# CONTENTS

1 User guide 3
   1.1 Introduction ........................................... 3
   1.2 Installation ........................................... 3
   1.3 API documentation ..................................... 5
   1.4 Examples .............................................. 72
   1.5 References ............................................. 87

2 Contributing 89
   2.1 How to contribute ...................................... 89
   2.2 Regenerating documentation ............................ 91
   2.3 Implementation of XML bindings for libNeuroML ........ 91
   2.4 Multicompartmental Python API Meeting ................ 92
   2.5 Nodes, Segments and Sections .......................... 95

3 Indices and tables 99

Bibliography 101

Python Module Index 103

Index 105
Welcome to the libNeuroML documentation. Here you will find information on installing, using, and contributing to libNeuroML. For more information on NeuroML standard, other tools in the NeuroML eco-system, the NeuroML community and how to get in touch with us, please see the documentation at https://docs.neuroml.org.
1.1 Introduction

This package provides Python libNeuroML, for working with neuronal models specified in NeuroML 2.

**Warning:** libNeuroML targets NeuroML v2.0

libNeuroML targets NeuroML v2.0, which is described in Cannon et al, 2014). NeuroML v1.8.1 (Gleeson et al. 2010) is now deprecated and not supported by libNeuroML.

For a detailed description of libNeuroML see Vella *et al.* [VCC+14]. *Please cite the paper if you use libNeuroML.*

1.1.1 NeuroML

NeuroML provides an object model for describing neuronal morphologies, ion channels, synapses and 3D network structure. For more information on NeuroML 2 and LEMS please see the NeuroML documentation.

1.1.2 Serialisations

The XML serialisation will be the “natural” serialisation and will follow closely the NeuroML object model. The format of the XML will be specified by the XML Schema definition (XSD file).

Other serialisations have been developed (HDF5, JSON, SWC). Please see Vella *et al.* [VCC+14] for more details.

1.2 Installation

1.2.1 Using Pip

On most systems with a Python installation, libNeuroML can be installed using the default Python package manager, Pip:

```
pip install libNeuroML
```

It is recommended to use a virtual environment when installing Python packages using *pip* to prevent these from conflicting with other system libraries.

This will support the default XML serialization. To install all of requirements to include the other serialisations, use
# On Ubuntu based systems

```bash
sudo apt-get install libhdf5-dev
pip install libNeuroML[full]
```

The `apt` line is required at time of writing because PyTables’ wheels for python 3.7 depend on the system libhdf5.

## 1.2.2 On Fedora based systems

On Fedora Linux systems, the NeuroFedora community provides libNeuroML in the standard Fedora repos and can be installed using the following commands:

```bash
sudo dnf install python3-libNeuroML
```

## 1.2.3 Install from source

You can clone the GitHub repository and also build libNeuroML from the sources. For this, you will need `git`:

```bash
git clone git://github.com/NeuralEnsemble/libNeuroML.git
cd libNeuroML
```

More details about the git repository and making your own branch/fork are here. To build and install libNeuroML, you can use the standard install method for Python packages (preferably in a virtual environment):

```bash
python setup.py install
```

To use the latest development version of libNeuroML, switch to the development branch:

```bash
git checkout development
sudo python setup.py install
```

## 1.2.4 Run an example

Some sample scripts are included in `neuroml/examples`, e.g.:

```bash
cd neuroml/examples
python build_network.py
```

The standard examples can also be found `Examples`.

## 1.2.5 Unit tests

To run unit tests cd to the directory `neuroml/test` and use the Python unittest module discover method:

```bash
cd neuroml/test/
python -m unittest discover
```

If all tests passed correctly, your output should look something like this:
Ran 55 tests in 40.1s
OK

You can also use PyTest to run tests.

```bash
pip install pytest
pytest -v --strict -W all
```

To ignore some tests, like the MongoDB test which requires a MongoDB setup, run:

```bash
pytest -v -k "not mongodb" --strict -W all
```

# 1.3 API documentation

The libNeuroML API includes the core NeuroML classes and various utilities. You can find information on these in the pages below.

## 1.3.1 nml Module (NeuroML Core classes)

These NeuroML core classes are Python representations of the Component Types defined in the NeuroML standard. These can be used to build NeuroML models in Python, and these models can then be exported to the standard XML NeuroML representation. These core classes also contain some utility functions to make it easier for users to carry out common tasks.

Each NeuroML Component Type is represented here as a Python class. Due to implementation limitations, whereas NeuroML Component Types use lower camel case naming, the Python classes here use upper camel case naming. So, for example, the adExIaFCell Component Type in the NeuroML schema becomes the AdExIaFCell class here, and expTwoSynapse becomes the ExpTwoSynapse class.

The child and children elements that NeuroML Component Types can have are represented in the Python classes as variables. The variable names, to distinguish them from class names, use snake case. So for example, the cell NeuroML Component Type has a corresponding Cell Python class here. The biophysicalProperties child Component Type in cell is represented as the biophysical_properties list variable in the Cell Python class. The class signatures list all the child/children elements and text fields that the corresponding Component Type possesses. To again use the Cell class as an example, the construction signature is this:

```python
class neuroml.nml.nml.Cell(none_lex_id=None, id=None, metaid=None, notes=None,
    →properties=\texttt{None}, annotation=\texttt{None}, morphology_attr=\texttt{None}, biophysical_properties_
    →\texttt{attr}=\texttt{None}, morphology=\texttt{None}, biophysical_properties=\texttt{None}, extensiontype=\texttt{None}, **kwargs_
    →)
```

As can be seen here, it includes both the biophysical_properties and morphology child elements as variables.

Please see the examples in the NeuroML documentation to see usage examples of libNeuroML. Please also note that this module is also included in the top level of the neuroml package, so you can use these classes by importing neuroml:

```python
from neuroml import AdExIaFCell
```
List of Component classes

This documentation is auto-generated from the NeuroML schema. In case of issues, please refer to the schema documentation for clarifications. If the schema documentation does not resolve the issue, please contact us.

AdExIaFCCell

class AdExIaFCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, C=None, g_l=None, EL=None, reset=None, VT=None, thresh=None, del_t=None, tauw=None, refract=None, a=None, b=None, gds_collector_=None, **kwargs_)

Bases: BaseCellMembPotCap


Parameters

- \( g_L \) (conductance) –
- \( EL \) (voltage) –
- \( VT \) (voltage) –
- \( \text{thresh} \) (voltage) –
- \( \text{reset} \) (voltage) –
- \( \text{delT} \) (voltage) –
- \( \text{tauw} \) (time) –
- \( \text{refract} \) (time) –
- \( a \) (conductance) –
- \( b \) (current) –
- \( C \) (capacitance) – Total capacitance of the cell membrane

AlphaCondSynapse

class AlphaCondSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, tau_syn=None, e_rev=None, gds_collector_=None, **kwargs_)

Bases: BasePynnSynapse

AlphaCondSynapse – Alpha synapse: rise time and decay time are both \( \tau_{\text{syn}} \). Conductance based synapse.

Parameters

- \( e_{\text{rev}} \) (none) –
- \( \tau_{\text{syn}} \) (none) –
AlphaCurrSynapse

class neuroml.nml.nml.AlphaCurrSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, tau_syn=None, gds_collector_=None, **kwargs_)

Bases: BasePynnSynapse

AlphaCurrSynapse – Alpha synapse: rise time and decay time are both tau_syn. Current based synapse.

Parameters
    • **tau_syn (none)** –

AlphaCurrentSynapse

class neuroml.nml.nml.AlphaCurrentSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, tau=None, ibase=None, gds_collector_=None, **kwargs_)

Bases: BaseCurrentBasedSynapse

AlphaCurrentSynapse – Alpha current synapse: rise time and decay time are both tau.

Parameters
    • **tau (time)** – Time course for rise and decay
    • **ibase (current)** – Baseline current increase after receiving a spike

AlphaSynapse

class neuroml.nml.nml.AlphaSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, gbase=None, erev=None, tau=None, gds_collector_=None, **kwargs_)

Bases: BaseConductanceBasedSynapse

AlphaSynapse – Ohmic synapse model where rise time and decay time are both tau. Max conductance reached during this time (assuming zero conductance before) is gbase * weight.

Parameters
    • **tau (time)** – Time course of rise/decay
    • **gbase (conductance)** – Baseline conductance, generally the maximum conductance following a single spike
    • **erev (voltage)** – Reversal potential of the synapse
Annotation

class neuroml.nml.Annotation(anytypeobjs_=None, gds_collector_=None, **kwargs_)
   Bases: GeneratedsSuper

Annotation – A structured annotation containing metadata, specifically RDF or property elements

Base

class neuroml.nml.Base(neuro_lex_id=None, id=None, extensiontype_=None, gds_collector_=None, **kwargs_)
   Bases: BaseWithoutId

Base – Anything which can have a unique (within its parent) id of the form NmlId (spaceless combination of letters, numbers and underscore).

BaseCell

class neuroml.nml.BaseCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, extensiontype_=None, gds_collector_=None, **kwargs_)
   Bases: Standalone

BaseCell – Base type of any cell ( e.g. point neuron like izhikevich2007Cell, or a morphologically detailed Cell with segment s ) which can be used in a population

BaseCellMembPotCap

class neuroml.nml.BaseCellMembPotCap(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, C=None, extensiontype_=None, gds_collector_=None, **kwargs_)
   Bases: BaseCell

BaseCellMembPotCap – Any cell with a membrane potential v with voltage units and a membrane capacitance C. Also defines exposed value iSyn for current due to external synapses and iMemb for total transmembrane current (usually channel currents plus iSyn)

Parameters
   C (capacitance) – Total capacitance of the cell membrane

BaseConductanceBasedSynapse

class neuroml.nml.BaseConductanceBasedSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, gbase=None, erev=None, extensiontype_=None, gds_collector_=None, **kwargs_)
   Bases: BaseVoltageDepSynapse

BaseConductanceBasedSynapse – Synapse model which exposes a conductance g in addition to producing a current. Not necessarily ohmic!! cno_0000027

Parameters
• **gbase** *(conductance)* – Baseline conductance, generally the maximum conductance following a single spike
• **erev** *(voltage)* – Reversal potential of the synapse

**BaseConductanceBasedSynapseTwo**

class neuroml.nml.nml.BaseConductanceBasedSynapseTwo(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, gbase1=None, gbase2=None, erev=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: BaseVoltageDepSynapse

BaseConductanceBasedSynapseTwo – Synapse model suited for a sum of two expTwoSynapses which exposes a conductance g in addition to producing a current. Not necessarily ohmic!! cno_0000027

**Parameters**

• **gbase1** *(conductance)* – Baseline conductance 1
• **gbase2** *(conductance)* – Baseline conductance 2
• **erev** *(voltage)* – Reversal potential of the synapse

**BaseConnection**

class neuroml.nml.nml.BaseConnection(neuro_lex_id=None, id=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: BaseNonNegativeIntegerId

BaseConnection – Base of all synaptic connections (chemical/electrical/analog, etc.) inside projections

**BaseConnectionNewFormat**

class neuroml.nml.nml.BaseConnectionNewFormat(neuro_lex_id=None, id=None, pre_cell=None, pre_segment=0.0, pre_fraction_along=0.5, post_cell=None, post_segment=0.0, post_fraction_along=0.5, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: BaseConnection

BaseConnectionNewFormat – Base of all synaptic connections with preCell, postSegment, etc. See BaseConnectionOldFormat

1.3. API documentation
**BaseConnectionOldFormat**

```python
class neuroml.nml.nml.BaseConnectionOldFormat(  
    neuro_lex_id=None, id=None, pre_cell_id=None,  
    pre_segment_id='0', pre_fraction_along='0.5',  
    post_cell_id=None, post_segment_id='0',  
    post_fraction_along='0.5', extensiontype_=None,  
    gds_collector_=None, **kwargs_)
```

**Bases:** BaseConnection

BaseConnectionOldFormat – Base of all synaptic connections with preCellId, postSegmentId, etc. Note: this is not the best name for these attributes, since Id is superfluous, hence BaseConnectionNewFormat

**BaseCurrentBasedSynapse**

```python
class neuroml.nml.nml.BaseCurrentBasedSynapse(  
    neuro_lex_id=None, id=None, metaid=None,  
    notes=None, properties=None, annotation=None,  
    extensiontype_=None, gds_collector_=None,  
    **kwargs_)
```

**Bases:** BaseSynapse

BaseCurrentBasedSynapse – Synapse model which produces a synaptic current.

**BaseNonNegativeIntegerId**

```python
class neuroml.nml.nml.BaseNonNegativeIntegerId(  
    neuro_lex_id=None, id=None, extensiontype_=None,  
    gds_collector_=None, **kwargs_)
```

**Bases:** BaseWithoutId

BaseNonNegativeIntegerId – Anything which can have a unique (within its parent) id, which must be an integer zero or greater.

**BaseProjection**

```python
class neuroml.nml.nml.BaseProjection(  
    neuro_lex_id=None, id=None, presynaptic_population=None,  
    postsynaptic_population=None, extensiontype_=None,  
    gds_collector_=None, **kwargs_)
```

**Bases:** Base

BaseProjection – Base for projection (set of synaptic connections) between two populations

**BasePynnSynapse**

```python
class neuroml.nml.nml.BasePynnSynapse(  
    neuro_lex_id=None, id=None, metaid=None, notes=None,  
    properties=None, annotation=None, tau_syn=None,  
    extensiontype_=None, gds_collector_=None, **kwargs_)
```

**Bases:** BaseSynapse

BasePynnSynapse – Base type for all PyNN synapses. Note, the current \( I \) produced is dimensionless, but it requires a membrane potential \( v \) with dimension voltage
Parameters

tau_syn (none) –

**BaseSynapse**

class neuroml.nml.nml.BaseSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None,
properties=None, annotation=None, extensiontype=None,
gds_collector=None, **kwargs)

Bases: Standalone

BaseSynapse – Base type for all synapses, i.e. ComponentTypes which produce a current (dimension: current) and change Dynamics in response to an incoming event. cno_0000009

**BaseVoltageDepSynapse**

class neuroml.nml.nml.BaseVoltageDepSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None,
properties=None, annotation=None, extensiontype=None,
gds_collector=None, **kwargs)

Bases: BaseSynapse

BaseVoltageDepSynapse – Base type for synapses with a dependence on membrane potential

**BaseWithoutId**

class neuroml.nml.nml.BaseWithoutId(neuro_lex_id=None, extensiontype=None, gds_collector=None,
**kwargs)

Bases: GeneratedsSuper

BaseWithoutId – Base element without ID specified yet, e.g. for an element with a particular requirement on its id which does not comply with NmlId (e.g. Segment needs nonNegativeInteger).

add(obj=None, hint=None, force=False)

Generic function to allow easy addition of a new member to a NeuroML object.

Without arguments, when obj=None, it simply calls the info() method to provide the list of valid member types for the NeuroML class.

Use info(show_contents=True) to see the valid members of this class, and their current contents.

Parameters

• obj (any NeuroML Type defined by the API) – object member to add
• hint (string) – member name to add to when there are multiple members that obj can be added to
• force (bool) – boolean to force addition when an obj has already been added previously

Raises

• Exception – if a member compatible to obj could not be found
• Exception – if multiple members can accept the object and no hint is provided.
get_members()

Get member data items, also from ancestors.

