sPyNNaker Documentation

Release 6.0.0

Feb 22, 2023

Contents

These pages document the python code for the sPyNNaker module which is part of the SpiNNaker Project.

This code depends on SpiNNUtils, SpiNNMachine, SpiNNMan, PACMAN, DataSpecification, SpiNNFrontEndCommon (Combined_documentation).

Contents:

CHAPTER 1

spynnaker

1.1 spynnaker package

1.1.1 Subpackages

1.1.1.1 spynnaker.pyNN package

Subpackages

spynnaker.pyNN.connections package

Module contents

class spynnaker.pyNN.connections.**EthernetCommandConnection**(*translator*, *command_containers=None*, *local_host=None*, *local_port=19999*)

Bases: spinn_front_end_common.utilities.database.database_connection.DatabaseConnection

A connection that can send commands to a device at the start and end of a simulation

Parameters

- **translator** (*AbstractEthernetTranslator*) – A translator of multicast commands to device commands
- **command_containers** (*list(AbstractSendMeMulticastCommandsVertex)*) – A list of vertices that have commands to be sent at the start and end of simulation
- **local_host** (*str*) – The optional host to listen on for the start/resume message
- **local_port** (*int*) – The optional port to listen on for the stop/pause message

add_command_container(*command_container*)
Add a command container.

Parameters **command_container**(*AbstractSendMeMulticastCommandsVertex*) – A vertex that has commands to be sent at the start and end of simulation

class spynnaker.pyNN.connections.**EthernetControlConnection**(*translator*, *label*, *live_packet_gather_label*, *local_host=None*, *local_port=None*)

Bases: spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection

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

Parameters

- **translator** (*AbstractEthernetTranslator*) – The translator of multicast to control commands
- **label** (*str*) – The label of the vertex to attach the translator to
- **live_packet_gather_label** (*str*) – The label of the LPG vertex that this control connection will listen to.
- **local_host** (*str*) – The optional host to listen on
- **local_port** (*int*) – The optional port to listen on

add_translator(*label*, *translator*)
Add another translator that routes via the LPG.

Parameters

- **label** (*str*) – The label of the vertex to attach the translator to
- **translator** (*AbstractEthernetTranslator*) – The translator of multicast to control commands

class spynnaker.pyNN.connections.**SpynnakerLiveSpikesConnection**(*receive_labels=None*, *send_labels=None*, *local_host=None*, *local_port=19999*, *live_packet_gather_label='LiveSpikeRec*

Bases: spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection

A connection for receiving and sending live spikes from and to SpiNNaker

Parameters

- **receive_labels** (*iterable(str)*) – Labels of population from which live spikes will be received.
- **send_labels** (*iterable(str)*) – Labels of population to which live spikes will be sent
- **local_host** (*str*) – Optional specification of the local hostname or IP address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)

send_spike(*label*, *neuron_id*, *send_full_keys=False*)
Send a spike from a single neuron

Parameters

- **label** (*str*) – The label of the population from which the spike will originate
- **neuron_id** (*int*) – The ID of the neuron sending a spike
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each neuron from the database, or whether to send 16-bit neuron IDs directly

send_spikes(*label*, *neuron_ids*, *send_full_keys=False*)
Send a number of spikes

Parameters

- **label** (*str*) – The label of the population from which the spikes will originate
- **neuron_ids** (*list(int)*) – array-like of neuron IDs sending spikes
- **send_full_keys** (*bool*) – Determines whether to send full 32-bit keys, getting the key for each neuron from the database, or whether to send 16-bit neuron IDs directly

class spynnaker.pyNN.connections.**SpynnakerPoissonControlConnection**(*poisson_labels=None*, *local_host=None*, *local_port=19999*, *control_label_extension='_control'*)
Bases: spinn_front_end_common.utilities.connections.live_event_connection.LiveEventConnection

Parameters

- **poisson_labels** (*iterable(str)*) – Labels of Poisson populations to be controlled
- **local_host** (*str*) – Optional specification of the local hostname or IP address of the interface to listen on
- **local_port** (*int*) – Optional specification of the local port to listen on. Must match the port that the toolchain will send the notification on (19999 by default)
- **control_label_extension** (*str*) – The extra name added to the label of each Poisson source

add_init_callback(*label*, *init_callback*)
Add a callback to be called to initialise a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **init_callback** (*callable(str, int, float, float) -> None*) – A function to be called to initialise the vertex. This should take as parameters the label of the vertex, the number of neurons in the population, the run time of the simulation in milliseconds, and the simulation timestep in milliseconds

add_pause_stop_callback(*label*, *pause_stop_callback*)
Add a callback for the pause and stop state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **pause_stop_callback** (*callable(str, LiveEventConnection) -> None*) – A function to be called when the pause or stop message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events.

Return type None

add_poisson_label(*label*)

Parameters **label** (*str*) – The label of the Poisson source population.

add_receive_callback(*label*, *live_event_callback*, *translate_key=False*)
Add a callback for the reception of live events from a vertex

Parameters

- **label** (*str*) – The label of the vertex to be notified about. Must be one of the vertices listed in the constructor
- **live_event_callback** (*callable(str, int, list(int)) -> None*) – A function to be called when events are received. This should take as parameters the label of the vertex, the simulation timestep when the event occurred, and an array-like of atom IDs.
- **translate_key** (*bool*) – True if the key is to be converted to an atom ID, False if the key should stay a key

add_start_callback(*label*, *start_callback*)
Add a callback for the start of the simulation

Parameters

- **start_callback** (*callable(str, LiveEventConnection) -> None*) – A function to be called when the start message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events
- **label** (*str*) – the label of the function to be sent

add_start_resume_callback(*label*, *start_resume_callback*)
Add a callback for the start and resume state of the simulation

Parameters

- **label** (*str*) – the label of the function to be sent
- **start_resume_callback** (*callable(str, LiveEventConnection) -> None*) – A function to be called when the start or resume message has been received. This function should take the label of the referenced vertex, and an instance of this class, which can be used to send events.

