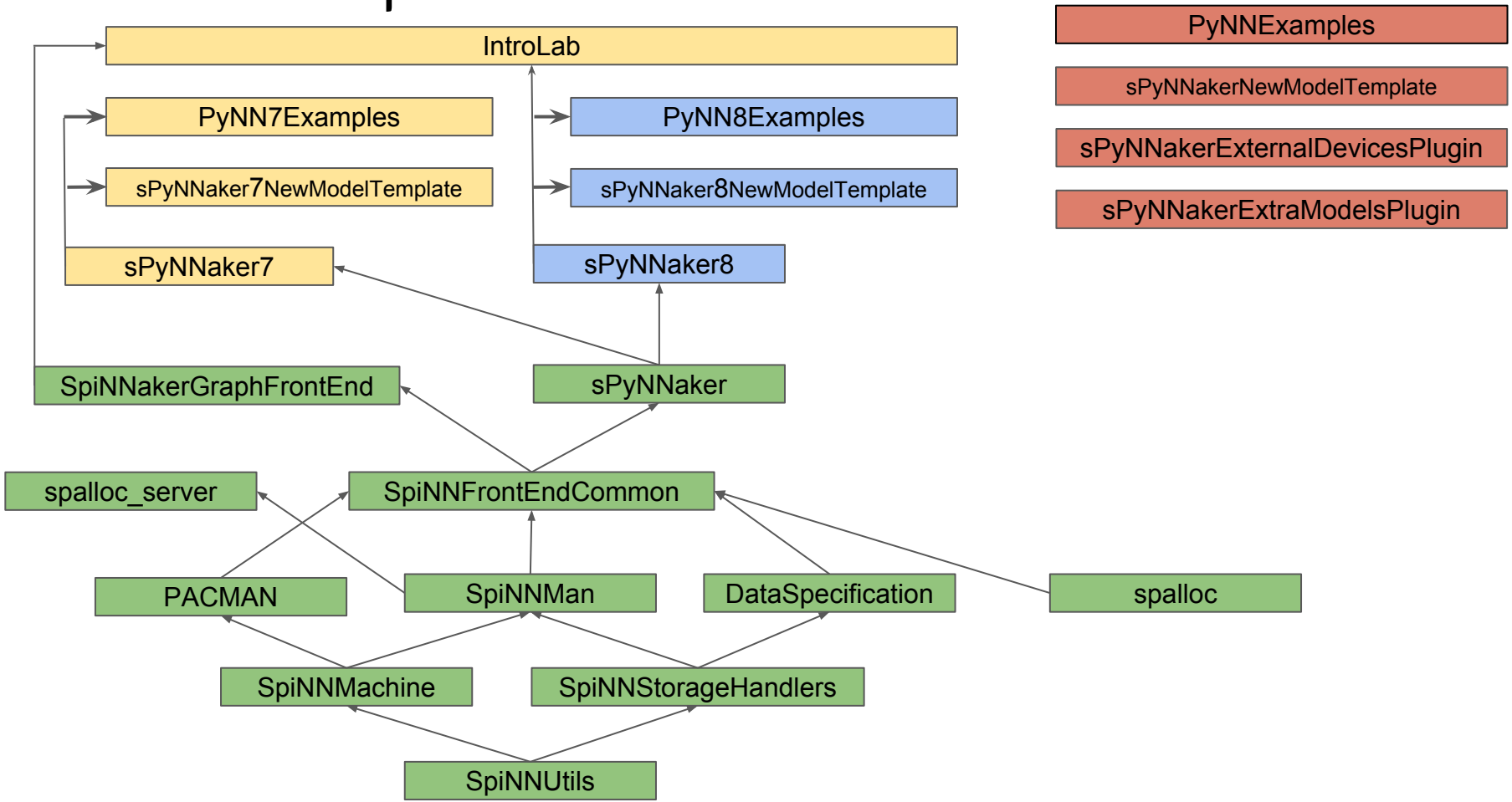
PyNN 0.8
vs
PyNN 0.75

Alan B Stokes &
Christian Brenninkmeijer &
Andy Gait &
Andrew Davidson

PyNN 0.8

1. Not backwards compatible with pynn 0.75
2. Not compatible with python 2.5 or backwards.
3. Uses NEO!
4. SpiNNaker software now has a PyNN 0.8 interface.
5. SpiNNaker software will cease software support for PyNN 0.75 in the near future.