This function is required because generateDS does not include inherited members in the member_data_items list for a derived class. So, for example, while IonChannelHH has gate_hh_rates which it inherits from IonChannel, IonChannelHH’s member_data_items_ is empty. It relies on the IonChannel classes’ member_data_items_ list.

Returns
list of members, including ones inherited from ancestors.

info(show_contents=False)

A helper function to get a list of members of this class.

This is useful to quickly check what members can go into a particular NeuroML class (which will match the Schema definitions). It lists these members and notes whether they are “single” type elements (Child elements) or “List” elements (Children elements). It will also note whether a member is optional or required.

See http://www.davekuhlman.org/generateDS.html#user-methods for more information on the MemberSpec class that generateDS uses.

Parameters
  show_contents (bool) – also prints out the contents of the members

Returns
  the string (for testing purposes)

BiophysicalProperties

class neuroml.nml.nml.BiophysicalProperties(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, membrane_properties=None, intracellular_properties=None, extracellular_properties=None, gds_collector_=None, **kwargs_)

Bases: Standalone

BiophysicalProperties – The biophysical properties of the cell, including the membraneProperties and the intracellularProperties

BiophysicalProperties2CaPools

class neuroml.nml.nml.BiophysicalProperties2CaPools(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, membrane_properties2_ca_pools=None, intracellular_properties2_ca_pools=None, extracellular_properties=None, gds_collector_=None, **kwargs_)

Bases: Standalone

BiophysicalProperties2CaPools – The biophysical properties of the cell, including the membraneProperties2CaPools and the intracellularProperties2CaPools for a cell with two Ca pools
BlockMechanism

class neuroml.nml.nml.BlockMechanism(type=None, species=None, block_concentration=None, scaling_conc=None, scaling_volt=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

BlockingPlasticSynapse

class neuroml.nml.nml.BlockingPlasticSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, gbase=None, erev=None, tau_decay=None, tau_rise=None, plasticity_mechanism=None, block_mechanism=None, gds_collector_=None, **kwargs_)

Bases: ExpTwoSynapse

BlockingPlasticSynapse – Biexponential synapse that allows for optional block and plasticity mechanisms, which can be expressed as child elements.

Parameters

- tauRise (time) –
- tauDecay (time) –
- gbase (conductance) – Baseline conductance, generally the maximum conductance following a single spike
- erev (voltage) – Reversal potential of the synapse

Case

class neuroml.nml.nml.Case(condition=None, value=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Cell

class neuroml.nml.nml.Cell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, morphology_attr=None, biophysical_properties_attr=None, morphology=None, biophysical_properties=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: BaseCell

Cell – Cell with segment s specified in a morphology element along with details on its biophysicalProperties. NOTE: this can only be correctly simulated using jLEMS when there is a single segment in the cell, and v of this cell represents the membrane potential in that isopotential segment.

get_actual_proximal(segment_id)

Get the proximal point of a segment.

Get the proximal point of a segment, even the proximal field is None and so the proximal point is on the parent (at a point set by fraction_along).
Parameters
segment_id – ID of segment

Returns
proximal point

get_all_segments_in_group(segment_group, assume_all_means_all=True)
Get all the segments in a segment group of the cell.

Parameters
• segment_group – segment group to get all segments of
• assume_all_means_all – return all segments if the segment group wasn’t explicitly de-
   fined

Todo
check docstring

Returns
list of segments

 Raises
Exception – if no segment group is found in the cell.

get_ordered_segments_in_groups(group_list, check_parentage=False,
include_cumulative_lengths=False, include_path_lengths=False,
path_length_metric='Path Length from root')
Get ordered list of segments in specified groups

Parameters
• group_list – list of groups to get segments from
• check_parentage – verify parentage
• include_cumulative_lengths – also include cumulative lengths
• include_path_lengths – also include path lengths
• path_length_metric –

Returns
dictionary of segments with additional information depending on what parameters were used:

 Raises
Exception if check_parentage is True and parentage cannot be verified

get_segment(segment_id)
Get segment object by its id

Parameters
segment_id – ID of segment

Returns
segment

 Raises
Exception – if the segment is not found in the cell

get_segment_group(sg_id)
Return the SegmentGroup object for the specified segment group id.
Parameters

\texttt{sg\_id (str)} – id of segment group to find

Returns

SegmentGroup object of specified ID

Raises

\texttt{Exception} – if segment group is not found in cell

\texttt{get\_segment\_groups\_by\_substring}(\texttt{substring})

Get a dictionary of segment group IDs and the segment groups matching the specified substring

Parameters

\texttt{substring (str)} – substring to match

Returns

dictionary with segment group ID as key, and segment group as value

Raises

\texttt{Exception} – if no segment groups are not found in cell

\texttt{get\_segment\_ids\_vs\_segments}()

Get a dictionary of segment IDs and the segments in the cell.

Returns

dictionary with segment ID as key, and segment as value

\texttt{get\_segment\_length}(\texttt{segment\_id})

Get the length of the segment.

Parameters

\texttt{segment\_id} – ID of segment

Returns

length of segment

\texttt{get\_segment\_surface\_area}(\texttt{segment\_id})

Get the surface area of the segment.

Parameters

\texttt{segment\_id} – ID of the segment

Returns

surface area of segment

\texttt{get\_segment\_volume}(\texttt{segment\_id})

Get volume of segment

Parameters

\texttt{segment\_id} – ID of the segment

Returns

volume of the segment

\texttt{get\_segments\_by\_substring}(\texttt{substring})

Get a dictionary of segment IDs and the segment matching the specified substring

Parameters

\texttt{substring (str)} – substring to match

Returns

dictionary with segment ID as key, and segment as value
Raises

**Exception** – if no segments are found

```python
summary()
```

Print cell summary.

**Cell2CaPools**

```python
class neuroml.nml.nml.Cell2CaPools(neuro_lex_id=None, id=None, metaid=None, notes=None,
                                      properties=None, annotation=None, morphology_attr=None,
                                      biophysical_properties_attr=None, morphology=None,
                                      biophysical_properties=None,
                                      biophysical_properties2_ca_pools=None, gds_collector_=None,
                                      **kwargs_)
```

Bases: *Cell*

Cell2CaPools – Variant of cell with two independent Ca2+ pools. Cell with segment s specified in a morphology element along with details on its biophysicalProperties. NOTE: this can only be correctly simulated using jLEMS when there is a single segment in the cell, and \( V \) of this cell represents the membrane potential in that isopotential segment.

**CellSet**

```python
class neuroml.nml.nml.CellSet(neuro_lex_id=None, id=None, select=None, anytypeobjs_=None,
                               gds_collector_=None, **kwargs_)
```

Bases: *Base*

**ChannelDensity**

```python
class neuroml.nml.nml.ChannelDensity(neuro_lex_id=None, id=None, ion_channel=None,
                                       cond_density=None, erev=None, segment_groups='all',
                                       segments=None, ion=None, variable_parameters=None,
                                       extensiontype_=None, gds_collector_=None, **kwargs_)
```

Bases: *Base*

ChannelDensity – Specifies a time varying ohmic conductance density, \( gDensity \), which is distributed on an area of the cell (specified in membraneProperties) with fixed reversal potential \( erev \) producing a current density \( iDensity \)

**Parameters**

- \( erev \) (voltage) – The reversal potential of the current produced
- \( condDensity \) (conductanceDensity) –
**ChannelDensityGHK**

**class** `ChannelDensityGHK`(`neuro_lex_id=None, id=None, ion_channel=None, permeability=None, segment_groups='all', segments=None, ion=None, gds_collector_=None, **kwargs_)`

**Bases:** `Base`

ChannelDensityGHK – Specifies a time varying conductance density, gDensity, which is distributed on an area of the cell, producing a current density iDensity and whose reversal potential is calculated from the Goldman Hodgkin Katz equation. Hard coded for Ca only! See [https://github.com/OpenSourceBrain/ghk-nernst](https://github.com/OpenSourceBrain/ghk-nernst).

**Parameters**
- `permeability (permeability)` –

**ChannelDensityGHK2**

**class** `ChannelDensityGHK2`(`neuro_lex_id=None, id=None, ion_channel=None, cond_density=None, segment_groups='all', segments=None, ion=None, gds_collector_=None, **kwargs_)`

**Bases:** `Base`

ChannelDensityGHK2 – Time varying conductance density, gDensity, which is distributed on an area of the cell, producing a current density iDensity. Modified version of Jaffe et al. 1994 (used also in Lawrence et al. 2006). See [https://github.com/OpenSourceBrain/ghk-nernst](https://github.com/OpenSourceBrain/ghk-nernst).

**Parameters**
- `condDensity (conductanceDensity)` –

**ChannelDensityNernst**

**class** `ChannelDensityNernst`(`neuro_lex_id=None, id=None, ion_channel=None, cond_density=None, segment_groups='all', segments=None, ion=None, variable_parameters=None, extensiontype_=None, gds_collector_=None, **kwargs_)`

**Bases:** `Base`

ChannelDensityNernst – Specifies a time varying conductance density, gDensity, which is distributed on an area of the cell, producing a current density iDensity and whose reversal potential is calculated from the Nernst equation. Hard coded for Ca only! See [https://github.com/OpenSourceBrain/ghk-nernst](https://github.com/OpenSourceBrain/ghk-nernst).

**Parameters**
- `condDensity (conductanceDensity)` –

**ChannelDensityNernstCa2**

**class** `ChannelDensityNernstCa2`(`neuro_lex_id=None, id=None, ion_channel=None, cond_density=None, segment_groups='all', segments=None, ion=None, variable_parameters=None, gds_collector_=None, **kwargs_)`

**Bases:** `ChannelDensityNernst`
ChannelDensityNernstCa2 – This component is similar to the original component type `channelDensityNernst` but it is changed in order to have a reversal potential that depends on a second independent Ca++ pool (ca2).

See https://github.com/OpenSourceBrain/ghk-nernst.

Parameters

condDensity (conductanceDensity) –

ChannelDensityNonUniform

class `neuroml.nml.nml.ChannelDensityNonUniform`(neuro_lex_id=None, id=None, ion_channel=None, erev=None, ion=None, variable_parameters=None, gds_collector_=None, **kwargs_)

Bases: `Base`

ChannelDensityNonUniform – Specifies a time varying ohmic conductance density, which is distributed on a region of the cell. The conductance density of the channel is not uniform, but is set using the `variableParameter`. Note, there is no dynamical description of this in LEMS yet, as this type only makes sense for multicompartmental cells. A ComponentType for this needs to be present to enable export of NeuroML 2 multicompartmental cells via LEMS/jNeuroML to NEURON.

Parameters

erev (voltage) – The reversal potential of the current produced

ChannelDensityNonUniformGHK

class `neuroml.nml.nml.ChannelDensityNonUniformGHK`(neuro_lex_id=None, id=None, ion_channel=None, ion=None, variable_parameters=None, gds_collector_=None, **kwargs_)

Bases: `Base`

ChannelDensityNonUniformGHK – Specifies a time varying conductance density, which is distributed on a region of the cell, and whose current is calculated from the Goldman-Hodgkin-Katz equation. Hard coded for Ca only!. The conductance density of the channel is not uniform, but is set using the `variableParameter`. Note, there is no dynamical description of this in LEMS yet, as this type only makes sense for multicompartmental cells. A ComponentType for this needs to be present to enable export of NeuroML 2 multicompartmental cells via LEMS/jNeuroML to NEURON.

ChannelDensityNonUniformNernst

class `neuroml.nml.nml.ChannelDensityNonUniformNernst`(neuro_lex_id=None, id=None, ion_channel=None, ion=None, variable_parameters=None, gds_collector_=None, **kwargs_)

Bases: `Base`

ChannelDensityNonUniformNernst – Specifies a time varying conductance density, which is distributed on a region of the cell, and whose reversal potential is calculated from the Nernst equation. Hard coded for Ca only!. The conductance density of the channel is not uniform, but is set using the `variableParameter`. Note, there is no dynamical description of this in LEMS yet, as this type only makes sense for multicompartmental cells. A ComponentType for this needs to be present to enable export of NeuroML 2 multicompartmental cells via LEMS/jNeuroML to NEURON.
ChannelDensityVShift

```python
class neuroml.nml.nml.ChannelDensityVShift(
    neuro_lex_id=None, id=None, ion_channel=None,
    cond_density=None, erev=None, segment_groups='all',
    segments=None, ion=None, variable_parameters=None,
    v_shift=None, gds_collector_=None, **kwargs_)
```

Bases: `ChannelDensity`

ChannelDensityVShift – Same as `channelDensity`, but with a `vShift` parameter to change voltage activation of gates. The exact usage of `vShift` in expressions for rates is determined by the individual gates.

**Parameters**

- `vShift (voltage)` –
- `erev (voltage)` – The reversal potential of the current produced
- `condDensity (conductanceDensity)` –

ChannelPopulation

```python
class neuroml.nml.nml.ChannelPopulation(
    neuro_lex_id=None, id=None, ion_channel=None,
    number=None, erev=None, segment_groups='all',
    segments=None, ion=None, variable_parameters=None,
    gds_collector_=None, **kwargs_)
```

Bases: `Base`

ChannelPopulation – Population of a `number` of ohmic ion channels. These each produce a conductance `channelgac` across a reversal potential `erev`, giving a total current `i`. Note that active membrane currents are more frequently specified as a density over an area of the cell using `channelDensity`

**Parameters**

- `number (none)` – The number of channels present. This will be multiplied by the time varying conductance of the individual ion channel (which extends `basIonChannel`) to produce the total conductance
- `erev (voltage)` – The reversal potential of the current produced

ClosedState

```python
class neuroml.nml.nml.ClosedState(
    neuro_lex_id=None, id=None, gds_collector_=None, **kwargs_)
```

Bases: `Base`

ClosedState – A `KState` with `relativeConductance` of 0

**Parameters**

- `relativeConductance (none)` –
ComponentType

```python
class neuroml.nml.nml.ComponentType(name=None, extends=None, description=None, Property=None, Parameter=None, Constant=None, Exposure=None, Requirement=None, InstanceRequirement=None, Dynamics=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

ComponentType – Contains an extension to NeuroML by creating custom LEMS ComponentType.
```

CompoundInput

```python
class neuroml.nml.nml.CompoundInput(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, pulseGenerators=None, sineGenerators=None, rampGenerators=None, gds_collector_=None, **kwargs_)

Bases: Standalone

CompoundInput – Generates a current which is the sum of all its child basePointCurrent element, e. g. can be a combination of pulseGenerator , sineGenerator elements producing a single i. Scaled by weight, if set
```

CompoundInputDL

```python
class neuroml.nml.nml.CompoundInputDL(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, pulse_generator_dls=None, sine_generator_dls=None, ramp_generator_dls=None, gds_collector_=None, **kwargs_)

Bases: Standalone

CompoundInputDL – Generates a current which is the sum of all its child basePointCurrentDL elements, e. g. can be a combination of pulseGeneratorDL , sineGeneratorDL elements producing a single i. Scaled by weight, if set
```

ConcentrationModel_D

```python
class neuroml.nml.nml.ConcentrationModel_D(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, ion=None, resting_conc=None, decay_constant=None, shell_thickness=None, type='decayingPoolConcentrationModel', gds_collector_=None, **kwargs_)

Bases: DecayingPoolConcentrationModel
```
ConditionalDerivedVariable

class neuroml.nml.nml.ConditionalDerivedVariable(name=None, dimension=None, description=None, exposure=None, Case=None, gds_collector_=None, **kwargs_)

Bases: NamedDimensionalVariable

ConditionalDerivedVariable – LEMS ComponentType for ConditionalDerivedVariable

Connection

class neuroml.nml.nml.Connection(neuro_lex_id=None, id=None, pre_cell_id=None, pre_segment_id='0', pre_fraction_along='0.5', post_cell_id=None, post_segment_id='0', post_fraction_along='0.5', gds_collector_=None, **kwargs_)

Bases: BaseConnectionOldFormat

Connection – Event connection directly between named components, which gets processed via a new instance of a synapse component which is created on the target component. Normally contained inside a projection element.

get_post_cell_id()
    Get the ID of the post-synaptic cell

    Returns
    ID of post-synaptic cell

    Return type
    str

get_post_fraction_along()
    Get post-synaptic fraction along information

get_post_info()
    Get post-synaptic information summary

get_post_segment_id()
    Get the ID of the post-synaptic segment

    Returns
    ID of post-synaptic segment.