Return type None

set_rate(*label*, *neuron_id*, *rate*)
Set the rate of a Poisson neuron within a Poisson source

Parameters

- **label** (*str*) – The label of the Population to set the rates of
- **neuron_id** (*int*) – The neuron ID to set the rate of
- **rate** (*float*) – The rate to set in Hz

set_rates(*label*, *neuron_id_rates*)
Set the rates of multiple Poisson neurons within a Poisson source

Parameters

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

spynnaker.pyNN.external_devices_models package

Subpackages

spynnaker.pyNN.external_devices_models.push_bot package

Subpackages

spynnaker.pyNN.external_devices_models.push_bot.control package

Module contents

spynnaker.pyNN.external_devices_models.push_bot.ethernet package

Module contents

spynnaker.pyNN.external_devices_models.push_bot.parameters package

Module contents

spynnaker.pyNN.external_devices_models.push_bot.spinnaker_link package

Module contents

Module contents

Module contents

spynnaker.pyNN.extra_algorithms package

Subpackages

spynnaker.pyNN.extra_algorithms.splitter_components package

Module contents

Module contents

spynnaker.pyNN.model_binaries package

Module contents

This module contains no python code.

spynnaker.pyNN.models package

Subpackages

spynnaker.pyNN.models.abstract_models package

Module contents

class spynnaker.pyNN.models.abstract_models.**AbstractAcceptsIncomingSynapses**
Bases: object

Indicates an application vertex that can be a post-vertex in a PyNN projection.

Note: See *verify_splitter()*

clear_connection_cache()
Clear the connection data stored in the vertex so far.

get_connections_from_machine(*transceiver*, *placements*, *app_edge*, *synapse_info*)
Get the connections from the machine post-run.

Parameters

- **transceiver** (*Transceiver*) – How to read the connection data
- **placements** (*Placements*) – Where the connection data is on the machine
- **app_edge** (*ProjectionApplicationEdge*) – The edge for which the data is being read
- **synapse_info** (*SynapseInformation*) – The specific projection within the edge

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

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

Return type int

set_synapse_dynamics(*synapse_dynamics*)
Set the synapse dynamics of this vertex.

Parameters **synapse_dynamics** (*AbstractSynapseDynamics*) –

verify_splitter(*splitter*)
Check that the splitter implements the API(s) expected by the SynapticMatrices

Any Vertex that implements this api should override ApplicationVertex.splitter method to also call this function

Parameters **splitter** (*AbstractSpynnakerSplitterDelay*) – the splitter

Raises **PacmanConfigurationException** – if the splitter is not an instance of AbstractSpynnakerSplitterDelay

class spynnaker.pyNN.models.abstract_models.**AbstractContainsUnits**
Bases: object

Indicates an application vertex class that can describe the units of some of its variables.

get_units(*variable*)
Get units for a given variable.

Parameters **variable** (*str*) – the variable to find units from

Returns the units as a string.

Return type str

class spynnaker.pyNN.models.abstract_models.**AbstractHasDelayStages**
Bases: object

Indicates that this object (an application vertex) has delay stages that are used to increase the space required for bitfields in spynnaker.pyNN.utilities.bit_field_utilities.get_estimated_sdram_for_bit_field_region()

n_delay_stages
The maximum number of delay stages required by any connection out of this delay extension vertex

Return type int

class spynnaker.pyNN.models.abstract_models.**AbstractMaxSpikes**
Bases: object

Indicates a class (a MachineVertex) that can describe the maximum rate that it sends spikes.

The SynapticManager assumes that all machine vertexes share the same synapse_information will have the same rates.

max_spikes_per_second()
Get maximum expected number of spikes per second

Parameters **variable** (*str*) – the variable to find units from

Returns the units as a string.

Return type str

max_spikes_per_ts(*machine_time_step*)
Get maximum expected number of spikes per timestep

Parameters **machine_time_step** (*int*) – The timestime used in ms

Return type int

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

Indicates that this application vertex 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** (*None or slice or int or list(bool) or list(int)*) – a description of the subrange to accept, or None for all. See: selector_to_ids()

Returns A list or an Object which act like a list

Return type iterable

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

Parameters **selector** (*None or slice or int or list(bool) or list(int)*) – a description of the subrange to accept, or None for all. See: selector_to_ids()

Return type dict(str,Any)

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

Return type dict(str,Any)

initialize(*variable*, *value*, *selector=None*)
Set the initial value of one of the state variables of the neurons in this population.

Parameters

- **variable** (*str*) – The name of the variable to set
- **value** (*float or int or Any*) – The value of the variable to set

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

Return type iterable(str)

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

Indicates that some properties of this application vertex 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** (*None or slice or int or list(bool) or list(int)*) – See get_value_by_selector() as this is just a pass through method
- **key** (*str*) – the name of the parameter to change

Return type list(float or int)

n_atoms
" See n_atoms()

set_value_by_selector(*selector*, *key*, *value*)
Sets the value for a particular key but only for the selected subset.