How our software tools look now



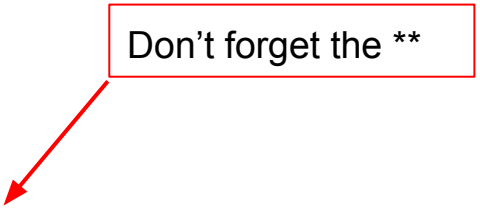
Populations

PyNN 0.75

```
import spynnaker7.pyNN as p # (or pynn.spinnaker)
cell_params_lif = {'cm': 0.25}
p1 = p.Population(nNeurons, p.IF_curr_exp, cell_params_lif, label='pop_1')
```

PyNN 0.8

```
import spynnaker8 as p
cell_params_lif = {'cm': 0.25}
p1 = p.Population(nNeurons, p.IF_curr_exp(**cell_params_lif), label='pop_1')
p2 = p.Population(nNeurons, p.IF_curr_exp(cm=0.25), label='pop_1')
P3 = p.Population(nNeurons, p.IF_curr_exp, cell_params_lif, label='pop_1')
```



Population Synapse type extra components

PyNN 0.75

```
cell_params_lif = {'cm': 0.25}
p1 = p.Population(nNeurons, p.IF_curr_exp, cell_params_lif, label='pop_1')
```

PyNN 0.8

```
cell_params_lif = {'cm': 0.25, 'isyn_exc': 0.1, 'isyn_inh': 0.2}
cell_params_lif2 = {'cm': 0.25}
initial_params = {'isyn_exc': 0.1, 'isyn_inh': 0.2}
p1 = p.Population(nNeurons, p.IF_curr_exp(**cell_params_lif), label='pop_1')
p2 = p.Population(nNeurons, p.IF_curr_exp(**cell_params_lif2), initial_values=initial_params)
```

Population model renamed

A few model objects have been renamed by PyNN

PyNN 7 name	PyNN 8 name
izk_cond_exp	Izhikevich_cond
izk_curr_exp	Izhikevich

Effect on PyNN models

PyNN 0.75

1. Models with synapse types need to hand down the **isyn_exc**, **isyn_inh** values down to the models synapse type.
2. All models and components have default parameters which need to be set.
3. All models need getters and setters for the isyn_exc and isyn_inh

PyNN 0.8

1. Need to build a **DataHolder** class which contains the parameters and points to the PyNN 0.75 model to build when needed.

NOTE: all models in master sPyNNaker7, sPyNNakerExtraModelsPlugin and sPyNNaker7/8NewModelTemplate have been updated. Branches have not been updated.

Data Holder

```
from spynnaker.pyNN.models.neuron.abstract_population_vertex import AbstractPopulationVertex
from spynnaker8.utilities.data_holder import DataHolder
from spynnaker.pyNN.models.neuron.builds.if_curr_exp_base import IFCurrExpBase


class IFCurrExpDataHolder(DataHolder):
  def __init__(
        self,  spikes_per_second=AbstractPopulationVertex.none_pynn_default_parameters['spikes_per_second'],
        ring_buffer_sigma=AbstractPopulationVertex.none_pynn_default_parameters['ring_buffer_sigma'],
        incoming_spike_buffer_size=AbstractPopulationVertex.none_pynn_default_parameters['incoming_spike_buffer_size'],
        constraints=AbstractPopulationVertex.none_pynn_default_parameters['constraints'],
        label=AbstractPopulationVertex.none_pynn_default_parameters['label'], v_init=IFCurrExpBase.none_pynn_default_parameters['v_init'],
        tau_m=IFCurrExpBase.default_parameters['tau_m'],  cm=IFCurrExpBase.default_parameters['cm'],
        v_rest=IFCurrExpBase.default_parameters['v_rest'],  v_reset=IFCurrExpBase.default_parameters['v_reset'],
        v_thresh=IFCurrExpBase.default_parameters['v_thresh'],  tau_syn_E=IFCurrExpBase.default_parameters['tau_syn_E'],
        tau_syn_I=IFCurrExpBase.default_parameters['tau_syn_I'],  tau_refrac=IFCurrExpBase.default_parameters['tau_refrac'],
        i_offset=IFCurrExpBase.default_parameters['i_offset']):

      DataHolder.__init__(
        self, {
          'spikes_per_second': spikes_per_second, 'ring_buffer_sigma': ring_buffer_sigma,
          'incoming_spike_buffer_size': incoming_spike_buffer_size,  'constraints': constraints,
          'label': label, 'tau_m': tau_m, 'cm': cm, 'v_rest': v_rest, 'v_reset': v_reset, 'v_thresh': v_thresh,
          'tau_syn_E': tau_syn_E, 'tau_syn_I': tau_syn_I,  'tau_refrac': tau_refrac, 'i_offset': i_offset,  'v_init': v_init})

  @staticmethod
  def build_model():
    return IFCurrExpBase
```