    Return type
    str

get_pre_cell_id()
    Get the ID of the pre-synaptic cell

    Returns
    ID of pre-synaptic cell

    Return type
    str

get_pre_fraction_along()
    Get pre-synaptic fraction along information
get_pre_info()

Get pre-synaptic information summary

get_pre_segment_id()

Get the ID of the pre-synaptic segment

Returns
ID of pre-synaptic segment.

Return type
str

ConnectionWD

class neuroml.nml.nml.ConnectionWD(neuro_lex_id=None, id=None, pre_cell_id=None, pre_segment_id='0',
pre_fraction_along='0.5', post_cell_id=None, post_segment_id='0',
post_fraction_along='0.5', weight=None, delay=None,
gds_collector_=None, **kwargs_)

Bases: BaseConnectionOldFormat

ConnectionWD – Event connection between named components, which gets processed via a new instance of a synapse component which is created on the target component, includes setting of weight and delay for the synaptic connection

Parameters

• weight (none) –
• delay (time) –

get_delay_in_ms()

Get connection delay in milli seconds

Returns
connection delay in milli seconds

Return type
float

get_post_cell_id()

Get the ID of the post-synaptic cell

Returns
ID of post-synaptic cell

Return type
str

get_post_fraction_along()

Get post-synaptic fraction along information

get_post_info()

Get post-synaptic information summary

get_post_segment_id()

Get the ID of the post-synaptic segment

Returns
ID of post-synaptic segment.
Return type
    str

get_pre_cell_id()
    Get the ID of the pre-synaptic cell

Returns
    ID of pre-synaptic cell

Return type
    str

get_pre_fraction_along()
    Get pre-synaptic fraction along information

get_pre_info()
    Get pre-synaptic information summary

get_pre_segment_id()
    Get the ID of the pre-synaptic segment

Returns
    ID of pre-synaptic segment.

Return type
    str

Constant

class neuroml.nml.nml.Constant(name=None, dimension=None, value=None, description=None, gds_collector_=None, **kwargs_)
    Bases: GeneratedsSuper

    Constant – LEMS ComponentType for Constant.

ContinuousConnection

class neuroml.nml.nml.ContinuousConnection(neuro_lex_id=None, id=None, pre_cell=None, pre_segment='0', pre_fraction_along='0.5', post_cell=None, post_segment='0', post_fraction_along='0.5', pre_component=None, post_component=None, extensiontype_=None, gds_collector_=None, **kwargs_)
    Bases: BaseConnectionNewFormat

    ContinuousConnection – An instance of a connection in a continuousProjection between presynapticPopulation to another postsynapticPopulation through a preComponent at the start and postComponent at the end. Can be used for analog synapses.

get_post_cell_id()
    Get the ID of the post-synaptic cell

Returns
    ID of post-synaptic cell

Return type
    str
get_post_fraction_along()  
Get post-synaptic fraction along information

get_post_info()  
Get post-synaptic information summary

get_post_segment_id()  
Get the ID of the post-synaptic segment

    Returns  
    ID of post-synaptic segment.

    Return type  
    str

get_pre_cell_id()  
Get the ID of the pre-synaptic cell

    Returns  
    ID of pre-synaptic cell

    Return type  
    str

get_pre_fraction_along()  
Get pre-synaptic fraction along information

get_pre_info()  
Get pre-synaptic information summary

get_pre_segment_id()  
Get the ID of the pre-synaptic segment

    Returns  
    ID of pre-synaptic segment.

    Return type  
    str

ContinuousConnectionInstance

class neuroml.nml.nml.ContinuousConnectionInstance(neuro_lex_id=None, id=None, pre_cell=None,  
pre_segment='0', pre_fraction_along='0.5',  
post_cell=None, post_segment='0',  
post_fraction_along='0.5', pre_component=None,  
post_component=None, extensiontype_=None,  
gds_collector_=None, **kwargs_)

Bases: ContinuousConnection

ContinuousConnectionInstance – An instance of a connection in a continuousProjection between presynapticPopulation to another postsynapticPopulation through a preComponent at the start and postComponent at the end. Populations need to be of type populationList and contain instance and location elements. Can be used for analog synapses.
ContinuousConnectionInstanceW

class neuroml.nml.nml.ContinuousConnectionInstanceW(neuro_lex_id=None, id=None, pre_cell=None, pre_segment='0', pre_fraction_along=0.5, post_cell=None, post_segment='0', post_fraction_along=0.5, pre_component=None, post_component=None, weight=None, gds_collector_=None, **kwargs_)

Bases: ContinuousConnectionInstance

ContinuousConnectionInstanceW – An instance of a connection in a continuousProjection between presynapticPopulation to another postsynapticPopulation through a preComponent at the start and postComponent at the end. Populations need to be of type populationList and contain instance and location elements. Can be used for analog synapses. Includes setting of weight for the connection

Parameters
    weight (none) –

get_weight()
    Get weight.
    If weight is not set, the default value of 1.0 is returned.

ContinuousProjection

class neuroml.nml.nml.ContinuousProjection(neuro_lex_id=None, id=None, presynaptic_population=None, postsynaptic_population=None, continuous_connections=None, continuous_connection_instances=None, continuous_connection_instance_ws=None, gds_collector_=None, **kwargs_)

Bases: BaseProjection

ContinuousProjection – A projection between presynapticPopulation and postsynapticPopulation through components preComponent at the start and postComponent at the end of a continuousConnection or continuousConnectionInstance. Can be used for analog synapses.

exportHdf5(h5file, h5Group)
    Export to HDF5 file.

DecayingPoolConcentrationModel

class neuroml.nml.nml.DecayingPoolConcentrationModel(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, ion=None, resting_conc=None, decay_constant=None, shell_thickness=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: Standalone

DecayingPoolConcentrationModel – Model of an intracellular buffering mechanism for ion (currently hard coded to be calcium, due to requirement for iCa) which has a baseline level restingConc and tends to this
value with time course decayConstant. The ion is assumed to occupy a shell inside the membrane of thickness shellThickness.

Parameters

- restingConc (concentration) –
- decayConstant (time) –
- shellThickness (length) –

DerivedVariable

class neuroml.nml.nml.DerivedVariable(name=None, dimension=None, description=None, exposure=None, value=None, select=None, gds_collector_=None, **kwargs_)

Bases: NamedDimensionalVariable

DerivedVariable – LEMS ComponentType for DerivedVariable

DistalDetails

class neuroml.nml.nml.DistalDetails(normalization_end=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

DoubleSynapse

class neuroml.nml.nml.DoubleSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, synapse1=None, synapse2=None, synapse1_path=None, synapse2_path=None, gds_collector_=None, **kwargs_)

Bases: BaseVoltageDepSynapse

DoubleSynapse – Synapse consisting of two independent synaptic mechanisms (e.g. AMPA-R and NMDA-R), which can be easily colocalized in connections

Dynamics

class neuroml.nml.nml.Dynamics(StateVariable=None, DerivedVariable=None, ConditionalDerivedVariable=None, TimeDerivative=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Dynamics – LEMS ComponentType for Dynamics
**EIF_cond_alpha_isfa_ista**

```python
class neuroml.nml.nml.EIF_cond_alpha_isfa_ista(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, tau_refrac=None, v_reset=None, v_rest=None, v_thresh=None, e_rev_E=None, e_rev_I=None, a=None, b=None, delta_T=None, tau_w=None, v_spike=None, gds_collector_=None, **kwargs_)
```

**Bases:** *EIF_cond_exp_isfa_ista*

`EIF_cond_alpha_isfa_ista` – Adaptive exponential integrate and fire neuron according to Brette R and Gerstner W (2005) with alpha-function-shaped post-synaptic conductance

**Parameters**

- **v_spike** *(none)* –
- **delta_T** *(none)* –
- **tau_w** *(none)* –
- **a** *(none)* –
- **b** *(none)* –
- **e_rev_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **e_rev_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **tau_refrac** *(none)* –
- **v_thresh** *(none)* –
- **tau_m** *(none)* –
- **v_rest** *(none)* –
- **v_reset** *(none)* –
- **cm** *(none)* –
- **i_offset** *(none)* –
- **tau_syn_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **tau_syn_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **v_init** *(none)* –
**EIF_cond_exp_isfa_ista**

```python
class neuroml.nml.nml.EIF_cond_exp_isfa_ista(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, tau_m=None, tau_refrac=None, v_reset=None, v_rest=None, v_thresh=None, e_rev_E=None, e_rev_I=None, a=None, b=None, delta_T=None, tau_w=None, v_spike=None, extensiontype_=None, gds_collector_=None, **kwargs_)
```

Bases: `basePyNNiAFCondCell`

**EIF_cond_exp_isfa_ista** – Adaptive exponential integrate and fire neuron according to Brette R and Gerstner W (2005) with exponentially-decaying post-synaptic conductance

**Parameters**

- **v_spike** *(none)* –
- **delta_T** *(none)* –
- **tau_w** *(none)* –
- **a** *(none)* –
- **b** *(none)* –
- **e_rev_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **e_rev_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **tau_refrac** *(none)* –
- **v_thresh** *(none)* –
- **tau_m** *(none)* –
- **v_rest** *(none)* –
- **v_reset** *(none)* –
- **cm** *(none)* –
- **i_offset** *(none)* –
- **tau_syn_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **tau_syn_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **v_init** *(none)* –
ElectricalConnection

class neuroml.nml.nml.ElectricalConnection(neuro_lex_id=None, id=None, pre_cell=None, 
pre_segment='0', pre_fraction_along='0.5', post_cell=None, 
post_segment='0', post_fraction_along='0.5', 
synapse=None, extensiontype_=None, 
gds_collector_=None, **kwargs_)

Bases: BaseConnectionNewFormat

ElectricalConnection – To enable connections between populations through gap junctions.

get_post_cell_id()
    Get the ID of the post-synaptic cell

    Returns
    ID of post-synaptic cell

    Return type
    str

get_post_fraction_along()
    Get post-synaptic fraction along information

get_post_info()
    Get post-synaptic information summary

get_post_segment_id()
    Get the ID of the post-synaptic segment

    Returns
    ID of post-synaptic segment.

    Return type
    str

get_pre_cell_id()
    Get the ID of the pre-synaptic cell

    Returns
    ID of pre-synaptic cell

    Return type
    str

get_pre_fraction_along()
    Get pre-synaptic fraction along information

get_pre_info()
    Get pre-synaptic information summary

get_pre_segment_id()
    Get the ID of the pre-synaptic segment

    Returns
    ID of pre-synaptic segment.

    Return type
    str
**ElectricalConnectionInstance**

```python
class neuroml.nml.nml.ElectricalConnectionInstance(neuro_lex_id=None, id=None, pre_cell=None, pre_segment='0', pre_fraction_along='0.5', post_cell=None, post_segment='0', post_fraction_along='0.5', synapse=None, extensiontype_=None, gds_collector_=None, **kwargs_)
```

**Bases:** ElectricalConnection

ElectricalConnectionInstance – To enable connections between populations through gap junctions. Populations need to be of type `populationList` and contain `instance` and `location` elements.

**ElectricalConnectionInstanceW**

```python
class neuroml.nml.nml.ElectricalConnectionInstanceW(neuro_lex_id=None, id=None, pre_cell=None, pre_segment='0', pre_fraction_along='0.5', post_cell=None, post_segment='0', post_fraction_along='0.5', synapse=None, weight=None, gds_collector_=None, **kwargs_)
```

**Bases:** ElectricalConnectionInstance

ElectricalConnectionInstanceW – To enable connections between populations through gap junctions. Populations need to be of type `populationList` and contain `instance` and `location` elements. Includes setting of `weight` for the connection.

**Parameters**

- **weight** *(none)*

**get_weight()**

Get the weight of the connection

If a weight is not set (or is set to None), returns the default value of 1.0.

**Returns**

- weight of connection or 1.0 if not set

**Return type**

- float

**ElectricalProjection**

```python
class neuroml.nml.nml.ElectricalProjection(neuro_lex_id=None, id=None, presynaptic_population=None, postsynaptic_population=None, electrical_connections=None, electrical_connection_instances=None, electrical_connection_instance_ws=None, gds_collector_=None, **kwargs_)
```

**Bases:** BaseProjection

ElectricalProjection – A projection between `presynapticPopulation` to another `postsynapticPopulation` through gap junctions.
exportHdf5(h5file, h5Group)
    Export to HDF5 file.

ExpCondSynapse
class neuroml.nml.nml.ExpCondSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None,
    properties=None, annotation=None, tau_syn=None, e_rev=None,
    gds_collector_=None, **kwargs_)
    Bases: BasePynnSynapse
    ExpCondSynapse – Conductance based synapse with instantaneous rise and single exponential decay (with time constant tau_syn)
    Parameters
    • e_rev (none) –
    • tau_syn (none) –

ExpCurrSynapse
class neuroml.nml.nml.ExpCurrSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None,
    properties=None, annotation=None, tau_syn=None,
    gds_collector_=None, **kwargs_)
    Bases: BasePynnSynapse
    ExpCurrSynapse – Current based synapse with instantaneous rise and single exponential decay (with time constant tau_syn)
    Parameters
    • tau_syn (none) –

ExpOneSynapse
class neuroml.nml.nml.ExpOneSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None,
    properties=None, annotation=None, gbase=None, erev=None,
    tau_decay=None, gds_collector_=None, **kwargs_)
    Bases: BaseConductanceBasedSynapse
    ExpOneSynapse – Ohmic synapse model whose conductance rises instantaneously by (gbase * weight) on receiving an event, and which decays exponentially to zero with time course tauDecay
    Parameters
    • tauDecay (time) – Time course of decay
    • gbase (conductance) – Baseline conductance, generally the maximum conductance following a single spike
    • erev (voltage) – Reversal potential of the synapse
ExpThreeSynapse

class neuroml.nml.nml.ExpThreeSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, gbase1=None, gbase2=None, erev=None, tau_decay1=None, tau_decay2=None, tau_rise=None, gds_collector_=None, **kwargs_)

Bases: BaseConductanceBasedSynapseTwo

ExpThreeSynapse – Ohmic synapse similar to expTwoSynapse but consisting of two components that can differ in decay times and max conductances but share the same rise time.