Parameters

- **selector** (*None or slice or int or list(bool) or list(int)*) – See RangedList.set_value_by_selector as this is just a pass through method
- **key** (*str*) – the name of the parameter to change
- **value** (*float or int or list(float) or list(int)*) – 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** (*Transceiver*) – the SpinnMan interface
- **placement** (*Placement*) – the placement of a vertex
- **vertex_slice** (*Slice*) – the slice of atoms for this vertex

Return type None

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

Parameters **key** (*str*) – the name of the property

Return type Any or float or int or list(float) or list(int)

set_value(*key*, *value*)
Set a property

Parameters

- **key** (*str*) – the name of the parameter to change
- **value** (*Any or float or int or list(float) or list(int)*) – the new value of the parameter to assign

class spynnaker.pyNN.models.abstract_models.**AbstractSynapseExpandable**
Bases: object

Indicates a class (a MachineVertex) that has may need to run the SYNAPSE_EXPANDER aplx

Cores that do not use the synapse_manager should not implement this API even though their app vertex may hold a synapse_manager.

Note: This is *not* implemented by the DelayExtensionMachineVertex, which needs a different expander aplx

---

gen_on_machine()
True if the synapses of a the slice of this vertex should be generated on the machine.

Note: The typical implementation for this method will be to ask the app_vertex's synapse_manager

Return type bool

read_generated_connection_holders(*transceiver*, *placement*)
Fill in the connection holders

Note: The typical implementation for this method will be to ask the app_vertex's synapse_manager

Parameters

- **transceiver** (*Transceiver*) – How the data is to be read
- **placement** (*Placement*) – Where the data is on the machine

class spynnaker.pyNN.models.abstract_models.**AbstractWeightUpdatable**
Bases: object

An object whose weight can be updated.

update_weight()
Update the weight.

spynnaker.pyNN.models.common package

Submodules

spynnaker.pyNN.models.common.recording_utils module

Module contents

spynnaker.pyNN.models.neural_projections package

Subpackages

spynnaker.pyNN.models.neural_projections.connectors package

Module contents

Module contents

spynnaker.pyNN.models.neural_properties package

Module contents

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

A settable parameter of a neuron model.

Parameters

- **value** (*int or float or bool or list(int) or list(float) or list(bool) or ndarray or AbstractList*) – what the value of the parameter is; if a list or array, potentially provides a different value for each neuron
- **data_type** (*DataType*) – The serialization type of the parameter in the neuron model.

get_dataspec_datatype()
Get the serialization type of the parameter in the neuron model.

Return type DataType

get_value()
What the value of the parameter is; if a list or array, potentially provides a different value for each neuron.

Return type int or float or bool or list(int) or list(float) or list(bool) or ndarray or AbstractList

iterator_by_slice(*slice_start*, *slice_stop*, *spec*)
Creates an iterator over the commands to use to write the parameter to the data specification being generated.

Parameters

- **slice_start** (*int*) – Inclusive start of the range
- **slice_stop** (*int*) – Exclusive end of the range
- **spec** (*DataSpecificationGenerator*) – The data specification to eventually write to. (Note that this does not actually do the write).

Returns Iterator that produces a command to write to the specification for each element in the slice.

Return type iterator(tuple(bytearray, str))

spynnaker.pyNN.models.neuron package

Subpackages

spynnaker.pyNN.models.neuron.additional_inputs package

Module contents

spynnaker.pyNN.models.neuron.builds package

Module contents

spynnaker.pyNN.models.neuron.implementations package

Module contents

spynnaker.pyNN.models.neuron.input_types package

Module contents

spynnaker.pyNN.models.neuron.neuron_models package

Module contents

spynnaker.pyNN.models.neuron.plasticity package

Subpackages

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

Subpackages

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

Module contents

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

Module contents

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

Module contents

Submodules

spynnaker.pyNN.models.neuron.plasticity.stdp.common module

Module contents

Module contents

spynnaker.pyNN.models.neuron.structural_plasticity package

Subpackages

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis package

Subpackages

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.elimination package

Module contents

spynnaker.pyNN.models.neuron.structural_plasticity.synaptogenesis.formation package

spynnaker.pyNN.models.spike_source.spike_source_array_vertex module

spynnaker.pyNN.models.spike_source.spike_source_poisson_machine_vertex module

spynnaker.pyNN.models.spike_source.spike_source_poisson_vertex module

Module contents

spynnaker.pyNN.models.utility_models package

Subpackages

spynnaker.pyNN.models.utility_models.delays package

Module contents

spynnaker.pyNN.models.utility_models.spike_injector package

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(AbstractConstraint) or None*) – A list of constraints to give to the vertex, or None

Returns An application vertex for the population

Return type 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, Any)

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** (*iterable(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** (*iterable(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*)
Get an argument initialisation dictionary by examining an __init__ method or function.

Parameters

- **init** (*callable*) – The method.
- **skip** (*frozenset(str)*) – The arguments to be skipped, if any
- **include** (*frozenset(str)*) – The arguments that must be present, if any

Returns an initialisation dictionary

Return type dict(str, Any)

spynnaker.pyNN.models.projection module

spynnaker.pyNN.models.recorder module

Module contents

spynnaker.pyNN.protocols package

Module contents

class spynnaker.pyNN.protocols.**MunichIoEthernetProtocol**

Bases: object

Implementation of the Munich robot IO protocol, communicating over ethernet.

static disable_motor()

static disable_retina()

static enable_motor()

static enable_retina()

static laser_active_time(*active_time*)

static laser_frequency(*frequency*)

static laser_total_period(*total_period*)

static led_back_active_time(*active_time*)

static led_frequency(*frequency*)

static led_front_active_time(*active_time*)

static led_total_period(*total_period*)

static motor_0_leaky_velocity(*velocity*)

static motor_0_permanent_velocity(*velocity*)

static motor_1_leaky_velocity(*velocity*)

static motor_1_permanent_velocity(*velocity*)

static set_retina_transmission(*event_format*)

static speaker_active_time(*active_time*)

static speaker_frequency(*frequency*)

static speaker_total_period(*total_period*)

class spynnaker.pyNN.protocols.**MunichIoSpiNNakerLinkProtocol**(*mode*, *instance_key=None*, *uart_id=0*)