Population setting parameters

SpiNNaker

PyNN 0.75

```
p1 = p.Population(nNeurons, p.IF_curr_exp, cell_params_lif, label='pop_1')
p1.set(tau_m, 20.0)
p1.rset(tau_m, rng)
p1.tset(tau_m, array)
p1.initialize(v, 20.0)
p1.randomInit(v)
```

PyNN 0.8

```
p1 = p.Population(nNeurons, p.IF_curr_exp(**cell_params_lif), label='pop_1')
p1.set(tau_m=20.0, v_rest=array, v_reset=rng)
p1.initialize(v=20.0)
```

Record stuff

SpiNNaker

PyNN 0.75

```
p1 = p.Population(nNeurons, p.IF_curr_exp, cell_params_lif, label='pop_1'))
p1.record_v()
p1.record_gsyn()
p1.record()
```

PyNN 0.8

```
p2 = p.Population(nNeurons, p.IF_curr_exp(cm=0.25), label='pop_1')
p2.record("v")
p2.record(["v", "gsyn_exc", "gsyn_inh", "spikes"])
p2.record("all")
p2.record() <-------- means reset! (currently not supported but will be soon)
```

Run

PyNN 0.75

```
p.run(5000)
```

PyNN 0.8

```
p.run(300.0)
p.run(300.0, callbacks=[report_time])
p.run_until(600.0) <------- Absolute timing
p.run_for(600.0)
```

Get recorded Data

PyNN 0.75

```
v = p1.get_v(compatible_output=True)
gsyn = p1.get_gsyn(compatible_output=True)
spikes = p1.getSpikes(compatible_output=True)
```

PyNN 0.8

```
v = p1.get_data('v')
data = p1.get_data(['v', 'gsyn_exc', 'spikes', 'gsyn_inh'])
data = p1.get_data('all')
data = p1.get_data()
data = p1.spinnaker_get_data(variable)
```

- None PyNN method to get PyNN 0.75 format

Write recorded data

PyNN 0.75

```
p1.print_v(file=file_path, compatible_output=True)
p1.print_gsyn(file=file_path, compatible_output=True)
p1.printSpikes(file=file_path, compatible_output=True)
```

PyNN 0.8

```
p1.write_data(io=file_path, variables='v', clear=False, annotations=None)
p1.write_data(io=file_path, variables=['v', 'gsyn_exc', 'spikes', 'gsyn_inh'])
p1.write_data(io=file_path, variables='all')
```

NOTE: File extensions supported by neo by default are : .h5, .pkl, .mat

Neo Blocks and Neo segments

Neo Block

1. Default data type returned by get_data
2. Contains a list of **NeoSegment** that represents the data acquired during all calls to run between a reset.
    a. Currently there is a bug where we don't build another segment after reset. So you only get 1 segment for the current run so far.