Parameters

- **tauRise** (time) –
- **tauDecay1** (time) –
- **tauDecay2** (time) –
- **gbase1** (conductance) – Baseline conductance 1
- **gbase2** (conductance) – Baseline conductance 2
- **erev** (voltage) – Reversal potential of the synapse

ExpTwoSynapse

class neuroml.nml.nml.ExpTwoSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, gbase=None, erev=None, tau_decay=None, tau_rise=None, gds_collector_=None, **kwargs_)

Bases: BaseConductanceBasedSynapse

ExpTwoSynapse – Ohmic synapse model whose conductance waveform on receiving an event has a rise time of **tauRise** and a decay time of **tauDecay**. Max conductance reached during this time (assuming zero conductance before) is **gbase** * weight.

Parameters

- **tauRise** (time) –
- **tauDecay** (time) –
- **gbase** (conductance) – Baseline conductance, generally the maximum conductance following a single spike
- **erev** (voltage) – Reversal potential of the synapse

ExplicitInput

class neuroml.nml.nml.ExplicitInput(target=None, input=None, destination=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

ExplicitInput – An explicit input (anything which extends **basePointCurrent**) to a target cell in a population
```
get_fraction_along()
Get fraction along.
Returns 0.5 if fraction_along was not set.

get_segment_id()
Get the ID of the segment.
Returns 0 if segment_id was not set.

get_target_cell_id()
Get target cell ID

get_target_population()
Get target population.
```

**Exposure**

class neuroml.nml.nml.Exposure(name=None, dimension=None, description=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Exposure – LEMS Exposure (ComponentType property)

**ExtracellularProperties**

class neuroml.nml.nml.ExtracellularProperties(neuro_lex_id=None, id=None, species=None, gds_collector_=None, **kwargs_)

Bases: Base

**ExtracellularPropertiesLocal**

class neuroml.nml.nml.ExtracellularPropertiesLocal(species=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

**FitzHughNagumo1969Cell**

class neuroml.nml.nml.FitzHughNagumo1969Cell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, a=None, b=None, I=None, phi=None, V0=None, W0=None, gds_collector_=None, **kwargs_)

Bases: BaseCell

Parameters

- \( a \) (none) –
- \( b \) (none) –
- \( I \) (none) – plays the role of an external injected current
- \( \phi \) (none) –
- \( V_0 \) (none) –
- \( W_0 \) (none) –

**FitzHughNagumoCell**

class neuroml.nml.nml.FitzHughNagumoCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, I=None, gds_collector_=None, **kwargs_)

Bases: BaseCell

FitzHughNagumoCell – Simple dimensionless model of spiking cell from FitzHugh and Nagumo. Superseded by FitzHughNagumo1969Cell (See https://github.com/NeuroML/NeuroML2/issues/42)

Parameters
- \( I \) (none) –

**FixedFactorConcentrationModel**

class neuroml.nml.nml.FixedFactorConcentrationModel(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, ion=None, resting_conc=None, decay_constant=None, rho=None, gds_collector_=None, **kwargs_)

Bases: Standalone

FixedFactorConcentrationModel – Model of buffering of concentration of an ion ( currently hard coded to be calcium, due to requirement for \( iCa \) ) which has a baseline level \( \text{restingConc} \) and tends to this value with time course \( \text{decayConstant} \). A fixed factor \( \rho \) is used to scale the incoming current independently of the size of the compartment to produce a concentration change.

Parameters
- \( \text{restingConc} \) (concentration) –
- \( \text{decayConstant} \) (time) –
- \( \rho \) (rho_factor) –
**ForwardTransition**

class neuroml.nml.nml.ForwardTransition(neuro_lex_id=None, id=None, from_=None, to=None, anytypeobjs_=None, gds_collector_=None, **kwargs_)

Bases: Base

ForwardTransition – A forward only KSTransition for a gateKS which specifies a rate (type baseHHRate) which follows one of the standard Hodgkin Huxley forms (e.g. HHExpRate, HHSigmoidRate, HHExpLinearRate)

**GapJunction**

class neuroml.nml.nml.GapJunction(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, conductance=None, gds_collector_=None, **kwargs_)

Bases: BaseSynapse

GapJunction – Gap junction/single electrical connection

**Parameters**

conductance (conduction) –

**GateFractional**

class neuroml.nml.nml.GateFractional(neuro_lex_id=None, id=None, instances=None, notes=None, q10_settings=None, sub_gates=None, gds_collector_=None, **kwargs_)

Bases: Base

GateFractional – Gate composed of subgates contributing with fractional conductance

**Parameters**

instances (none) –

**GateFractionalSubgate**

class neuroml.nml.nml.GateFractionalSubgate(neuro_lex_id=None, id=None, fractional_conductance=None, notes=None, q10_settings=None, steady_state=None, time_course=None, gds_collector_=None, **kwargs_)

Bases: Base
GateHHInstantaneous

class neuroml.nml.nml.GateHHInstantaneous(neuro_lex_id=None, id=None, instances=None, notes=None, steady_state=None, gds_collector_=None, **kwargs_)

    Bases: Base

    GateHHInstantaneous – Gate which follows the general Hodgkin Huxley formalism but is instantaneous, so tau = 0 and gate follows exactly inf value

    Parameters
    instances (none) –

GateHHRates

class neuroml.nml.nml.GateHHRates(neuro_lex_id=None, id=None, instances=None, notes=None, q10_settings=None, forward_rate=None, reverse_rate=None, gds_collector_=None, **kwargs_)

    Bases: Base

    GateHHRates – Gate which follows the general Hodgkin Huxley formalism

    Parameters
    instances (none) –

GateHHRatesInf

class neuroml.nml.nml.GateHHRatesInf(neuro_lex_id=None, id=None, instances=None, notes=None, q10_settings=None, forward_rate=None, reverse_rate=None, steady_state=None, gds_collector_=None, **kwargs_)

    Bases: Base

    GateHHRatesInf – Gate which follows the general Hodgkin Huxley formalism

    Parameters
    instances (none) –

GateHHRatesTau

class neuroml.nml.nml.GateHHRatesTau(neuro_lex_id=None, id=None, instances=None, notes=None, q10_settings=None, forward_rate=None, reverse_rate=None, time_course=None, gds_collector_=None, **kwargs_)

    Bases: Base

    GateHHRatesTau – Gate which follows the general Hodgkin Huxley formalism

    Parameters
    instances (none) –
GateHHRatesTauInf

class neuroml.nml.nml.GateHHRatesTauInf(neuro_lex_id=None, id=None, instances=None, notes=None, q10_settings=None, forward_rate=None, reverse_rate=None, time_course=None, steady_state=None, gds_collector_=None, **kwargs_)

Bases: Base

GateHHRatesTauInf – Gate which follows the general Hodgkin Huxley formalism

Parameters

  instances (none) –

GateHHTauInf

class neuroml.nml.nml.GateHHTauInf(neuro_lex_id=None, id=None, instances=None, notes=None, q10_settings=None, time_course=None, steady_state=None, gds_collector_=None, **kwargs_)

Bases: Base

GateHHTauInf – Gate which follows the general Hodgkin Huxley formalism

Parameters

  instances (none) –

GateHHUndetermined

class neuroml.nml.nml.GateHHUndetermined(neuro_lex_id=None, id=None, instances=None, type=None, notes=None, q10_settings=None, forward_rate=None, reverse_rate=None, time_course=None, steady_state=None, sub_gates=None, gds_collector_=None, **kwargs_)

Bases: Base

GateHHUndetermined – Note all sub elements for gateHHRates, gateHHRatesTau, gateFractional etc. allowed here. Which are valid should be constrained by what type is set

GateKS

class neuroml.nml.nml.GateKS(neuro_lex_id=None, id=None, instances=None, notes=None, q10_settings=None, closed_states=None, open_states=None, forward_transition=None, reverse_transition=None, tau_inf_transition=None, gds_collector_=None, **kwargs_)

Bases: Base

GateKS – A gate which consists of multiple KSSate s and KSTransition s giving the rates of transition between them

Parameters

  instances (none) –
GradedSynapse

class neuroml.nml.GradedSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, conductance=None, delta=None, Vth=None, k=None, erev=None, gds_collector_=None, **kwargs_)

Bases: BaseSynapse


Parameters

- conductance (conductance)
- delta (voltage) – Slope of the activation curve
- k (per_time) – Rate constant for transmitter-receptor dissociation rate
- Vth (voltage) – The half-activation voltage of the synapse
- erev (voltage) – The reversal potential of the synapse

GridLayout

class neuroml.nmlGridLayout(x_size=None, y_size=None, z_size=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

HHRate

class neuroml.nml.HHRate(type=None, rate=None, midpoint=None, scale=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

HHTime

class neuroml.nml.HHTime(type=None, rate=None, midpoint=None, scale=None, tau=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

HHVariable

class neuroml.nml.HHVariable(type=None, rate=None, midpoint=None, scale=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper
HH_cond_exp

class neuroml.nml.nml.HH_cond_exp(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, v_offset=None, e_rev_E=None, e_rev_I=None, e_rev_K=None, e_rev_Na=None, e_rev_leak=None, g_leak=None, gbar_K=None, gbar_Na=None, gds_collector_=None, **kwargs_)

Bases: basePyNNCell

HH_cond_exp – Single-compartment Hodgkin-Huxley-type neuron with transient sodium and delayed-rectifier potassium currents using the ion channel models from Traub.

Parameters

- gbar_K (none) –
- gbar_Na (none) –
- g_leak (none) –
- e_rev_K (none) –
- e_rev_Na (none) –
- e_rev_leak (none) –
- v_offset (none) –
- e_rev_E (none) –
- e_rev_I (none) –
- cm (none) –
- i_offset (none) –
- tau_syn_E (none) – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- tau_syn_I (none) – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell

:type tau_syn_I: none :param v_init: :type v_init: none

IF_cond_alpha

class neuroml.nml.nml.IF_cond_alpha(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, tau_m=None, tau_refrac=None, v_reset=None, v_rest=None, v_thresh=None, e_rev_E=None, e_rev_I=None, gds_collector_=None, **kwargs_)

Bases: basePyNNIaFCondCell

IF_cond_alpha – Leaky integrate and fire model with fixed threshold and alpha-function-shaped post-synaptic conductance

Parameters

- e_rev_E (none) – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
• **e_rev_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell

• **tau_refrac** *(none)* –

• **v_thresh** *(none)* –

• **tau_m** *(none)* –

• **v_rest** *(none)* –

• **v_reset** *(none)* –

• **cm** *(none)* –

• **i_offset** *(none)* –

• **tau_syn_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell

• **tau_syn_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell

• **v_init** *(none)* –

---

**IF_cond_exp**

```python
class neuroml.nml.nml.IF_cond_exp(neuro_lex_id=None, id=None, metaid=None, notes=None,
  properties=None, annotation=None, cm=None, i_offset=None,
  tau_syn_E=None, tau_syn_I=None, v_init=None, tau_m=None,
  tau_refrac=None, v_rest=None, v_reset=None, v_thresh=None,
  e_rev_E=None, e_rev_I=None, gds_collector_=None, **kwargs_)
```

**Bases:** basePyNNIaFCondCell

IF_cond_exp – Leaky integrate and fire model with fixed threshold and exponentially-decaying post-synaptic conductance

**Parameters**

• **e_rev_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell

• **e_rev_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell

• **tau_refrac** *(none)* –

• **v_thresh** *(none)* –

• **tau_m** *(none)* –

• **v_rest** *(none)* –

• **v_reset** *(none)* –

• **cm** *(none)* –

• **i_offset** *(none)* –

• **tau_syn_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell

• **tau_syn_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell
• **v_init** *(none)* –

**IF_curr_alpha**

```python
class neuroml.nml.nml.IF_curr_alpha(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, tau_m=None, tau_refrac=None, v_reset=None, v_rest=None, v_thresh=None, gds_collector_=None, **kwargs_)
```

Bases: `basePyNNIaFCell`

**IF_curr_alpha** – Leaky integrate and fire model with fixed threshold and alpha-function-shaped post-synaptic current

**Parameters**

- **tau_refrac** *(none)* –
- **v_thresh** *(none)* –
- **tau_m** *(none)* –
- **v_rest** *(none)* –
- **v_reset** *(none)* –
- **cm** *(none)* –
- **i_offset** *(none)* –
- **tau_syn_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell
- **tau_syn_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell
- **v_init** *(none)* –

**IF_curr_exp**

```python
class neuroml.nml.nml.IF_curr_exp(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, tau_m=None, tau_refrac=None, v_reset=None, v_rest=None, v_thresh=None, gds_collector_=None, **kwargs_)
```

Bases: `basePyNNIaFCell`

**IF_curr_exp** – Leaky integrate and fire model with fixed threshold and decaying-exponential post-synaptic current

**Parameters**

- **tau_refrac** *(none)* –
- **v_thresh** *(none)* –
- **tau_m** *(none)* –
- **v_rest** *(none)* –
- **v_reset** *(none)* –
• cm (none) –
• _i_offset (none) –
• tau_syn_E (none) – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell
• tau_syn_I (none) – This parameter is never used in the NeuroML2 description of this cell!
  Any synapse producing a current can be placed on this cell
• v_init (none) –

IafCell

class neuroml.nml.nml.IafCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, leak_reversal=None, thresh=None, reset=None, C=None, leak_conductance=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: BaseCell

IafCell – Integrate and fire cell with capacitance C, leakConductance and leakReversal

Parameters
• leakConductance (conductance) –
• leakReversal (voltage) –
• thresh (voltage) –
• reset (voltage) –
• C (capacitance) – Total capacitance of the cell membrane

IafRefCell

class neuroml.nml.nml.IafRefCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, leak_reversal=None, thresh=None, reset=None, C=None, leak_conductance=None, refract=None, gds_collector_=None, **kwargs_)

Bases: IafCell

IafRefCell – Integrate and fire cell with capacitance C, leakConductance, leakReversal and refractory period refract

Parameters
• refract (time) –
• leakConductance (conductance) –
• leakReversal (voltage) –
• thresh (voltage) –
• reset (voltage) –
• C (capacitance) – Total capacitance of the cell membrane
IafTauCell

class neuroml.nml.nml.IafTauCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, leak_reversal=None, thresh=None, reset=None, tau=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: BaseCell

IafTauCell – Integrate and fire cell which returns to its leak reversal potential of leakReversal with a time constant tau

Parameters

- leakReversal (voltage) –
- tau (time) –
- thresh (voltage) – The membrane potential at which to emit a spiking event and reset voltage
- reset (voltage) – The value the membrane potential is reset to on spiking

IafTauRefCell

class neuroml.nml.nml.IafTauRefCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, leak_reversal=None, thresh=None, reset=None, tau=None, refract=None, gds_collector_=None, **kwargs_)

Bases: IafTauCell

IafTauRefCell – Integrate and fire cell which returns to its leak reversal potential of leakReversal with a time course tau. It has a refractory period of refract after spiking

Parameters

- refract (time) –
- leakReversal (voltage) –
- tau (time) –
- thresh (voltage) – The membrane potential at which to emit a spiking event and reset voltage
- reset (voltage) – The value the membrane potential is reset to on spiking

Include

class neuroml.nml.nml.Include(segment_groups=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Include – Include all members of another segmentGroup in this group
IncludeType

class neuroml.nml.IncludeType(href=None, gds_collector_=None, **kwargs_)
    Bases: GeneratedsSuper

InhomogeneousParameter

class neuroml.nml.InhomogeneousParameter(neuro_lex_id=None, id=None, variable=None, metric=None, proximal=None, distal=None, gds_collector_=None, **kwargs_)
    Bases: Base
    InhomogeneousParameter – An inhomogeneous parameter specified across the segmentGroup (see variableParameter for usage).