Bases: object

Provides Multicast commands for the Munich SpiNNaker-Link protocol

Parameters

- **mode** – The mode of operation of the protocol
- **instance_key** (*int or None*) – The optional instance key to use

- **uart_id** (*int*) – The ID of the UART when needed

class MODES

Bases: enum.Enum

types of modes supported by this protocol

BALL_BALANCER = 3

FREE = 5

MY_ORO_BOTICS = 4

PUSH_BOT = 1

RESET_TO_DEFAULT = 0

SPOMNIBOT = 2

add_payload_logic_to_current_output(*payload*, *time=None*)

add_payload_logic_to_current_output_key

bias_values(*bias_id*, *bias_value*, *time=None*)

bias_values_key

configure_master_key(*new_key*, *time=None*)

configure_master_key_key

disable_retina(*time=None*)

disable_retina_key

enable_disable_motor_key

generic_motor0_raw_output_leak_to_0(*pwm_signal*, *time=None*)

generic_motor0_raw_output_leak_to_0_key

generic_motor0_raw_output_permanent(*pwm_signal*, *time=None*)

generic_motor0_raw_output_permanent_key

generic_motor1_raw_output_leak_to_0(*pwm_signal*, *time=None*)

generic_motor1_raw_output_leak_to_0_key

generic_motor1_raw_output_permanent(*pwm_signal*, *time=None*)

generic_motor1_raw_output_permanent_key

generic_motor_disable(*time=None*)

generic_motor_enable(*time=None*)

generic_motor_total_period(*time_in_ms*, *time=None*)

generic_motor_total_period_key

instance_key

The key of this instance of the protocol

Return type int

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

Return type *MODES*

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** (RetinaKey) – the new key for the retina
- **retina_payload** (*RetinaPayload or None*) – the new payload for the set retina key command packet
- **time** (*int or float or None*) – when to transmit this packet

Returns the command to send

Return type MultiCastCommand

set_retina_transmission_key

turn_off_sensor_reporting(*sensor_id*, *time=None*)

turn_off_sensor_reporting_key

uart_id

Return type int

class spynnaker.pyNN.protocols.**RetinaKey**(*value*, *pixels*, *bits_per_coordinate*)

Bases: enum.Enum

An enumeration.

DOWNSAMPLE_16_X_16 = 268435456

DOWNSAMPLE_32_X_32 = 201326592

DOWNSAMPLE_64_X_64 = 134217728

FIXED_KEY = 0

NATIVE_128_X_128 = 67108864

bits_per_coordinate

n_neurons

pixels

class spynnaker.pyNN.protocols.**RetinaPayload**(*value*, *n_payload_bytes*)

Bases: enum.Enum

An enumeration.

ABSOLUTE_2_BYTE_TIMESTAMPS = 1073741824

ABSOLUTE_3_BYTE_TIMESTAMPS = 1610612736

ABSOLUTE_4_BYTE_TIMESTAMPS = 2147483648

DELTA_TIMESTAMPS = 536870912

EVENTS_IN_PAYLOAD = 0

NO_PAYLOAD = 0

n_payload_bytes

spynnaker.pyNN.utilities package

Subpackages

spynnaker.pyNN.utilities.random_stats package

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsBinomialImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for binomial distributions

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsExponentialImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for exponential distributions

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsGammaImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for gamma distributions

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsLogNormalImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for log normal distributions

cdf(*dist*, *v*)
Return the cumulative distribution function value for the value v

high(*dist*)
Return the variance of the distribution

low(*dist*)
Return the variance of the distribution

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsNormalClippedImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for normal distributions that are clipped to a boundary (redrawn)

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsNormalImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for normal distributions

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsPoissonImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for poisson distributions

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsRandIntImpl**
Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for uniform distributions

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsScipyImpl**(*distribution_type*)

Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

A Random Statistics object that uses scipy directly

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsUniformImpl**

Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for uniform distributions

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

class spynnaker.pyNN.utilities.random_stats.**RandomStatsVonmisesImpl**

Bases: spynnaker.pyNN.utilities.random_stats.abstract_random_stats.AbstractRandomStats

An implementation of AbstractRandomStats for vonmises distributions

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

Module contents

class spynnaker.pyNN.utilities.ranged.**SpynnakerRangeDictionary**(*size*, *defaults=None*)

Bases: spinn_utilities.ranged.range_dictionary.RangeDictionary

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

Parameters

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

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

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

Parameters

- **size** (*int*) – 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

Return type *SpynnakerRangedList*

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** (*int or None*) – Fixed length of the list; if None, the value must be a sized object.
- **value** (*object or Sized*) – value to given to all elements in the list

- **key** – The dict key this list covers. This is used only for better Exception messages
- **use_list_as_value** (*bool*) – True if the value *is* a list

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

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

Parameters **value** –

Returns value as a list

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

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.bit_field_utilities module

spynnaker.pyNN.utilities.constants module

spynnaker.pyNN.utilities.constants.**LIVE_POISSON_CONTROL_PARTITION_ID = 'CONTROL'**
The partition ID used for Poisson live control data

spynnaker.pyNN.utilities.constants.**MIN_SUPPORTED_DELAY = 1**
the minimum supported delay slot between two neurons

spynnaker.pyNN.utilities.constants.**OUT_SPIKE_BYTES = 32**
The number of bytes for each spike line

spynnaker.pyNN.utilities.constants.**OUT_SPIKE_SIZE = 8**
The size of each output spike line

class spynnaker.pyNN.utilities.constants.**POPULATION_BASED_REGIONS**
Bases: enum.Enum

Regions for populations.