Neo Segment

1. Contains:
    a. spiketrains : spike data.
    b. analogsignalarrays : v, gsyn_exc, gsyn_inh data.

Spiketrain representation

Segment.spiketrains returns a List of SpikeTrain

One for each Neuron Id

SpikeTrain

Subclass of One Dimensional Quantity array

Dimension = time

SpikeTrain.annotations['source_index'] -> neuron id

Analogsignal(Array)'s Representation

Segment.analogsignalarrays = List[AnalogSignal]

Segment.filter(name=xyz) = List[AnalogSignal]

AnalogSignal/AnalogSignalArray

Two Dimensional Subclass of Quantity array

Dimension 1 = channel_index / neuron id

Dimension 2 = time

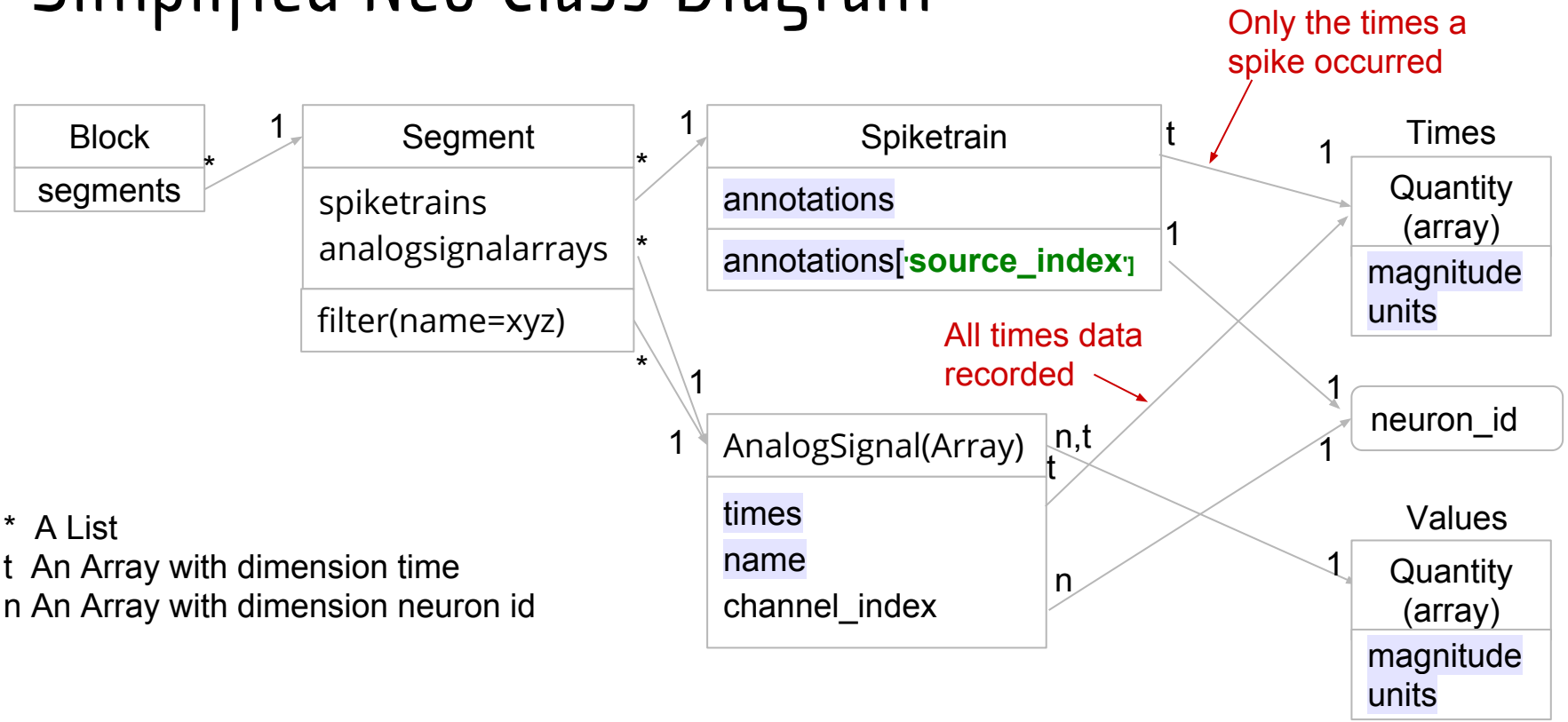
AnalogSignal.times -> Quantity array of time
AnalogSignal.Channel_index -> Array of Neuron ids
AnalogSignal.name -> type of data
AnalogSignal.sampling_rate -> time between two values
AnalogSignal.sampling_period -> total time of run