InhomogeneousValue

class neuroml.nml.InhomogeneousValue(inhomogeneous_parameters=None, value=None, gds_collector_=None, **kwargs_)
    Bases: GeneratedsSuper
    InhomogeneousValue – Specifies the value of an inhomogeneousParameter. For usage see variableParameter

InitMembPotential

class neuroml.nml.InitMembPotential(value=None, segment_groups='all', gds_collector_=None, **kwargs_)
    Bases: GeneratedsSuper
    InitMembPotential – Explicitly set initial membrane potential for the cell
    Parameters
    value (voltage) –

Input

class neuroml.nml.Input(id=None, target=None, destination=None, segment_id=None, fraction_along=None, extensiontype_=None, gds_collector_=None, **kwargs_)
    Bases: GeneratedsSuper
    Input – Specifies a single input to a target, optionally giving the segmentId (default 0) and fractionAlong the segment (default 0.5).
    get_fraction_along()
        Get fraction along.
        Returns 0.5 if fraction_along was not set.
libNeuroML Documentation, Release 0.3.1

get_segment_id()
Get the ID of the segment.
Returns 0 if segment_id was not set.

get_target_cell_id()
Get ID of target cell.

InputList
class neuroml.nml.nml.InputList(neuro_lex_id=None, id=None, populations=None, component=None, input=None, input_ws=None, gds_collector_=None, **kwargs_)
Bases: Base
InputList – An explicit list of inputs to a population.
exportHdf5(h5file, h5Group)
Export to HDF5 file.

InputW
class neuroml.nml.nml.InputW(id=None, target=None, destination=None, segment_id=None, fraction_along=None, weight=None, gds_collector_=None, **kwargs_)
Bases: Input
InputW – Specifies input lists. Can set weight to scale individual inputs.
Parameters
weight (none) –

get_weight()
Get weight.
If weight is not set, the default value of 1.0 is returned.

Instance
class neuroml.nml.nml.Instance(id=None, i=None, j=None, k=None, location=None, gds_collector_=None, **kwargs_)
Bases: GeneratedsSuper
Instance – Specifies a single instance of a component in a population (placed at location).

InstanceRequirement
class neuroml.nml.nml.InstanceRequirement(name=None, type=None, gds_collector_=None, **kwargs_)
Bases: GeneratedsSuper
IntracellularProperties

class neuroml.nml.nml.IntracellularProperties(species=None, resistivities=None, extensiontype_=None, gds_collector_=None, **kwargs_):

Bases: GeneratedsSuper

IntracellularProperties – Biophysical properties related to the intracellular space within the cell, such as the resistivity and the list of ionic species present. caConc and caConcExt are explicitly exposed here to facilitate accessing these values from other Components, even though caConcExt is clearly not an intracellular property

IntracellularProperties2CaPools

class neuroml.nml.nml.IntracellularProperties2CaPools(species=None, resistivities=None, gds_collector_=None, **kwargs_):

Bases: IntracellularProperties

IntracellularProperties2CaPools – Variant of intracellularProperties with 2 independent Ca pools

IonChannel

class neuroml.nml.nml.IonChannel(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, q10_conductance_scalings=None, species=None, type=None, conductance=None, gates=None, gate_hh_rates=None, gate_h_hrates_taus=None, gate_hh_tau_infs=None, gate_h_hrates_infs=None, gate_h_hrates_tau_infs=None, gate_hh_instantaneouses=None, gate_fractionals=None, extensiontype_=None, gds_collector_=None, **kwargs_):

Bases: IonChannelScalable

IonChannel – Note ionChannel and ionChannelHH are currently functionally identical. This is needed since many existing examples use ionChannel, some use ionChannelHH. NeuroML v2beta4 should remove one of these, probably ionChannelHH.

Parameters

conductance (conductance) –

IonChannelHH

class neuroml.nml.nml.IonChannelHH(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, q10_conductance_scalings=None, species=None, type=None, conductance=None, gates=None, gate_hh_rates=None, gate_h_hrates_taus=None, gate_hh_tau_infs=None, gate_h_hrates_infs=None, gate_h_hrates_tau_infs=None, gate_hh_instantaneouses=None, gate_fractionals=None, gds_collector_=None, **kwargs_):

Bases: IonChannel

IonChannelHH – Note ionChannel and ionChannelHH are currently functionally identical. This is needed since many existing examples use ionChannel, some use ionChannelHH. NeuroML v2beta4 should remove one of these, probably ionChannelHH.
Parameters

conductance (conductance) –

IonChannelKS

class IonChannelKS(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, species=None, conductance=None, gate_kses=None, gds_collector_=None, **kwargs_)

Bases: Standalone

A kinetic scheme based ion channel with multiple gateKS s, each of which consists of multiple KSState s and KSTransition s giving the rates of transition between them IonChannelKS – A kinetic scheme based ion channel with multiple gateKS s, each of which consists of multiple KSState s and KSTransition s giving the rates of transition between them

Parameters

cconductance (conductance) –

IonChannelScalable

class IonChannelScalable(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, q10_conductance_scalings=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: Standalone

IonChannelVShift

class IonChannelVShift(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, q10_conductance_scalings=None, species=None, type=None, conductance=None, gates=None, gate_hh_rates=None, gate_h_hrates_taus=None, gate_h_hrates_taus_inf=None, gate_h_hrates_taus_inf=None, gate_h_hrates_taus_inf=None, gate_h_hrates_taus_inf=None, gate_h_hrates_taus_inf=None, gate_h_hrates_taus_inf=None, gate_h_hrates_taus_inf=None, v_shift=None, gds_collector_=None, **kwargs_)

Bases: IonChannel

IonChannelVShift – Same as ionChannel, but with a vShift parameter to change voltage activation of gates. The exact usage of vShift in expressions for rates is determined by the individual gates.

Parameters

• vShift (voltage) –
• conductance (conductance) –
libNeuroML Documentation, Release 0.3.1

Izhikevich2007Cell

class neuroml.nml.Izhikevich2007Cell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, C=None, v0=None, k=None, vr=None, vt=None, vpeak=None, a=None, b=None, c=None, d=None, gds_collector_=None, **kwargs_)

Bases: BaseCellMembPotCap

Izhikevich2007Cell – Cell based on the modified Izhikevich model in Izhikevich 2007, Dynamical systems in neuroscience, MIT Press

Parameters

- **v0** (voltage) –
- **k** (conductance_per_voltage) –
- **vr** (voltage) –
- **vt** (voltage) –
- **vpeak** (voltage) –
- **a** (per_time) –
- **b** (conductance) –
- **c** (voltage) –
- **d** (current) –
- **C** (capacitance) – Total capacitance of the cell membrane

IzhikevichCell

class neuroml.nml.IzhikevichCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, v0=None, thresh=None, a=None, b=None, c=None, d=None, gds_collector_=None, **kwargs_)

Bases: BaseCell

IzhikevichCell – Cell based on the 2003 model of Izhikevich, see http://izhikevich.org/publications/spikes.htm

Parameters

- **v0** (voltage) – Initial membrane potential
- **a** (none) – Time scale of the recovery variable U
- **b** (none) – Sensitivity of U to the subthreshold fluctuations of the membrane potential V
- **c** (none) – After-spike reset value of V
- **d** (none) – After-spike increase to U
- **thresh** (voltage) – Spike threshold
LEMS_Property

class neuroml.nml.nml.LEMS_Property(name=None, dimension=None, description=None, default_value=None, gds_collector_=None, **kwargs_)
  Bases: NamedDimensionalType

Layout

class neuroml.nml.nml.Layout(spaces=None, random=None, grid=None, unstructured=None, gds_collector_=None, **kwargs_)
  Bases: GeneratedsSuper

LinearGradedSynapse

class neuroml.nml.nml.LinearGradedSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, conductance=None, gds_collector_=None, **kwargs_)
  Bases: BaseSynapse
  LinearGradedSynapse – Behaves just like a one way gap junction.
    Parameters
    conductance (conductance) –

Location

class neuroml.nml.nml.Location(x=None, y=None, z=None, gds_collector_=None, **kwargs_)
  Bases: GeneratedsSuper
  Location – Specifies the (x, y, z) location of a single instance of a component in a population
    Parameters
    • x (none) –
    • y (none) –
    • z (none) –

Member

class neuroml.nml.nml.Member(segments=None, gds_collector_=None, **kwargs_)
  Bases: GeneratedsSuper
  Member – A single identified segment which is part of the segmentGroup
MembraneProperties

class neuroml.nml.nml.MembraneProperties(channel_populations=None, channel_densities=None, 
channel_density_v_shifts=None, channel_density_nernsts=None, channel_density_ghks=None, channel_density_ghk2s=None, 
channel_density_non_uniforms=None, channel_density_non_uniform_nernsts=None, channel_density_non_uniform_ghks=None, 
spike_threses=None, specific_capacitances=None, init_memb_potentials=None, extensiontype_=None, 
gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

MembraneProperties – Properties specific to the membrane, such as the populations of channels, channelDensities, specificCapacitance, etc.

MembraneProperties2CaPools

class neuroml.nml.nml.MembraneProperties2CaPools(channel_populations=None, channel_densities=None, 
channel_density_v_shifts=None, channel_density_nernsts=None, channel_density_ghks=None, channel_density_ghk2s=None, 
channel_density_non_uniforms=None, channel_density_non_uniform_nernsts=None, channel_density_non_uniform_ghks=None, 
spike_threses=None, specific_capacitances=None, init_memb_potentials=None, extensiontype_=None, 
gchannel_density_nernst_ca2s=None, gds_collector_=None, **kwargs_)

Bases: MembraneProperties

MembraneProperties2CaPools – Variant of membraneProperties with 2 independent Ca pools

MixedContainer:

Morphology

class neuroml.nml.nml.Morphology(neuro_lex_id=None, id= None, metaid=None, notes=None, 
properties=None, annotation=None, segments=None, segment_groups=None, gds_collector_=None, **kwargs_)

Bases: Standalone

Morphology – The collection of segment s which specify the 3D structure of the cell, along with a number of segmentGroup s

property num_segments

Get the number of segments included in this cell morphology.
Returns
number of segments

Return type
int

NamedDimensionalType
class neuroml.nml.nml.NamedDimensionalType(name=None, dimension=None, description=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

NamedDimensionalVariable
class neuroml.nml.nml.NamedDimensionalVariable(name=None, dimension=None, description=None, exposure=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Network
class neuroml.nml.nml.Network(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, type=None, temperature=None, spaces=None, regions=None, extracellular_properties=None, populations=None, cell_sets=None, synaptic_connections=None, projections=None, electrical_projections=None, continuous_projections=None, explicit_inputs=None, input_lists=None, gds_collector_=None, **kwargs_)

Bases: Standalone

Network – Network containing: population s ( potentially of type populationList , and so specifying a list of cell location s ); projection s ( with lists of connection s ) and/or explicitConnection s ; and inputList s ( with lists of input s ) and/or explicitInput s . Note: often in NeuroML this will be of type networkWithTemperature if there are temperature dependent elements ( e. g. ion channels ).

exportHdf5(h5file, h5Group)
    Export to HDF5 file.

get_by_id(id)
    Get a component by its ID

Parameters
    id (str) – ID of component to find

Returns
    component with specified ID or None if no component with specified ID found
NeuroMLDocument

class neuroml.nml.nml.NeuroMLDocument:
    
    neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, includes=None, extracellular_properties=None, intracellular_properties=None, morphology=None, ion_channel=None, ion_channel_hhs=None, ion_channel_v_shifts=None, ion_channel_kses=None, decaying_pool_concentration_models=None, fixed_factor_concentration_models=None, alpha_current_synapses=None, alpha_synapses=None, exp_one_synapses=None, exp_two_synapses=None, exp_three_synapses=None, blocking_plastic_synapses=None, double_synapses=None, gap_junctions=None, silent_synapses=None, linear_graded_synapses=None, graded_synapses=None, biophysical_properties=None, cells=None, cell2_ca_poolses=None, base_cells=None, iaf_tau_cells=None, iaf_tau_ref_cells=None, iaf_cells=None, iaf_ref_cells=None, izhikevich_cells=None, izhikevich2007_cells=None, ad_ex_ia_f_cells=None, fitz_hugh_nagumo_cells=None, fitz_hugh_nagumo1969_cells=None, pinsky_rinzel_ca3_cells=None, pulse_generators=None, pulse_generator_dls=None, sine_generators=None, sine_generator_dls=None, ramp_generators=None, ramp_generator_dls=None, compound_inputs=None, compound_input_dls=None, voltage_clamps=None, voltage_clamp_triples=None, spike_arrays=None, timed_synaptic_inputs=None, spike_generators=None, spike_generator_randoms=None, spike_generator_poissons=None, spike_generator_ref_poissons=None, poisson_firing_synapses=None, transient_poisson_firing_synapses=None, IF_curr_alpha=None, IF_curr_exp=None, IF_cond_alpha=None, IF_cond_exp=None, EIF_cond_exp_isfa_ista=None, EIF_cond_alpha_isfa_ista=None, HH_cond_exp=None, exp_cond_synapses=None, alpha_cond_synapses=None, exp_curr_synapses=None, alpha_curr_synapses=None, SpikeSourcePoisson=None, networks=None, ComponentType=None, gds_collector_=None, **kwargs_)

Bases: Standalone

append(element)

Append an element

Parameters

  element (Object) – element to append

get_by_id(id)

Get a component by specifying its ID.

Parameters

  id (str) – id of Component to get

Returns

Component with given ID or None if no Component with provided ID was found
summary(show_includes=True, show_non_network=True)

Get a pretty-printed summary of the complete NeuroMLDocument.
This includes information on the various Components included in the NeuroMLDocument: networks, cells, projections, synapses, and so on.