BIT_FIELD_BUILDER = 13

BIT_FIELD_FILTER = 12

BIT_FIELD_KEY_MAP = 14

CONNECTOR_BUILDER = 10

DIRECT_MATRIX = 11

NEURON_PARAMS = 1

NEURON_RECORDING = 7

POPULATION_TABLE = 3

PROFILING = 9

PROVENANCE_DATA = 8

STRUCTURAL_DYNAMICS = 6

SYNAPSE_DYNAMICS = 5

SYNAPSE_PARAMS = 2

SYNAPTIC_MATRIX = 4

SYSTEM = 0

spynnaker.pyNN.utilities.constants.**POP_TABLE_MAX_ROW_LENGTH = 256**
The maximum row length of the master population table

spynnaker.pyNN.utilities.constants.**SPIKE_PARTITION_ID = 'SPIKE'**
The partition ID used for spike data

spynnaker.pyNN.utilities.constants.**SYNAPTIC_ROW_HEADER_WORDS = 3**
Words: 2 for row length and number of rows and 1 for plastic region size (which might be 0)

spynnaker.pyNN.utilities.data_cache module

class spynnaker.pyNN.utilities.data_cache.**DataCache**(*label*, *description*, *segment_number*, *recording_start_time*, *t*)

Bases: object

Storage object to hold all the data to (re)create a Neo Segment

Note: Required because deep-copy does not work on neo Objects

Stores the Data shared by all variable types at the top level and holds a cache for the variable specific data.

Parameters

- **label** (*str*) – cache label
- **description** (*str or dict*) – cache description
- **segment_number** (*int*) – cache segment number
- **recording_start_time** (*float*) – when this cache was started in recording space.
- **t** (*float*) – time

description

get_data(*variable*)
Get the variable cache for the named variable

Parameters **variable** (*str*) – name of variable to get cache for

Returns The cache data, IDs, indexes and units

Return type *VariableCache*

has_data(*variable*)
Checks if data for a variable has been cached

Parameters **variable** (*str*) – Name of variable

Returns True if there is cached data

Return type bool

label

rec_datetime

recording_start_time

save_data(*variable*, *data*, *indexes*, *n_neurons*, *units*, *sampling_interval*)
Saves the data for one variable in this segment

Parameters

- **variable** (*str*) – name of variable data applies to
- **data** (*ndarray*) – raw data in sPyNNaker format
- **indexes** (*ndarray*) – population indexes for which data should be returned
- **n_neurons** (*int*) – Number of neurons in the population, regardless of if they where recording or not.
- **units** (*str*) – the units in which the data is
- **sampling_interval** (*float or int*) – The number of milliseconds between samples.

segment_number

t

variables
Provides a list of which variables data has been cached for

Return type Iterator (str)

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.

get(*projection*, *attribute*)
Allow getting data from a given projection and attribute

Parameters

- **projection** (*Projection*) – the projection data was extracted from
- **attribute** (*list(int) or tuple(int) or None*) – the attribute to retrieve

Returns the attribute data in a connection holder

Return type ConnectionHolder

set(*projection*, *attribute*, *data*)
Allow the addition of data from a projection and attribute.

Parameters

- **projection** (*Projection*) – the projection data was extracted from
- **attribute** (*list(int) or tuple(int) or None*) – the attribute to store
- **data** (*ConnectionHolder*) – attribute data in a connection holder

Return type None

spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider module

class spynnaker.pyNN.utilities.fake_HBP_Portal_machine_provider.**FakeHBPPortalMachineProvide**

Bases: object

create()

destroy()

get_machine_info()

wait_till_not_ready()

wait_until_ready()

spynnaker.pyNN.utilities.neo_compare module

spynnaker.pyNN.utilities.neo_compare.**compare_analogsignal**(*as1*, *as2*, *same_length=True*)

Compares two analogsignal Objects to see if they are the same

Parameters

- **as1** (*AnalogSignal*) – first analogsignal holding list of individual analogsignal Objects
- **as2** (*AnalogSignal*) – second analogsignal holding list of individual analogsignal Objects
- **same_length** (*bool*) – Flag to indicate if the same length of data is held, i.e., all spikes up to the same time. If False allows one trains to have additional data after the first ends. This is used to compare data extracted part way with data extracted at the end.

Raises **AssertionError** – If the analogsignals are not equal

spynnaker.pyNN.utilities.neo_compare.**compare_blocks**(*neo1*, *neo2*, *same_runs=True*, *same_data=True*, *same_length=True*)

Compares two neo Blocks to see if they hold the same data.

Parameters

- **neo1** (*Block*) – First block to check
- **neo2** (*Block*) – Second block to check
- **same_runs** (*bool*) – Flag to signal if blocks are the same length. If False extra segments in the larger block are ignored
- **same_data** (*bool*) – Flag to indicate if the same type of data is held, i.e., same spikes, v, gsyn_exc and gsyn_inh. If False only data in both blocks is compared

- **same_length** (*bool*) – Flag to indicate if the same length of data is held, i.e., all spikes up to the same time. If False allows one trains to have additional data after the first ends. This is used to compare data extracted part way with data extracted at the end.