Quantity (array)

Numpy array of data plus units
Can be iterated or index to give individual Quantity Objects
Quantity.magnitude -> numpy array of data in numerical format (int, float...)
Quantity.units -> UnitQuantity object (V MS ...)
Quantity.rescale(UnitQuantity) -> Quantity (with magnitude adjusted)
All Numpy array methods work

- Quantity.shape
- Quantity.dtype -> numerical format of data
- Quantity.sum() ect

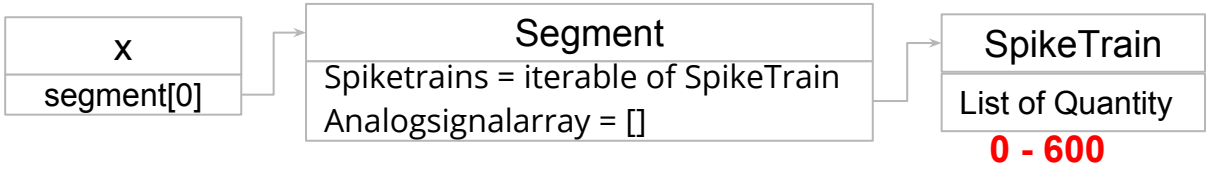
Simplified Neo Class Diagram



* A List
t An Array with dimension time
n An Array with dimension neuron id

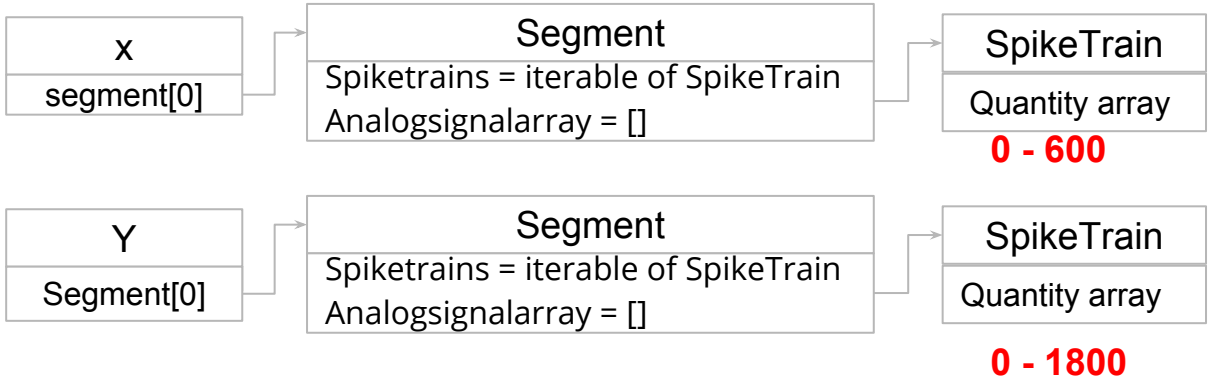
Neo Example 1 of 2

p.setup()
Pop = p.Pop(blah)
pop.record(“spikes”)
p.run(600)
X = pop.get_data(“spikes”)
p.end()



Neo Example 2 of 2

```
p.setup()
Pop = p.Pop(blah)
pop.record("spikes")
p.run(600)
X = pop.get_data("spikes")
p.run(600)
p.run(600)
Y = pop.get_data("spikes")
p.end()
```



PyNN plotting and Neo

PyNN 0.75

1. No Neo support.
2. Christian has placed some plotting tools which can be imported via
   a. **from** spynnaker **import** plot_utils