OpenState
class neuroml.nml.nml.OpenState(neuro_lex_id=None, id=None, gds_collector_=None, **kwargs_)
    Bases: Base

OpenState – A KSState with relativeConductance of 1
Parameters
    relativeConductance (none) –

Parameter
class neuroml.nml.nml.Parameter(name=None, dimension=None, description=None, gds_collector_=None, **kwargs_)
    Bases: NamedDimensionalType

Path
class neuroml.nml.nml.Path(from_=None, to=None, gds_collector_=None, **kwargs_)
    Bases: GeneratedSuper

Path – Include all the segment s between those specified by from and to , inclusive

PinskyRinzelCA3Cell
class neuroml.nml.nml.PinskyRinzelCA3Cell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, i_soma=None, i_dend=None, gc=None, g_ls=None, g_ld=None,
    g_na=None, g_kd=None, g_ca=None, g_kahp=None, g_kc=None, g_nmda=None, g_ampa=None, e_na=None,
    e_ca=None, e_k=None, e_l=None, qd0=None, pp=None, alphac=None, betac=None, cm=None, gds_collector_=None,
    **kwargs_)
    Bases: BaseCell

PinskyRinzelCA3Cell – Reduced CA3 cell model from Pinsky and Rinzel 1994. See https://github.com/OpenSourceBrain/PinskyRinzelModel
Parameters
    • iSoma (currentDensity) –
    • iDend (currentDensity) –
    • gLs (conductanceDensity) –
    • gLd (conductanceDensity) –
libNeuroML Documentation, Release 0.3.1

- $g_{Na}$ (conductanceDensity)
- $g_{Kdr}$ (conductanceDensity)
- $g_{Ca}$ (conductanceDensity)
- $g_{Kahp}$ (conductanceDensity)
- $g_{KC}$ (conductanceDensity)
- $g_c$ (conductanceDensity)
- $e_{Na}$ (voltage)
- $e_{Ca}$ (voltage)
- $e_{K}$ (voltage)
- $e_{L}$ (voltage)
- $pp$ (none)
- $cm$ (specificCapacitance)
- $\alpha_{c}$ (none)
- $\beta_{c}$ (none)
- $g_{Nmda}$ (conductanceDensity)
- $g_{Ampa}$ (conductanceDensity)
- $q_d_{0}$ (none)

PlasticityMechanism

class neuroml.nml.nml.PlasticityMechanism(type=None, init_release_prob=None, tau_rec=None, tau_fac=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Point3DWithDiam

class neuroml.nml.nml.Point3DWithDiam(x=None, y=None, z=None, diameter=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Point3DWithDiam – Base type for ComponentTypes which specify an (x, y, z) coordinate along with a diameter. Note: no dimension used in the attributes for these coordinates! These are assumed to have dimension micrometer (10^-6 m). This is due to micrometers being the default option for the majority of neuronal morphology formats, and dimensions are omitted here to facilitate reading and writing of morphologies in NeuroML.

Parameters
- $x$ (none) – x coordinate of the point. Note: no dimension used, see description of point3DWithDiam for details.
- $y$ (none) – y coordinate of the point. Note: no dimension used, see description of point3DWithDiam for details.
- $z$ (none) – z coordinate of the point. Note: no dimension used, see description of point3DWithDiam for details.
• **diameter** *(none)* – Diameter of the ppoint. Note: no dimension used, see description of `Point3DWithDiam` for details.

**distance_to** *(other_3d_point)*
Find the distance between this point and another.

**Parameters**
- **other_3d_point** *(Point3DWithDiam)* – other 3D point to calculate distance to

**Returns**
distance between the two points

**Return type**
float

### PoissonFiringSynapse

class **neuroml.nml.nml.PoissonFiringSynapse**(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, average_rate=None, synapse=None, spike_target=None, gds_collector_=None, **kwargs_)

Bases: **Standalone**

PoissonFiringSynapse – Poisson spike generator firing at **averageRate**, which is connected to single **synapse** that is triggered every time a spike is generated, producing an input current. See also **transientPoissonFiringSynapse**.

**Parameters**
- **averageRate** *(per_time)* – The average rate at which spikes are emitted

### Population

class **neuroml.nml.nml.Population**(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, component=None, size=None, type=None, extracellular_properties=None, layout=None, instances=None, gds_collector_=None, **kwargs_)

Bases: **Standalone**

Population – A population of components, with just one parameter for the **size**, i.e. number of components to create. Note: quite often this is used with type= **populationList** which means the size is determined by the number of **instance** s ( with **location** s ) in the list. The **size** attribute is still set, and there will be a validation error if this does not match the number in the list.

**Parameters**
- **size** *(none)* – Number of instances of this Component to create when the population is instantiated

**exportHdf5**(h5file, h5Group)
Export to HDF5 file.

**get_size** ()
**Projection**

```python
class neuroml.nml.nml.Projection(neuro_lex_id=None, id=None, presynaptic_population=None, postsynaptic_population=None, synapse=None, connections=None, connection_wds=None, gds_collector_=None, **kwargs_)
```

**Bases:** `BaseProjection`

Projection – Projection from one population, `presynapticPopulation` to another, `postsynapticPopulation`, through synapse. Contains lists of `connection` or `connectionWD` elements.

```python
exportHdf5(h5file, h5Group)
```

Export to HDF5 file.

**Property**

```python
class neuroml.nml.nml.Property(tag=None, value=None, gds_collector_=None, **kwargs_)
```

**Bases:** `GeneratedsSuper`

Property – A property (a `tag` and `value` pair), which can be on any `baseStandalone` either as a direct child, or within an `Annotation`. Generally something which helps the visual display or facilitates simulation of a Component, but is not a core physiological property. Common examples include: `numberInternalDivisions`, equivalent of `nseg` in NEURON; `radius`, for a radius to use in graphical displays for abstract cells (i.e. without defined morphologies); `color`, the color to use for a `Population` or `populationList` of cells; `recommended_dt_ms`, the recommended timestep to use for simulating a `Network`, `recommended_duration_ms` the recommended duration to use when running a `Network`.

**ProximalDetails**

```python
class neuroml.nml.nml.ProximalDetails(translation_start=None, gds_collector_=None, **kwargs_)
```

**Bases:** `GeneratedsSuper`

**PulseGenerator**

```python
class neuroml.nml.nml.PulseGenerator(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, delay=None, duration=None, amplitude=None, gds_collector_=None, **kwargs_)
```

**Bases:** `Standalone`

PulseGenerator – Generates a constant current pulse of a certain `amplitude` for a specified `duration` after a `delay`. Scaled by `weight`, if set

**Parameters**

- `delay` *(time)* – Delay before change in current. Current is zero prior to this.
- `duration` *(time)* – Duration for holding current at amplitude. Current is zero after delay + duration.
- `amplitude` *(current)* – Amplitude of current pulse
PulseGeneratorDL

class neuroml.nml.nml.PulseGeneratorDL(neuro_lex_id=None, id=None, metaid=None, notes=None,
properties=None, annotation=None, delay=None, duration=None, amplitude=None, gds_collector_=None,
**kwargs_)

Bases: Standalone

PulseGeneratorDL – Dimensionless equivalent of pulseGenerator. Generates a constant current pulse of a certain amplitude for a specified duration after a delay. Scaled by weight, if set

Parameters

• delay (time) – Delay before change in current. Current is zero prior to this.
• duration (time) – Duration for holding current at amplitude. Current is zero after delay + duration.
• amplitude (none) – Amplitude of current pulse

Q10ConductanceScaling

class neuroml.nml.nml.Q10ConductanceScaling(q10_factor=None, experimental_temp=None,
gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

Q10ConductanceScaling – A value for the conductance scaling which varies as a standard function of the difference between the current temperature, temperature, and the temperature at which the conductance was originally determined, experimentalTemp

Parameters

• q10Factor (none) –
• experimentalTemp (temperature) –

Q10Settings

class neuroml.nml.nml.Q10Settings(type=None, fixed_q10=None, q10_factor=None,
experimental_temp=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

RampGenerator

class neuroml.nml.nml.RampGenerator(neuro_lex_id=None, id=None, metaid=None, notes=None,
properties=None, annotation=None, delay=None, duration=None,
start_amplitude=None, finish_amplitude=None, baseline_amplitude=None, gds_collector_=None,
**kwargs_)

Bases: Standalone

RampGenerator – Generates a ramping current after a time delay, for a fixed duration. During this time the current steadily changes from startAmplitude to finishAmplitude. Scaled by weight, if set

Parameters
• delay (time) – Delay before change in current. Current is baselineAmplitude prior to this.
• duration (time) – Duration for holding current at amplitude. Current is baselineAmplitude after delay + duration.
• startAmplitude (current) – Amplitude of linearly varying current at time delay
• finishAmplitude (current) – Amplitude of linearly varying current at time delay + duration
• baselineAmplitude (current) – Amplitude of current before time delay, and after time delay + duration

RampGeneratorDL

class neuroml.nml.nml.RampGeneratorDL(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, delay=None, duration=None, start_amplitude=None, finish_amplitude=None, baseline_amplitude=None, gds_collector_=None, **kwargs_)

Bases: Standalone

RampGeneratorDL – Dimensionless equivalent of rampGenerator. Generates a ramping current after a time delay, for a fixed duration. During this time the dimensionless current steadily changes from startAmplitude to finishAmplitude. Scaled by weight, if set

Parameters

• delay (time) – Delay before change in current. Current is baselineAmplitude prior to this.
• duration (time) – Duration for holding current at amplitude. Current is baselineAmplitude after delay + duration.
• startAmplitude (none) – Amplitude of linearly varying current at time delay
• finishAmplitude (none) – Amplitude of linearly varying current at time delay + duration
• baselineAmplitude (none) – Amplitude of current before time delay, and after time delay + duration

RandomLayout

class neuroml.nml.nml.RandomLayout(number=None, regions=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

ReactionScheme

class neuroml.nml.nml.ReactionScheme(neuro_lex_id=None, id=None, source=None, type=None, anytypeobjs_=None, gds_collector_=None, **kwargs_)

Bases: Base
Region

class neuroml.nml.nml.Region(neuro_lex_id=None, id=None, spaces=None, anytypeobjs_=None, gds_collector_=None, **kwargs_)

    Bases: Base

Region – Initial attempt to specify 3D region for placing cells. Work in progress...

Requirement

class neuroml.nml.nml.Requirement(name=None, dimension=None, description=None, gds_collector_=None, **kwargs_)

    Bases: NamedDimensionalType

Resistivity

class neuroml.nml.nml.Resistivity(value=None, segment_groups='all', gds_collector_=None, **kwargs_)

    Bases: GeneratedsSuper

    Resistivity – The resistivity, or specific axial resistance, of the cytoplasm

        Parameters

        value (resistivity) –

        validate_Nml2Quantity_resistivity(value)

        validate_Nml2Quantity_resistivity_patterns_ = [['\^(-?([0-9]*(\.[0-9]+)?([eE]-?[0-9]+)?[\s]*\(ohm_cm|kohm_cm|ohm_m\))$']]

ReverseTransition

class neuroml.nml.nml.ReverseTransition(neuro_lex_id=None, id=None, from_=None, to=None, anytypeobjs_=None, gds_collector_=None, **kwargs_)

    Bases: Base

    ReverseTransition – A reverse only KSTransition for a gateKS which specifies a rate ( type baseHHRate ) which follows one of the standard Hodgkin Huxley forms ( e.g. HHExpRate, HHSigmoidRate, HHExpLinearRate

Segment

class neuroml.nml.nml.Segment(neuro_lex_id=None, id=None, name=None, parent=None, proximal=None, distal=None, gds_collector_=None, **kwargs_)

    Bases: BaseNonNegativeIntegerId

    Segment – A segment defines the smallest unit within a possibly branching structure ( morphology ), such as a dendrite or axon. Its id should be a nonnegative integer ( usually soma/root = 0 ). Its end points are given by the proximal and distal points. The proximal point can be omitted, usually because it is the same as a point on the parent segment, see proximal for details. parent specifies the parent segment. The first segment of a cell ( with no parent ) usually represents the soma. The shape is normally a cylinder ( radii of the proximal and distal equal, but positions different ) or a conical frustum ( radii and positions different ). If the x, y, x positions of
the proximal and distal are equal, the segment can be interpreted as a sphere, and in this case the radii of these points must be equal. NOTE: LEMS does not yet support multicompartmental modelling, so the Dynamics here is only appropriate for single compartment modelling.

**property length**

Get the length of the segment.

- **Returns**
  - length of the segment

- **Return type**
  - float

**property surface_area**

Get the surface area of the segment.

- **Returns**
  - surface area of segment

- **Return type**
  - float

**property volume**

Get the volume of the segment.

- **Returns**
  - volume of segment

- **Return type**
  - float

---

### SegmentEndPoint

**class** `neuroml.nml.nml.SegmentEndPoint`(`segments=None, gds_collector_=None, **kwargs_`)

- **Bases:** GeneratedsSuper

---

### SegmentGroup

**class** `neuroml.nml.nml.SegmentGroup`(`neuro_lex_id=None, id=None, notes=None, properties=None, annotation=None, members=None, includes=None, paths=None, sub_trees=None, inhomogeneous_parameters=None, gds_collector_=None, **kwargs_`)

- **Bases:** Base

**SegmentGroup** – A method to describe a group of segment s in a morphology, e. g. soma_group, dendrite_group, axon_group. While a name is useful to describe the group, the neuroLexId attribute can be used to explicitly specify the meaning of the group, e. g. sao1044911821 for ‘Neuronal Cell Body’, sao1211023249 for ‘Dendrite’. The segment s in this group can be specified as: a list of individual member segments; a path, all of the segments along which should be included; a subTree of the cell to include; other segmentGroups to include (so all segments from those get included here). An inhomogeneousParameter can be defined on the region of the cell specified by this group (see variableParameter for usage).
**SegmentParent**

class neuroml.nml.nml.SegmentParent(segments=None, fraction_along='1', gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

**SilentSynapse**

class neuroml.nml.nml.SilentSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, gds_collector_=None, **kwargs_)

Bases: BaseSynapse

SilentSynapse – Dummy synapse which emits no current. Used as presynaptic endpoint for analog synaptic connection.

**SineGenerator**

class neuroml.nml.nml.SineGenerator(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, delay=None, phase=None, duration=None, amplitude=None, period=None, gds_collector_=None, **kwargs_)

Bases: Standalone

SineGenerator – Generates a sinusoidally varying current after a time delay, for a fixed duration. The period and maximum amplitude of the current can be set as well as the phase at which to start. Scaled by weight, if set.