Raises **AssertionError** – If the blocks are not equal

spynnaker.pyNN.utilities.neo_compare.**compare_segments**(*seg1*, *seg2*, *same_data=True*, *same_length=True*)

Parameters

- **seg1** (*Segment*) – First Segment to check
- **seg2** (*Segment*) – Second Segment to check
- **same_data** (*bool*) – Flag to indicate if the same type of data is held, i.e., same spikes, v, gsyn_exc and gsyn_inh. If False only data in both blocks is compared
- **same_length** (*bool*) – Flag to indicate if the same length of data is held, i.e., all spikes up to the same time. If False allows one trains to have additional data after the first ends. This is used to compare data extracted part way with data extracted at the end.

Raises **AssertionError** – If the segments are not equal

spynnaker.pyNN.utilities.neo_compare.**compare_spiketrain**(*spiketrain1*, *spiketrain2*, *same_length=True*)

Checks two Spiketrains have the exact same data

Parameters

- **spiketrain1** (*SpikeTrain*) – first spiketrain
- **spiketrain2** (*SpikeTrain*) – second spiketrain
- **same_length** (*bool*) – Flag to indicate if the same length of data is held, i.e., all spikes up to the same time. If False allows one trains to have additional spikes after the first ends. This is used to compare data extracted part way with data extracted at the end.

Return type None

Raises **AssertionError** – If the spiketrains are not equal

spynnaker.pyNN.utilities.neo_compare.**compare_spiketrains**(*spiketrains1*, *spiketrains2*, *same_data=True*, *same_length=True*)

Check two Lists of SpikeTrains have the exact same data

Parameters

- **spiketrains1** (*list(SpikeTrain)*) – First list SpikeTrains to compare
- **spiketrains2** (*list(SpikeTrain)*) – Second list of SpikeTrains to compare
- **same_data** (*bool*) – Flag to indicate if the same type of data is held, i.e., same spikes, v, gsyn_exc and gsyn_inh. If False allows one or both lists to be Empty. Even if False none empty lists must be the same length
- **same_length** (*bool*) – Flag to indicate if the same length of data is held, i.e., all spikes up to the same time. If False allows one trains to have additional spikes after the first ends. This is used to compare data extracted part way with data extracted at the end.

Raises **AssertionError** – If the spiketrains are not equal

spynnaker.pyNN.utilities.neo_convertor module

spynnaker.pyNN.utilities.neo_convertor.**convert_analog_signal**(*signal_array*, *time_unit=<sphinx.ext.autodoc.importer._M object>*)

Converts part of a NEO object into told spynnaker7 format

Parameters

- **signal_array** (*AnalogSignal*) – Extended Quantities object
- **time_unit** (*quantities.unitquantity.UnitTime*) – Data time unit for time index

Return type ndarray

spynnaker.pyNN.utilities.neo_convertor.**convert_data**(*data*, *name*, *run=0*)

Converts the data into a numpy array in the format ID, time, value

Parameters

- **data** (*Block*) – Data as returned by a getData() call
- **name** (*str*) – Name of the data to be extracted. Same values as used in getData()
- **run** (*int*) – Zero based index of the run to extract data for

Return type ndarray

spynnaker.pyNN.utilities.neo_convertor.**convert_data_list**(*data*, *name*, *runs=None*)

Converts the data into a list of numpy arrays in the format ID, time, value

Parameters

- **data** (*Block*) – Data as returned by a getData() call
- **name** (*str*) – Name of the data to be extracted. Same values as used in getData()
- **runs** (*list(int) or None*) – List of Zero based index of the run to extract data for. Or None to extract all runs

Return type list(ndarray)

spynnaker.pyNN.utilities.neo_convertor.**convert_gsyn**(*gsyn_exc*, *gsyn_inh*)

Converts two neo objects into the spynnaker7 format

Note: It is acceptable for both neo parameters to be the same object

Parameters

- **gsyn_exc** (*Block*) – neo with gsyn_exc data
- **gsyn_inh** (*Block*) – neo with gsyn_exc data

Return type ndarray

spynnaker.pyNN.utilities.neo_convertor.**convert_gsyn_exc_list**(*data*, *runs=None*)

Converts the gsyn_exc into a list numpy array one per segment (all runs) in the format ID, time, value

Parameters

- **data** (*Block*) – The data to convert; it must have Gsyn_exc data in it

- **runs** (*list(int) or None*) – List of Zero based index of the run to extract data for. Or None to extract all runs

Return type list(ndarray)

spynnaker.pyNN.utilities.neo_convertor.**convert_gsyn_inh_list**(*data*, *runs=None*)
Converts the gsyn_inh into a list numpy array one per segment (all runs) in the format ID, time, value

Parameters

- **data** (*Block*) – The data to convert; it must have Gsyn_inh data in it
- **runs** (*list(int) or None*) – List of Zero based index of the run to extract data for. Or None to extract all runs

Return type list(ndarray)

spynnaker.pyNN.utilities.neo_convertor.**convert_spikes**(*neo*, *run=0*)
Extracts the spikes for run one from a Neo Object

Parameters

- **neo** (*Block*) – neo Object including Spike Data
- **run** (*int*) – Zero based index of the run to extract data for

Return type ndarray

spynnaker.pyNN.utilities.neo_convertor.**convert_spiketrains**(*spiketrains*)
Converts a list of spiketrains into spynnaker7 format

Parameters **spiketrains** (*list(SpikeTrain)*) – List of SpikeTrains

Return type ndarray

spynnaker.pyNN.utilities.neo_convertor.**convert_v_list**(*data*, *runs=None*)
Converts the voltage into a list numpy array one per segment (all runs) in the format ID, time, value

Parameters

- **data** (*Block*) – The data to convert; it must have V data in it
- **runs** (*list(int) or None*) – List of Zero based index of the run to extract data for. Or None to extract all runs

Return type list(ndarray)

spynnaker.pyNN.utilities.neo_convertor.**count_spikes**(*neo*)
Help function to count the number of spikes in a list of spiketrains

Only counts run 0

Parameters **neo** (*Block*) – Neo Object which has spikes in it

Returns The number of spikes in the first segment

spynnaker.pyNN.utilities.neo_convertor.**count_spiketrains**(*spiketrains*)
Help function to count the number of spikes in a list of spiketrains

Parameters **spiketrains** (*list(SpikeTrain)*) – List of SpikeTrains

Returns Total number of spikes in all the spiketrains

Return type int

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*)
Adds an item to the running statistics.