PyNN 0.8

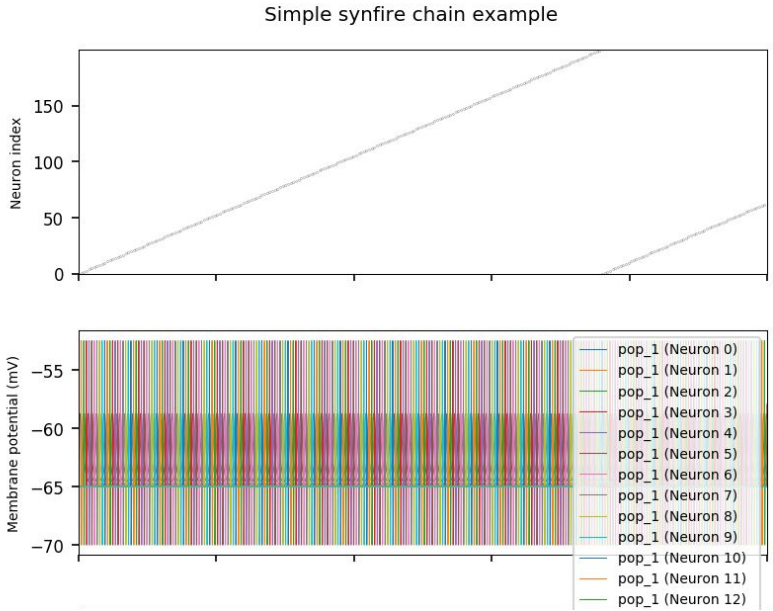
1. Has plotting tools for plotting neo objects.
2. Can be imported via:
   a. **from** pyNN.utility.plotting **import** Figure, Panel
3. Christian has placed some plotting tools which can be imported via
   a. **from** spynnaker8.spynakker_plotting **import** SpynakkerPanel

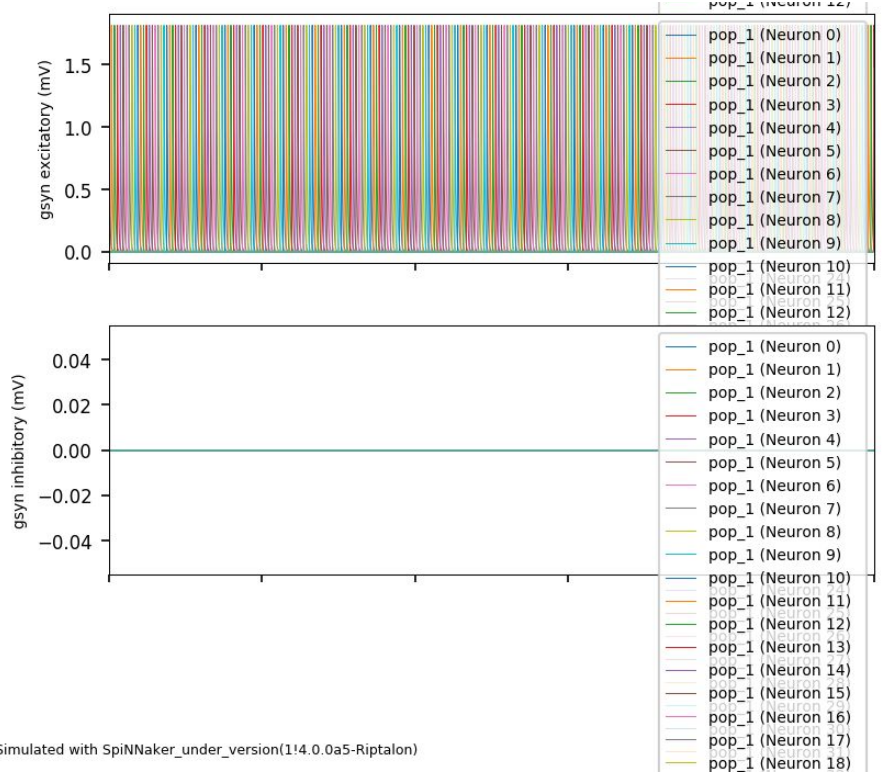
PyNN plotting example

```
import matplotlib.pyplot as plt
import pyNN.utility.plotting as plot
plot.Figure(
    # raster plot of the presynaptic neuron spike times
    plot.Panel(spikes.segments[0].spiketrains, yticks=True, markersize=2, xlim=(0, runtime)),
    # membrane potential of the postsynaptic neuron
    plot.Panel(v.segments[0].filter(name='v')[0], ylabel="Membrane potential (mV)",
               data_labels=[populations[0].label], yticks=True, xlim=(0, runtime)),
    plot.Panel(gsyn_exc.segments[0].filter(name='gsyn_exc')[0], ylabel="gsyn excitatory (mV)",
               data_labels=[populations[0].label], yticks=True, xlim=(0, runtime)),
    plot.Panel(gsyn_inh.segments[0].filter(name='gsyn_inh')[0], ylabel="gsyn inhibitory (mV)",
           data_labels=[populations[0].label], yticks=True, xlim=(0, runtime)),
    title="Simple synfire chain example",
    annotations="Simulated with {}".format(p.name())
)
plt.show()
```

NOTE: is much slower than using matplotlib plottings with numpy array in pynn 0.75.

