`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeAlpha` `attribute), 127`
`method), 185` `v_rest(spynnaker.pyNN.models.neuron.neuron_models.neuron_model_izhikevich`
`update_values()` (spyn- `attribute), 123` `attribute), 127`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDelta` `attribute), 127`
`method), 183` `attribute), 127`
`update_values()` (spyn- `v_thresh(spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeDualExponential` `attribute), 186`
`method), 180` `v_thresh(spynnaker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic`
`update_values()` (spyn- `attribute), 187` `attribute), 189`
`naker.pyNN.models.neuron.synapse_types.SynapseTypeExponential` `attribute), 189`
`method), 182` `attribute), 189`
`update_values()` (spyn- `v_thresh(spynnaker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic`
`naker.pyNN.models.neuron.threshold_types.threshold_type_maass_stochastic.ThresholdTypeMaassStochastic`
`method), 186` `validate_mars_kiss_64_seed()` (in module
`update_values()` (spyn- `spynnaker.pyNN.utilities.utility_calls), 252`
`naker.pyNN.models.neuron.threshold_types.threshold_type_static.ThresholdTypeStatic`
`method), 187` `spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats`
`update_values()` (spyn- `var()` (spynnaker.pyNN.utilities.random_stats.AbstractRandomStats
`naker.pyNN.models.neuron.threshold_types.ThresholdTypeMaassStochastic` `method), 245`
`method), 189` `variance(spynnaker.pyNN.utilities.running_stats.RunningStats`
`update_values()` (spyn- `attribute), 249`
`naker.pyNN.models.neuron.threshold_types.ThresholdTypeStatic` `attribute), 51`
`method), 188` `attribute), 51`
`update_weight()` (spyn- `verbose(spynnaker.pyNN.models.neural_projections.connectors.AbstractConnector`
`naker.pyNN.models.abstract_models.abstract_weight_updatable.AbstractWeightUpdatable`
`method), 37` `vertex_executable_suffix` (spyn-
`update_weight()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abstract_weight_updatable`
`naker.pyNN.models.abstract_models.AbstractWeightUpdatable` `attribute), 129`
`method), 40` `vertex_executable_suffix` (spyn-
`update_weight()` (spyn- `naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.AbstractWeightUpdatable`
`naker.pyNN.models.neural_projections.delay_afferent_machine_edges.DelayAfferentMachineEdge`
`method), 80` `method), 154`

weight_dynamics	(spyn-	139	
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon	(spyn-		
attribute), 165	naker.pyNN.models.neuron.synaptic_manager.SynapticManager		
weight_maximum	(spyn-	method), 200	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence			
attribute), 138	naker.pyNN.models.neuron.SynapticManager		
weight_maximum	(spyn-	method), 208	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence.AbstractWeightDependence	(spyn-		
attribute), 140	naker.pyNN.models.utility_models.delays.delay_extension_vertex.		
weight_maximum	(spyn-	method), 222	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive.WeightDependenceAdditive			
attribute), 138	naker.pyNN.models.utility_models.delays.DelayExtensionVertex		
weight_maximum	(spyn-	method), 226	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceAdditiveTriplet			
attribute), 139	naker.pyNN.models.neuron.plasticity.stdp.common.plasticity_help		
weight_maximum	(spyn-	127	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multiplicative.WeightDependenceMultiplicative			
attribute), 140	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.abs		
weight_maximum	(spyn-	method), 129	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive.WeightDependenceAdditive	(spyn-		
attribute), 141	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.Abs		
weight_maximum	(spyn-	method), 134	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive_triplet.WeightDependenceAdditiveTriplet	(spyn-		
attribute), 142	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
weight_maximum	(spyn-	method), 130	
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_multiplicative.WeightDependenceMultiplicative	(spyn-		
attribute), 141	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
weight_scale	(spyn-	method), 131	
naker.pyNN.models.neuron.abstract_population_vertex.AbstractPopulationVertex	(spyn-		
attribute), 196	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
weight_scale	(spyn-	method), 132	
naker.pyNN.models.neuron.AbstractPopulationVertex.write_parameters()	(spyn-		
attribute), 207	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
WeightDependenceAdditive (class in spyn-	method), 133		
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.write_parameters()	(spyn-		
140	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
WeightDependenceAdditive (class in spyn-	method), 133		
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.weight_dependence_additive),	(spyn-		
138	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
WeightDependenceAdditiveTriplet	method), 135		
(class in spyn- write_parameters())	(spyn-		
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.write_parameters()	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
141	method), 136		
WeightDependenceAdditiveTriplet write_parameters()	(spyn-		
(class in spyn- naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing			
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence_additive_triplet),			
139	write_parameters()	(spyn-	
WeightDependenceMultiplicative naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing			
(class in spyn- method), 134			
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.write_parameters()	(spyn-		
141	naker.pyNN.models.neuron.plasticity.stdp.timing_dependence.timing		
WeightDependenceMultiplicative method), 137			
(class in spyn- write_parameters())	(spyn-		
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abstract_weight_dependence_multiplicative),	naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.abs		

```

method), 138
write_parameters() (spyn- write_parameters() (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative.SynapseDynamicsStatic
method), 140 method), 169
write_parameters() (spyn- write_setup_info() (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative.SynapseDynamicsStatic
method), 138 method), 222
write_parameters() (spyn- write_setup_info() (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative.SynapseDynamicsStatic
method), 139 method), 226
write_parameters() (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative.SynapseDynamicsStatic
method), 140
write_parameters() (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative.SynapseDynamicsStatic
method), 141
write_parameters() (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative.SynapseDynamicsStatic
method), 142
write_parameters() (spyn-
naker.pyNN.models.neuron.plasticity.stdp.weight_dependence.multiplicative.SynapseDynamicsStatic
method), 141
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics
method), 145
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.abstract_synapse_dynamics.AbstractSynapseDynamics
method), 157
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_static.SynapseDynamicsStatic
method), 147
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_stdp.SynapseDynamicsSTDP
method), 149
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_common.SynapseDynamicsStructuralCommon
method), 152
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_static.SynapseDynamicsStructuralStatic
method), 154
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.synapse_dynamics_structural_stdp.SynapseDynamicsStructuralSTDP
method), 156
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStatic
method), 160
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsSTDP
method), 163
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralCommon
method), 165
write_parameters() (spyn-
naker.pyNN.models.neuron.synapse_dynamics.SynapseDynamicsStructuralStatic
method), 167

```