Parameters **x** (*int or float*) – The item to add

add_items(*mean*, *variance*, *n_items*)
Add a bunch of items (via their statistics).

Parameters

- **mean** (*float*) – The mean of the items to add.
- **variance** (*float*) – The variance of the items to add.
- **n_items** (*int*) – The number of items represented.

mean
The mean of the items seen.

Return type float

n_items
The number of items seen.

Return type int

standard_deviation
The population standard deviation of the items seen.

Return type float

variance
The variance of the items seen.

Return type float

spynnaker.pyNN.utilities.spynnaker_failed_state module

spynnaker.pyNN.utilities.struct module

class spynnaker.pyNN.utilities.struct.**Struct**(*field_types*)
Bases: object

Represents a C code structure.

Parameters **field_types** (*list(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(DataType)

get_data(*values*, *offset=0*, *array_size=1*)
Get a numpy array of uint32 of data for the given values

Parameters

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

Return type ndarray(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** (*int*) – The number of elements in an array of structs

Return type int

numpy_dtype
The numpy data type of the struct

Return type dtype

read_data(*data*, *offset=0*, *array_size=1*)
Read a bytearray of data and convert to struct values

Parameters

- **data** (*bytes or bytearray*) – The data to be read
- **offset** (*int*) – Index of the byte at the start of the valid data
- **array_size** (*int*) – The number of struct elements to read

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

Return type list(float)

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** (*str*) – The file whose parent directory is to be created

spynnaker.pyNN.utilities.utility_calls.**convert_param_to_numpy**(*param*, *no_atoms*)
Convert parameters into numpy arrays.

Parameters

- **param** (*NumpyRNG or int or float or list(int) or list(float) or ndarray*) – the param to convert
- **no_atoms** (*int*) – the number of atoms available for conversion of param

Returns the converted param as an array of floats

Return type ndarray(float)

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** (*DataType*) – The data type to convert to

Returns The converted data as a numpy data type

Return type ndarray(int32)

spynnaker.pyNN.utilities.utility_calls.**create_mars_kiss_seeds**(*rng*, *seed=None*)
generates and checks that the seed values generated by the given random number generator or seed to a random number generator are suitable for use as a mars 64 kiss seed.

Parameters

- **rng** (*None or RandomState*) – the random number generator.
- **seed** (*int or None*) – the seed to create a random number generator if not handed.

Returns a list of 4 ints which are used by the mars64 kiss random number generator for seeds.

Return type list(int)

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

Parameters **n_values** (*int*) – the number of values (starting at 0)

Returns the number of bits required to express that many values

Return type int

spynnaker.pyNN.utilities.utility_calls.**get_probability_within_range**(*dist*, *lower*, *upper*)
Get the probability that a value will fall within the given range for a given RandomDistribution

spynnaker.pyNN.utilities.utility_calls.**get_probable_maximum_selected**(*n_total_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_probable_minimum_selected**(*n_total_trials*, *n_trials*, *selection_prob*, *chance=0.01*)
Get the likely minimum 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.**moved_in_v6**(*old_location*, *new_location*)
Warns the users that they are using an old import.

In version 7 this will ne upgraded to a exception and then later removed

Parameters

- **old_location** (*str*) – old import
- **new_location** (*str*) – new import

Raise an exception if in CONTINUOUS_INTEGRATION

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** (*str*) – absolute path to a file containing the data
- **min_atom** (*int*) – min neuron ID to which neurons to read in
- **max_atom** (*int*) – max neuron ID to which neurons to read in
- **extra** –
- **min_time** (*float or int*) – min time slot to read neurons values of.
- **max_time** (*float or int*) – max time slot to read neurons values of.

Returns a numpy array of (time stamp, atom ID, data value)

Return type ndarray(tuple(float, int, float))

spynnaker.pyNN.utilities.utility_calls.**read_spikes_from_file**(*file_path*, *min_atom=0*, *max_atom=inf*, *min_time=0*, *max_time=inf*, *split_value='\t'*)

Read spikes from a file formatted as: <time> <neuron ID>

Parameters

- **file_path** (*str*) – absolute path to a file containing spike values
- **min_atom** (*int or float*) – min neuron ID to which neurons to read in
- **max_atom** (*int or float*) – max neuron ID to which neurons to read in
- **min_time** (*float or int*) – min time slot to read neurons values of.
- **max_time** (*float or 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.ndarray(int, int)

spynnaker.pyNN.utilities.variable_cache module

class spynnaker.pyNN.utilities.variable_cache.**VariableCache**(*data*, *indexes*, *n_neurons*, *units*, *sampling_interval*)

Bases: object

Simple holder method to keep data, IDs, indexes and units together

Typically used to recreate the Neo object for one type of variable for one segment.