PyNN plotting example





Projection Static Synapses

PyNN 0.75

```
p.Projection(
        p1, p2, p.FixedProbabilityConnector(p_connect=0.1, weights=0.1, delays=3),
        target='excitatory', synapse_dynamics=None, source=None)
```

PyNN 0.8

```
p.Projection(
        p1, p2, p.FixedProbabilityConnector(p_connect=0.1),
        synapse_type=p.StaticSynapse(weight=0.1, delay=3),
        receptor_type="excitatory", source=None, space=None)
```

Projection RNG setting

PyNN 0.75

```
p.Projection(
        p1, p2, p.FixedProbabilityConnector(p_connect=0.1, weights=0.1, delays=3),
        target='excitatory', synapse_dynamics=None, source=None, rng=p.NativeRNG())
```

PyNN 0.8

```
p.Projection(
        p1, p2, p.FixedProbabilityConnector(p_connect=0.1, rng=p.NativeRNG()),
        synapse_type=p.StaticSynapse(weight=0.1, delay=3),
        receptor_type="excitatory", source=None, space=None)
```

Warning: Be careful with arguments without keywords as connector in pynn0.7 have weights and delay arguements and these values would have other meanings in pynn0.8

AllToAllConnector (weights=0.0, delays=1, allow_self_connections=True, space=Space(), safe=True, verbose=None)

(allow_self_connections=True, safe=True, verbose=None, callbacks=None

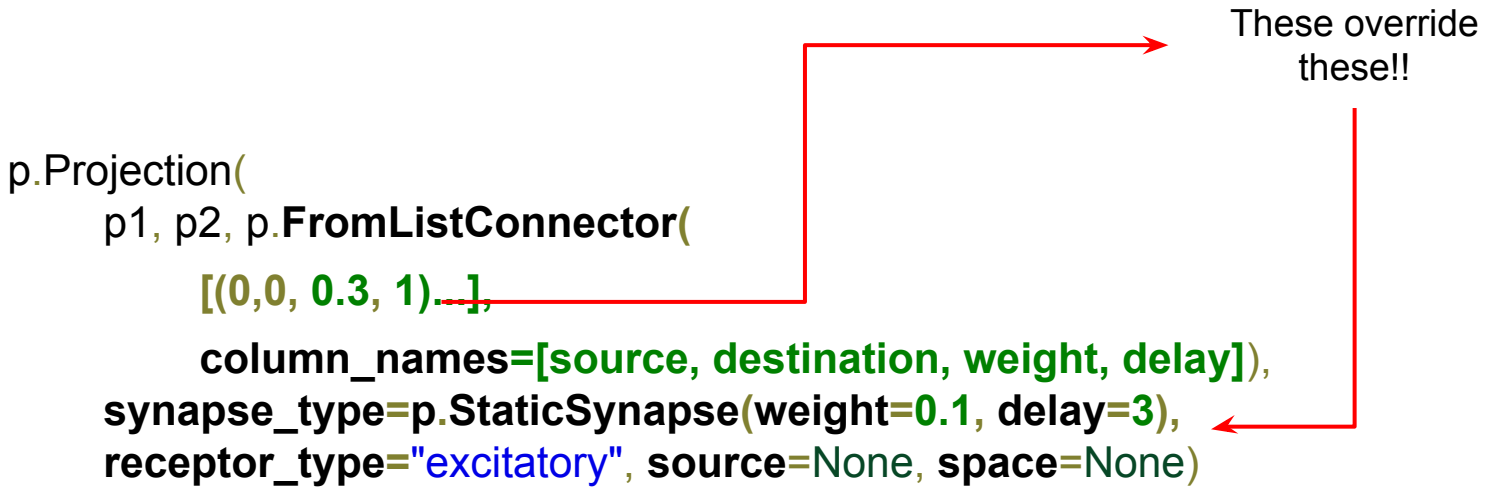
PyNN 0.8 Projection From List Connector Part 1

```
p.Projection(
        p1, p2, p.FromListConnector([(0,0)...]),
        synapse_type=p.StaticSynapse(weight=0.1, delay=3),
        receptor_type="excitatory", source=None, space=None)
```