Parameters

- **phase** *(none)* – Phase (between 0 and 2*pi) at which to start the varying current (i.e. at time given by delay)
- **delay** *(time)* – Delay before change in current. Current is zero prior to this.
- **duration** *(time)* – Duration for holding current at amplitude. Current is zero after delay + duration.
- **amplitude** *(current)* – Maximum amplitude of current
- **period** *(time)* – Time period of oscillation

**SineGeneratorDL**

class neuroml.nml.nml.SineGeneratorDL(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, delay=None, phase=None, duration=None, amplitude=None, period=None, gds_collector_=None, **kwargs_)

Bases: Standalone

SineGeneratorDL – Dimensionless equivalent of SineGenerator. Generates a sinusoidally varying current after a time delay, for a fixed duration. The period and maximum amplitude of the current can be set as well as the phase at which to start. Scaled by weight, if set.
Parameters

- **phase** *(none)* – Phase (between 0 and 2*π) at which to start the varying current (i.e. at time given by delay)
- **delay** *(time)* – Delay before change in current. Current is zero prior to this.
- **duration** *(time)* – Duration for holding current at amplitude. Current is zero after delay + duration.
- **amplitude** *(none)* – Maximum amplitude of current
- **period** *(time)* – Time period of oscillation

**Space**

class neuroml.nml.nml.Space(neuro_lex_id=None, id=None, based_on=None, structure=None, gds_collector_=None, **kwargs_)

    Bases: Base

**SpaceStructure**

class neuroml.nml.nml.SpaceStructure(x_spacing=None, y_spacing=None, z_spacing=None, x_start=0, y_start=0, z_start=0, gds_collector_=None, **kwargs_)

    Bases: GeneratedsSuper

**Species**

class neuroml.nml.nml.Species(id=None, concentration_model=None, ion=None, initial_concentration=None, initial_ext_concentration=None, segment_groups='all', gds_collector_=None, **kwargs_)

    Bases: GeneratedsSuper

Species – Description of a chemical species identified by **ion**, which has internal, **concentration**, and external, **extConcentration** values for its concentration

:|param initialConcentration : :type initialConcentration: concentration |
:|param initialExtConcentration: :type initialExtConcentration: concentration |

**SpecificCapacitance**

class neuroml.nml.nml.SpecificCapacitance(value=None, segment_groups='all', gds_collector_=None, **kwargs_)

    Bases: GeneratedsSuper

SpecificCapacitance – Capacitance per unit area

Parameters

- **value** *(specificCapacitance)* –
Spike

class neuroml.nml.nml.Spike(neuro_lex_id=None, id=None, time=None, gds_collector_=None, **kwargs_)

Bases: BaseNonNegativeIntegerId

Spike – Emits a single spike at the specified time

Parameters

time (time) – Time at which to emit one spike event

SpikeArray

class neuroml.nml.nml.SpikeArray(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, spikes=None, gds_collector_=None, **kwargs_)

Bases: Standalone

SpikeArray – Set of spike ComponentTypes, each emitting one spike at a certain time. Can be used to feed a predetermined spike train into a cell

SpikeGenerator

class neuroml.nml.nml.SpikeGenerator(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, period=None, gds_collector_=None, **kwargs_)

Bases: Standalone

SpikeGenerator – Simple generator of spikes at a regular interval set by period

Parameters

period (time) – Time between spikes. The first spike will be emitted after this time.

SpikeGeneratorPoisson

class neuroml.nml.nml.SpikeGeneratorPoisson(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, average_rate=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: Standalone

SpikeGeneratorPoisson – Generator of spikes whose ISI is distributed according to an exponential PDF with scale: 1 / averageRate

Parameters

averageRate (per_time) – The average rate at which spikes are emitted
SpikeGeneratorRandom

```python
class neuroml.nml.nml.SpikeGeneratorRandom(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, max_isi=None, min_isi=None, gds_collector_=None, **kwargs_)
```

**Bases:** Standalone

SpikeGeneratorRandom – Generator of spikes with a random interspike interval of at least **minISI** and at most **maxISI**

**Parameters**
- **maxISI** *(time)* – Maximum interspike interval
- **minISI** *(time)* – Minimum interspike interval

SpikeGeneratorRefPoisson

```python
class neuroml.nml.nml.SpikeGeneratorRefPoisson(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, average_rate=None, minimum_isi=None, gds_collector_=None, **kwargs_)
```

**Bases:** SpikeGeneratorPoisson

SpikeGeneratorRefPoisson – Generator of spikes whose ISI distribution is the maximum entropy distribution over [ *minimumISI*, +infinity ) with mean: 1 / averageRate

**Parameters**
- **minimumISI** *(time)* – The minimum interspike interval
- **averageRate** *(per_time)* – The average rate at which spikes are emitted

SpikeSourcePoisson

```python
class neuroml.nml.nml.SpikeSourcePoisson(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, start=None, duration=None, rate=None, gds_collector_=None, **kwargs_)
```

**Bases:** Standalone

SpikeSourcePoisson – Spike source, generating spikes according to a Poisson process.

**Parameters**
- **start** *(time)* –
- **duration** *(time)* –
- **rate** *(per_time)* –
**SpikeThresh**

class neuroml.nml.nml.SpikeThresh(value=None, segment_groups='all', gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

SpikeThresh – Membrane potential at which to emit a spiking event. Note, usually the spiking event will not be emitted again until the membrane potential has fallen below this value and rises again to cross it in a positive direction

Parameters

value (voltage) –

**Standalone**

class neuroml.nml.nml.Standalone(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: Base

Standalone – Elements which can stand alone and be referenced by id, e.g. cell, morphology.

**StateVariable**

class neuroml.nml.nml.StateVariable(name=None, dimension=None, description=None, exposure=None, gds_collector_=None, **kwargs_)

Bases: NamedDimensionalVariable

**SubTree**

class neuroml.nml.nml.SubTree(from_=None, to=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

SubTree – Include all the segment s distal to that specified by from in the segmentGroup

**SynapticConnection**

class neuroml.nml.nml.SynapticConnection(from_=None, to=None, synapse=None, destination=None, gds_collector_=None, **kwargs_)

Bases: GeneratedsSuper

SynapticConnection – Explicit event connection between named components, which gets processed via a new instance of a synapse component which is created on the target component
**TauInfTransition**

```python
class neuroml.nml.nml.TauInfTransition(neuro_lex_id=None, id=None, from_=None, to=None, steady_state=None, time_course=None, gds_collector_=None, **kwargs_)
```

Bases: `Base`

TauInfTransition – KS Transition specified in terms of time constant tau and steady state inf

**TimeDerivative**

```python
class neuroml.nml.nml.TimeDerivative(variable=None, value=None, gds_collector_=None, **kwargs_)
```

Bases: `GeneratedsSuper`

**TimedSynapticInput**

```python
class neuroml.nml.nml.TimedSynapticInput(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, synapse=None, spike_target=None, spikes=None, gds_collector_=None, **kwargs_)
```

Bases: `Standalone`

TimedSynapticInput – Spike array connected to a single synapse, producing a current triggered by each spike in the array.

**TransientPoissonFiringSynapse**

```python
class neuroml.nml.nml.TransientPoissonFiringSynapse(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, average_rate=None, delay=None, duration=None, synapse=None, spike_target=None, gds_collector_=None, **kwargs_)
```

Bases: `Standalone`

TransientPoissonFiringSynapse – Poisson spike generator firing at averageRate after a delay and for a duration, connected to single synapse that is triggered every time a spike is generated, providing an input current. Similar to ComponentType poissonFiringSynapse.

**Parameters**

- `averageRate` *(per_time)* –
- `delay` *(time)* –
- `duration` *(time)* –
UnstructuredLayout

```python
class neuroml.nml.UnstructuredLayout(number=None, gds_collector_=None, **kwargs_)
    Bases: GeneratedsSuper
```

VariableParameter

```python
class neuroml.nml.VariableParameter(parameter=None, segment_groups=None, inhomogeneous_value=None, gds_collector_=None, **kwargs_)
    Bases: GeneratedsSuper
```

VariableParameter – Specifies a `parameter` (e.g. `condDensity`) which can vary its value across a `segmentGroup`. The value is calculated from the `value` attribute of the `inhomogeneousValue` subelement. This element is normally a child of `channelDensityNonUniform`, `channelDensityNonUniformNernst` or `channelDensityNonUniformGHK` and is used to calculate the value of the conductance, etc. which will vary on different parts of the cell. The `segmentGroup` specified here needs to define an `inhomogeneousParameter` (referenced from `inhomogeneousParameter` in the `inhomogeneousValue`), which calculates a variable (e.g. `p`) varying across the cell (e.g. based on the path length from soma), which is then used in the `value` attribute of the `inhomogeneousValue` (so for example `condDensity = f(p)`)

VoltageClamp

```python
class neuroml.nml.VoltageClamp(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, delay=None, duration=None, target_voltage=None, simple_series_resistance=None, gds_collector_=None, **kwargs_)
    Bases: Standalone
```

VoltageClamp – Voltage clamp. Applies a variable current `i` to try to keep parent at `targetVoltage`. Not yet fully tested!!! Consider using voltageClampTriple!!

Parameters

- `delay (time)` – Delay before change in current. Current is zero prior to this.
- `duration (time)` – Duration for attempting to keep parent at `targetVoltage`. Current is zero after delay + duration.
- `targetVoltage (voltage)` – Current will be applied to try to get parent to this target voltage
- `simpleSeriesResistance (resistance)` – Current will be calculated by the difference in voltage between the target and parent, divided by this value
VoltageClampTriple

class neuroml.nml.nml.VoltageClampTriple(neuro_lex_id=None, id=None, metaid=None, notes=None,
properties=None, annotation=None, active=None,
delay=None, duration=None, conditioning_voltage=None,
testing_voltage=None, return_voltage=None,
simple_series_resistance=None, gds_collector_=None,
**kwargs_)

Bases: Standalone

VoltageClampTriple – Voltage clamp with 3 clamp levels. Applies a variable current \( i \) (through simpleSeriesResistance) to try to keep parent cell at conditioningVoltage until time delay, testingVoltage until delay + duration, and returnVoltage afterwards. Only enabled if active = 1.

Parameters

- **active** *(none)* – Whether the voltage clamp is active (1) or inactive (0).
- **delay** *(time)* – Delay before switching from conditioningVoltage to testingVoltage.
- **duration** *(time)* – Duration to hold at testingVoltage.
- **conditioningVoltage** *(voltage)* – Target voltage before time delay
- **testingVoltage** *(voltage)* – Target voltage between times delay and delay + duration
- **returnVoltage** *(voltage)* – Target voltage after time duration
- **simpleSeriesResistance** *(resistance)* – Current will be calculated by the difference in voltage between the target and parent, divided by this value

basePyNNCell

class neuroml.nml.nml.basePyNNCell(neuro_lex_id=None, id=None, metaid=None, notes=None,
properties=None, annotation=None, cm=None, i_offset=None,
tau_syn_E=None, tau_syn_I=None, v_init=None,
extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: BaseCell

basePyNNCell – Base type of any PyNN standard cell model. Note: membrane potential \( v \) has dimensions voltage, but all other parameters are dimensionless. This is to facilitate translation to and from PyNN scripts in Python, where these parameters have implicit units, see http://neuralensemble.org/trac/PyNN/wiki/StandardModels

Parameters

- **cm** *(none)* –
- **i_offset** *(none)* –
- **tau_syn_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **tau_syn_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **v_init** *(none)* –
libNeuroML Documentation, Release 0.3.1

basePyNNIaFCell

class neuroml.nml.nml.basePyNNIaFCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, tau_m=None, tau_refrac=None, v_reset=None, v_rest=None, v_thresh=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: basePyNNCell

basePyNNIaFCell – Base type of any PyNN standard integrate and fire model

Parameters

- **tau_refrac** *(none)* –
- **v_thresh** *(none)* –
- **tau_m** *(none)* –
- **v_rest** *(none)* –
- **v_reset** *(none)* –
- **cm** *(none)* –
- **i_offset** *(none)* –
- **tau_syn_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **tau_syn_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **v_init** *(none)* –

basePyNNIaFCondCell

class neuroml.nml.nml.basePyNNIaFCondCell(neuro_lex_id=None, id=None, metaid=None, notes=None, properties=None, annotation=None, cm=None, i_offset=None, tau_syn_E=None, tau_syn_I=None, v_init=None, tau_m=None, tau_refrac=None, v_reset=None, v_rest=None, v_thresh=None, e_rev_E=None, e_rev_I=None, extensiontype_=None, gds_collector_=None, **kwargs_)

Bases: basePyNNIaFCell

basePyNNIaFCondCell – Base type of conductance based PyNN IaF cell models

Parameters

- **e_rev_E** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **e_rev_I** *(none)* – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
- **tau_refrac** *(none)* –
- **v_thresh** *(none)* –
- **tau_m** *(none)* –
- **v_rest** *(none)* –
• v_reset (none) –
• cm (none) –
• i_offset (none) –
• tau_syn_E (none) – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
• tau_syn_I (none) – This parameter is never used in the NeuroML2 description of this cell! Any synapse producing a current can be placed on this cell
• v_init (none) –

1.3.2 loaders Module

class neuroml.loaders.ArrayMorphLoader

    Bases: object

    classmethod load(filepath)

        Right now this load method isn’t done in a very nice way. TODO: Complete refactoring.

class neuroml.loaders.NeuroMLHdf5Loader

    Bases: object

    classmethod load(src, optimized=False)

class neuroml.loaders.NeuroMLLoader

    Bases: object

    classmethod load(src)

class neuroml.loaders.SWCLoader

    Bases: object

    WARNING: Class defunct

    classmethod load_swc_single(src, name=None)

neuroml.loaders.print_(text, verbose=True)

neuroml.loaders.read_neuroml2_file(nml2_file_name, include_includes=False, verbose=False, already_included=[], print_method=<function print_>, optimized=False)

neuroml.loaders.read_neuroml2_string(nml2_string, include_includes=False, verbose=False, already_included=[], print_method=<function print_>, optimized=False, base_path=None)
1.3.3 writers Module

class neuroml.writers.ArrayMorphWriter
    Bases: object
    For now just testing a simple method which can write a morphology, not a NeuroMLDocument.
    classmethod write(data, filepath)

class neuroml.writers.NeuroMLHdf5Writer
    Bases: object
    classmethod write(nml_doc, h5_file_name, embed_xml=True, compress=True)

class neuroml.writers.NeuroMLWriter
    Bases: object
    classmethod write(nmldoc, file, close=True)
    Writes from NeuroMLDocument to nml file in future can implement from other types via chain of responsibility pattern.

1.3.4 utils Module

Utilities for checking generated code

neuroml.utils.add_all_to_document(nml_doc_src, nml_doc_tgt, verbose=False)
    Add all members of the source NeuroML document to the target NeuroML document.
    Parameters
        • nml_doc_src (NeuroMLDocument) – source NeuroML document to copy from
        • nml_doc_tgt (NeuroMLDocument) – target NeuroML document to copy to
        • verbose (bool) – control verbosity of working
    Raises
        Exception – if a member could not be copied.

neuroml.utils.append_to_element(parent, child)
    Append a child element to a parent Component
    Parameters
        • parent (Object) – parent NeuroML component to add element to
        • child (Object) – child NeuroML component to be added to parent
    Raises
        Exception – when the child could not be added to the parent

neuroml.utils.get_summary(nml_file_name)
    Get a summary of the given NeuroML file.
    Parameters
        • nml_file_name (str) – name of NeuroML file to get summary of
    Returns
        summary of provided file
libNeuroML Documentation, Release 0.3.1

Return type
str

neuroml.utils.has_segment_fraction_info(connections)
Check if connections include fraction information

Parameters
connections (list) – list of connection objects

Returns
True if connections include fragment information, otherwise False

Return type
Boolean

neuroml.utils.is_valid_neuroml2(file_name)
Check if a file is valid NeuroML2.

Parameters
file_name (str) – name of NeuroML file to check

Returns
True if file is valid, False if not.

Return type
Boolean

neuroml.utils.main()

neuroml.utils.print_summary(nml_file_name)
Print a summary of the NeuroML model in the given file.