Parameters

- **data** (*ndarray*) – raw data in sPyNNaker format
- **indexes** (*list(int)*) – Population indexes for which data was collected
- **n_neurons** (*int*) – Number of neurons in the population, regardless of whether they were recording or not.
- **units** (*str*) – the units in which the data is
- **sampling_interval** (*float or int*) – The number of milliseconds between samples.

data

Return type ndarray

indexes

Return type list(int)

n_neurons

Return type int

sampling_interval

Return type float or int

units

Return type str

Module contents

Submodules

spynnaker.pyNN.abstract_spinnaker_common module

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.**SpynnakerSplitterConfigurationException**
Bases: spinn_front_end_common.utilities.exceptions.ConfigurationException

Raised when a splitter configuration fails.

exception spynnaker.pyNN.exceptions.**SynapseRowTooBigException**(*max_size*, *message*)
Bases: *spynnaker.pyNN.exceptions.SpynnakerException*

Raised when a synapse row is bigger than is allowed.

Parameters

- **max_size** – the maximum permitted size of row
- **message** – the excepton message

max_size
The maximum size allowed.

exception spynnaker.pyNN.exceptions.**SynapticBlockGenerationException**
Bases: spinn_front_end_common.utilities.exceptions.ConfigurationException

Raised when the synaptic manager fails to generate a synaptic block.

exception spynnaker.pyNN.exceptions.**SynapticBlockReadException**
Bases: spinn_front_end_common.utilities.exceptions.ConfigurationException

Raised when the synaptic manager fails to read a synaptic block or convert it into readable values.

exception spynnaker.pyNN.exceptions.**SynapticConfigurationException**
Bases: spinn_front_end_common.utilities.exceptions.ConfigurationException

Raised when the synaptic manager fails for some reason.

exception spynnaker.pyNN.exceptions.**SynapticMaxIncomingAtomsSupportException**
Bases: spinn_front_end_common.utilities.exceptions.ConfigurationException

Raised when a synaptic sublist exceeds the max atoms possible to be supported.

spynnaker.pyNN.spynnaker_external_device_plugin_manager module

spynnaker.pyNN.spynnaker_simulator_interface module

Module contents

1.1.2 Submodules

1.1.3 spynnaker.gsyn_tools module

spynnaker.gsyn_tools.**check_gsyn**(*gsyn1*, *gsyn2*)
Compare two arrays of conductances. For testing.

Parameters

- **gsyn1** – An array of conductances.
- **gsyn2** – An array of conductances.

Raises **Exception** – If the arrays differ.

spynnaker.gsyn_tools.**check_path_gysn**(*path*, *n_neurons*, *runtime*, *gsyn*)
Compare an arrays of conductances with baseline data from a file. For testing.

Parameters

- **path** – A file path.
- **n_neurons** – The number of neurons that produced the data.
- **runtime** – The length of time that the generated data represents.
- **gsyn** – An array of conductances.

Raises **Exception** – If the arrays differ.

spynnaker.gsyn_tools.**check_sister_gysn**(*sister*, *n_neurons*, *runtime*, *gsyn*)
Compare an arrays of conductances with baseline data from a file next to a specified module. For testing.

Parameters

- **sister** – A module. The file read from will be gsyn.data adjacent to this module.
- **n_neurons** – The number of neurons that produced the data.
- **runtime** – The length of time that the generated data represents.
- **gsyn** – An array of conductances.

Raises **Exception** – If the arrays differ.

1.1.4 spynnaker.plot_utils module

spynnaker.plot_utils.**heat_plot**(*data_sets*, *ylabel=None*, *title=None*)
Build a heatmap plot or plots.

Parameters

- **data_sets** (*ndarray or list(ndarray)*) – Numpy array of data, or list of numpy arrays of data
- **ylabel** (*str or None*) – The label for the Y axis
- **title** (*str or None*) – The title for the plot

spynnaker.plot_utils.**line_plot**(*data_sets*, *title=None*)
Build a line plot or plots.

Parameters

- **data_sets** (*ndarray or list(ndarray)*) – Numpy array of data, or list of numpy arrays of data
- **title** (*str or None*) – The title for the plot

spynnaker.plot_utils.**plot_spikes**(*spikes*, *title='spikes'*)
Build a spike plot or plots.

Parameters

- **spikes** (*ndarray or list(ndarray)*) – Numpy array of spikes, or list of numpy arrays of spikes
- **title** (*str*) – The title for the plot

1.1.5 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** (*ndarray or list(ndarray)*) – The spikes
- **nNeurons** (*int*) – The number of neurons.
- **lines** (*int*) – Expected number of lines
- **wrap_around** (*bool*) – If True the lines will wrap around when reaching the last neuron.

Raises **Exception** – If there is a problem with the data

spynnaker.spike_checker.**synfire_spike_checker**(*spikes*, *nNeurons*)

Parameters

- **spikes** (*ndarray or list(ndarray)*) – The spike data to check.
- **nNeurons** (*int*) – The number of neurons.

Raises **Exception** – If there is a problem with the data

1.1.6 Module contents

CHAPTER 2

spynnaker8

2.1 spynnaker8 package

2.1.1 Subpackages

2.1.1.1 spynnaker8.external_devices package

Module contents

2.1.1.2 spynnaker8.extra_models package

Module contents

2.1.1.3 spynnaker8.models package

Subpackages

spynnaker8.models.connectors package

Module contents

spynnaker8.models.populations package

Module contents

spynnaker8.models.synapse_dynamics package

Subpackages

spynnaker8.models.synapse_dynamics.timing_dependence package

Module contents

spynnaker8.models.synapse_dynamics.weight_dependence package

Module contents

Module contents

Module contents

2.1.1.4 spynnaker8.utilities package

Submodules

spynnaker8.utilities.neo_convertor module

Module contents

2.1.2 Submodules

2.1.3 spynnaker8.spynnaker_plotting module

2.1.4 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W