PyNN 0.8 Projection From List Connector Part 2

```
p.Projection(
        p1, p2, p.FromListConnector([(0,0, <Synapse params>)..], <column_names>),
        synapse_type=p.StaticSynapse(weight=0.1, delay=3),
        receptor_type="excitatory", source=None, space=None)
```

PyNN 0.8 Projection Synapses Oddity



Projection part Plastic Synapses

PyNN 0.75

```
p.Projection(
        p1, p2, ..... , synapse_dynamics=p.SynapseDynamics(
                slow=p.STDPMechanism(
                        timing_dependence=sim.SpikePairRule(tau_plus=16.7, tau_minus=33.7),
                        weight_dependence=sim.AdditiveWeightDependence(
                                w_min=0.0, w_max=1.0, A_plus=0.005, A_minus=0.005))))
```

PyNN 0.8

```
p.Projection(
        p1, p2,...., synapse_type=p.STDPMechanism(
                timing_dependence=sim.SpikePairRule(
                        tau_plus=20., tau_minus=20.0, A_plus=0.005, A_minus=0.005),
                weight_dependence=sim.AdditiveWeightDependence(w_min=0, w_max=1.0),
                weight=0.1, delay=2))
```

Projection get params

PyNN 0.75

```
list = proj.getDelays(format='list', gather=True)
list = proj.getWeights(format='list', gather=True)
```

PyNN 0.8

```
Numpy array = proj.get(self, attribute_names=['weight', 'delay', <plastic params>],
                       format='list', gather=True, with_address=False, multiple_synapses='sum')
```

SpiNNaker Caveats

1. Gather must equal True
2. Format can be ‘list’ (any other value is assumed to be ‘array’)
3. If with_address is True results have source and destination added
   a. Format list is a list (nbarray) of tuples source, destination, value
   b. Format array is a tuple of three(or more) 2 dimensional nbarrays Sources, Destinations, Values
   c. With_address = False is same format as pynn7 (assuming just one attribute_name)
4. multiple_synapses must equal ‘sum’
5. future work might impl the options of ‘last’, ‘first’, ‘sum’, ‘min’, ‘max’ for multiple_synapses

Spike Source Array Sequence

1. Not currently supported (I missed it out accidentally)

```
def generate_spike_times(i_range):
    return [p.Sequence(numpy.add.accumulate(numpy.random.exponential(10.0, size=10))) for i in i_range]


p = sim.Population(30, p.SpikeSourceArray(spike_times=generate_spike_times))
```

PyNN 0.8 functions

pop.annote(annotations)

Add annotations to the pop.

pop.describe()

Uses the annotations within a PyNN defined data structure containing data on the neurons.

pop.annotations()

Returns the user annotations.

SpiNNaker related calls

PyNN 0.75

```
p.set_number_of_neurons_per_core("IF_curr_exp", nNeurons / 2)
spikes = p1.getSpikes(compatible_output=True)
```

PyNN 0.8

```
p.set_number_of_neurons_per_core(p.IF_curr_exp, nNeurons / 2)
numpy_array = p1.spinnaker_get_data('spikes')
```

RandomDistribution

from pyNN.random **import** RandomDistribution

This has significantly changed both in how it is initialised and the methods it supports.

See:

https://github.com/NeuralEnsemble/PyNN/blob/0.8.3/pyNN/random.py

For comparison old code was at:

https://github.com/NeuralEnsemble/PyNN/blob/0.7.5/src/random.py

Warning!!!!!

This may not be the complete changes. This is what we picked up during tests.

Please contact SpiNNaker Users Group [spinnakerusers@googlegroups.com] if you notice any omissions or errors.

But good Luck, and you need to move over before we remove pynn 0.75 support completely.