Parameters
nml_file_name (str) – name of NeuroML file to print summary of

neuroml.utils.validate_neuroml2(file_name)
Validate a NeuroML document against the NeuroML schema specification.

Parameters
file_name (str) – name of NeuroML file to validate.

1.3.5 arraymorph Module

1.4 Examples

The examples in this section are intended to give in depth overviews of how to accomplish specific tasks with libNeu-
roML.

These examples are located in the neuroml/examples directory and can be tested to confirm they work by running the
run_all.py script.

Examples
- Examples
  - Creating a NeuroML morphology
  - Loading and modifying a file
1.4.1 Creating a NeuroML morphology

```python
'''
Example of connecting segments together to create a multicompartmental model of a cell.
'''

import neuroml
import neuroml.writers as writers

p = neuroml.Point3DWithDiam(x=0, y=0, z=0, diameter=50)

d = neuroml.Point3DWithDiam(x=50, y=0, z=0, diameter=50)

soma = neuroml.Segment(proximal=p, distal=d)
soma.name = "Soma"
soma.id = 0

# Make an axon with 100 compartments:

parent = neuroml.SegmentParent(segments=soma.id)
parent_segment = soma
axon_segments = []
seg_id = 1

for i in range(100):
    p = neuroml.Point3DWithDiam(
        x=parent_segment.distal.x,
        y=parent_segment.distal.y,
        z=parent_segment.distal.z,
        diameter=0.1,
    )

d = neuroml.Point3DWithDiam(
    x=parent_segment.distal.x + 10,
    y=parent_segment.distal.y,
    z=parent_segment.distal.z,
    diameter=0.1,
)
```

(continues on next page)
axon_segment = neuroml.Segment(proximal=p, distal=d, parent=parent)

axon_segment.id = seg_id

axon_segment.name = "axon_segment_" + str(axon_segment.id)

# now reset everything:
parent = neuroml.SegmentParent(segments=axon_segment.id)
parent_segment = axon_segment
seg_id += 1

axon_segments.append(axon_segment)

test_morphology = neuroml.Morphology()
test_morphology.segments.append(soma)
test_morphology.segments += axon_segments
test_morphology.id = "TestMorphology"

cell = neuroml.Cell()
cell.name = "TestCell"
cell.id = "TestCell"
cell.morphology = test_morphology

doc = neuroml.NeuroMLDocument(id="TestNeuroMLDocument")
doc.cells.append(cell)
nml_file = "tmp/testmorphwrite.nml"

writers.NeuroMLWriter.write(doc, nml_file)

print("Written morphology file to: " + nml_file)

##### Validate the NeuroML #####

from neuroml.utils import validate_neuroml2

validate_neuroml2(nml_file)

---

1.4.2 Loading and modifying a file

""
In this example an axon is built, a morphology is loaded, the axon is then connected to the loaded morphology.
"""

import neuroml
import neuroml.loaders as loaders
import neuroml.writers as writers
fn = "./test_files/Purk2M9s.nml"
doc = loaders.NeuroMLLoader.load(fn)
print("Loaded morphology file from: " + fn)

# get the parent segment:
parent_segment = doc.cells[0].morphology.segments[0]

parent = neuroml.SegmentParent(segments=parent_segment.id)

# make an axon:
seg_id = 5000  # need a way to get a unique id from a morphology
axon_segments = []
for i in range(10):
    p = neuroml.Point3DWithDiam(
        x=parent_segment.distal.x,
        y=parent_segment.distal.y,
        z=parent_segment.distal.z,
        diameter=0.1,
    )

    d = neuroml.Point3DWithDiam(
        x=parent_segment.distal.x + 10,
        y=parent_segment.distal.y,
        z=parent_segment.distal.z,
        diameter=0.1,
    )

    axon_segment = neuroml.Segment(proximal=p, distal=d, parent=parent)
    axon_segment.id = seg_id
    axon_segment.name = "axon_segment_" + str(axon_segment.id)

# now reset everything:
parent = neuroml.SegmentParent(segments=axon_segment.id)
parent_segment = axon_segment
seg_id += 1
axon_segments.append(axon_segment)
doc.cells[0].morphology.segments += axon_segments

nml_file = "/tmp/modified_morphology.nml"
writers.NeuroMLWriter.write(doc, nml_file)
print("Saved modified morphology file to: " + nml_file)

###### Validate the NeuroML ######
1.4.3 Building a network

""
Example to build a full spiking IaF network
through libNeuroML, save it as XML and validate it
""

```python
from neuroml import NeuroMLDocument
from neuroml import IafCell
from neuroml import Network
from neuroml import ExpOneSynapse
from neuroml import Population
from neuroml import PulseGenerator
from neuroml import ExplicitInput
from neuroml import SynapticConnection
import neuroml.writers as writers
from random import random

nml_doc = NeuroMLDocument(id="IafNet")

IafCell0 = IafCell(
    id="iaf0",
    C="1.0 nF",
    thresh="-50mV",
    reset="-65mV",
    leak_conductance="10 nS",
    leak_reversal="-65mV",
)

nml_doc.iaf_cells.append(IafCell0)

IafCell1 = IafCell(
    id="iaf1",
    C="1.0 nF",
    thresh="-50mV",
    reset="-65mV",
    leak_conductance="20 nS",
    leak_reversal="-65mV",
)

nml_doc.iaf_cells.append(IafCell1)

syn0 = ExpOneSynapse(id="syn0", gbase="65nS", erev="0mV", tau_decay="3ms")
```

(continues on next page)
nml_doc.exp_one_synapses.append(syn0)

net = Network(id="IafNet")
nml_doc.networks.append(net)

size0 = 5
pop0 = Population(id="IafPop0", component=IafCell0.id, size=size0)
net.populations.append(pop0)

size1 = 5
pop1 = Population(id="IafPop1", component=IafCell0.id, size=size1)
net.populations.append(pop1)

prob_connection = 0.5

for pre in range(0, size0):
    pg = PulseGenerator(
        id="pulseGen_%i" % pre,
        delay="0ms",
        duration="100ms",
        amplitude="%f nA" % (0.1 * random()),
    )

    nml_doc.pulse_generators.append(pg)

    exp_input = ExplicitInput(target="%s[%i]" % (pop0.id, pre), input=pg.id)

    net.explicit_inputs.append(exp_input)

for post in range(0, size1):
    # fromxx is used since from is Python keyword
    if random() <= prob_connection:
        syn = SynapticConnection(
            from_="%s[%i]" % (pop0.id, pre),
            synapse=syn0.id,
            to="%s[%i]" % (pop1.id, post),
        )

        net.synaptic_connections.append(syn)

nml_file = "tmp/testnet.nml"
writers.NeuroMLWriter.write(nml_doc, nml_file)

print("Written network file to: " + nml_file)

##### Validate the NeuroML #####

1.4. Examples
from neuroml.utils import validate_neuroml2

validate_neuroml2(nml_file)

### 1.4.4 Building a 3D network

Example to build a full spiking IaF network throught libNeuroML & save it as XML & validate it

```python
from neuroml import NeuroMLDocument
from neuroml import Network
from neuroml import ExpOneSynapse
from neuroml import Population
from neuroml import Property
from neuroml import Cell
from neuroml import Location
from neuroml import Instance
from neuroml import Morphology
from neuroml import Point3DWithDiam
from neuroml import Segment
from neuroml import SegmentParent
from neuroml import Projection
from neuroml import Connection

import neuroml.writers as writers
from random import random

soma_diam = 10
soma_len = 10
dend_diam = 2
dend_len = 10
dend_num = 10

def generateRandomMorphology():
    morphology = Morphology()
    p = Point3DWithDiam(x=0, y=0, z=0, diameter=soma_diam)
    d = Point3DWithDiam(x=soma_len, y=0, z=0, diameter=soma_diam)
    soma = Segment(proximal=p, distal=d, name="Soma", id=0)

    morphology.segments.append(soma)
    parent_seg = soma
```

(continues on next page)
for dend_id in range(0, dend_num):
    p = Point3DWithDiam(x=d.x, y=d.y, z=d.z, diameter=dend_diam)
    d = Point3DWithDiam(x=p.x, y=p.y + dend_len, z=p.z, diameter=dend_diam)
    dend = Segment(proximal=p, distal=d, name="Dend_%i" % dend_id, id=1 + dend_id)
    dend.parent = SegmentParent(segments=parent_seg.id)
    parent_seg = dend
    morphology.segments.append(dend)

morphology.id = "TestMorphology"

return morphology

def run():
    cell_num = 10
    x_size = 500
    y_size = 500
    z_size = 500

    nml_doc = NeuroMLDocument(id="Net3DExample")
    syn0 = ExpOneSynapse(id="syn0", gbase="65nS", erev="0mV", tau_decay="3ms")
    nml_doc.exp_one_synapses.append(syn0)

    net = Network(id="Net3D")
    nml_doc.networks.append(net)

    proj_count = 0
    # conn_count = 0

    for cell_id in range(0, cell_num):
        cell = Cell(id="Cell_%i" % cell_id)
        cell.morphology = generateRandomMorphology()
        nml_doc.cells.append(cell)

        pop = Population(id="Pop_%i" % cell_id, component=cell.id, type="populationList")
        net.populations.append(pop)
        pop.properties.append(Property(tag="color", value="1 0 0"))

        inst = Instance(id="0")
        pop.instances.append(inst)

        inst.location = Location(x=str(x_size * random()), y=str(y_size * random()), z=str(z_size * random()))
prob_connection = 0.5
for post in range(0, cell_num):
    if post is not cell_id and random() <= prob_connection:
        from_pop = "Pop_%i" % cell_id
        to_pop = "Pop_%i" % post
        pre_seg_id = 0
        post_seg_id = 1
        projection = Projection(
            id="Proj_%i" % proj_count,
            presynaptic_population=from_pop,
            postsynaptic_population=to_pop,
            synapse=syn0.id,
        )
        net.projections.append(projection)
        connection = Connection(
            id=proj_count,
            pre_cell_id="%s[%i]" % (from_pop, 0),
            pre_segment_id=pre_seg_id,
            pre_fraction_along=random(),
            post_cell_id="%s[%i]" % (to_pop, 0),
            post_segment_id=post_seg_id,
            post_fraction_along=random(),
        )
        projection.connections.append(connection)
        proj_count += 1

# net.synaptic_connections.append(SynapticConnection(from_="%s[%i]" % (from_pop,0),
# to="%s[%i](to_pop,0))

####### Write to file ######

nml_file = "tmp/net3d.nml"
writers.NeuroMLWriter.write(nml_doc, nml_file)

print("Written network file to: " + nml_file)

####### Validate the NeuroML ######

from neuroml.utils import validate_neuroml2
validate_neuroml2(nml_file)

run()
1.4.5 Ion channels

"""
Generating a Hodgkin-Huxley Ion Channel and writing it to NeuroML
"""

```python
import neuroml
import neuroml.writers as writers

chan = neuroml.IonChannelHH(
    id="na",
    conductance="10pS",
    species="na",
    notes="This is an example voltage-gated Na channel",
)

m_gate = neuroml.GateHHRates(id="m", instances="3")
h_gate = neuroml.GateHHRates(id="h", instances="1")

m_gate.forward_rate = neuroml.HHRate(
    type="HHExpRate", rate="0.07per_ms", midpoint="-65mV", scale="-20mV"
)

m_gate.reverse_rate = neuroml.HHRate(
    type="HHExpLinearRate", rate="1per_ms", midpoint="-55mV", scale="10mV"
)

h_gate.forward_rate = neuroml.HHRate(
    type="HHExpRate", rate="0.125per_ms", midpoint="-65mV", scale="-80mV"
)

chan.gate_hh_rates.append(m_gate)
chan.gate_hh_rates.append(h_gate)

doc = neuroml.NeuroMLDocument()
doc.ion_channel_hhs.append(chan)

doc.id = "ChannelMLDemo"

nml_file = "./tmp/ionChannelTest.xml"
writers.NeuroMLWriter.write(doc, nml_file)

print("Written channel file to: " + nml_file)

###### Validate the NeuroML ######

from neuroml.utils import validate_neuroml2
```

(continues on next page)
**1.4.6 PyNN models**

''''

Example to build a PyNN based network

''''

```python
from neuroml import NeuroMLDocument
from neuroml import *
import neuroml.writers as writers
from random import random

############################ Build the network ################################

nml_doc = NeuroMLDocument(id="IafNet")

pynn0 = IF_curr_alpha(
    id="IF_curr_alpha_pop_IF_curr_alpha",
    cm="1.0",
    i_offset="0.9",
    tau_m="20.0",
    tau_refrac="10.0",
    tau_syn_E="0.5",
    tau_syn_I="0.5",
    v_init="-65",
    v_reset="-62.0",
    v_rest="-65.0",
    v_thresh="-52.0",
)

nml_doc.IF_curr_alpha.append(pynn0)

pynn1 = HH_cond_exp(
    id="HH_cond_exp_pop_HH_cond_exp",
    cm="0.2",
    e_rev_E="0.0",
    e_rev_I="-80.0",
    e_rev_K="-90.0",
    e_rev_Na="50.0",
    e_rev_leak="-65.0",
    g_leak="0.01",
    gbar_K="6.0",
    gbar_Na="20.0",
    i_offset="0.2",
    tau_syn_E="0.2",
    tau_syn_I="2.0",
)

nml_doc.HH_cond_exp.append(pynn1)
```

(continues on next page)
v_init=-65,
  v_offset=-63.0,
)
nml_doc.HH_cond_exp.append(pynn1)

pynnSynn0 = ExpCondSynapse(id="ps1", tau_syn="5", e_rev="0")
nml_doc.exp_cond_synapses.append(pynnSynn0)

nml_file = "tmp/pynn_network.xml"
writers.NeuroMLWriter.write(nml_doc, nml_file)
print("Saved to: " + nml_file)

##### Validate the NeuroML ######

from neuroml.utils import validate_neuroml2
validate_neuroml2(nml_file)

### 1.4.7 Synapses

Example to create a file with multiple synapse types

```python
from neuroml import NeuroMLDocument
from neuroml import *
import neuroml.writers as writers
from random import random

nml_doc = NeuroMLDocument(id="SomeSynapses")

expOneSyn0 = ExpOneSynapse(id="ampa", tau_decay="5ms", gbase="1nS", erev="0mV")
nml_doc.exp_one_synapses.append(expOneSyn0)

expTwoSyn0 = ExpTwoSynapse(
    id="gaba", tau_decay="12ms", tau_rise="3ms", gbase="1nS", erev="-70mV"
)
nml_doc.exp_two_synapses.append(expTwoSyn0)

bpSyn = BlockingPlasticSynapse(
    id="blockStpSynDep", gbase="1nS", erev="0mV", tau_rise="0.1ms", tau_decay="2ms"
)
bpSyn.notes = "This is a note"
bpSyn.plasticity_mechanism = PlasticityMechanism(
    type="tsodyksMarkramDepMechanism", init_release_prob="0.5", tau_rec="120 ms"
)
```

(continues on next page)
bpSyn.block_mechanism = BlockMechanism(
    type="voltageConcDepBlockMechanism",
    species="mg",
    block_concentration="1.2 mM",
    scaling_conc="1.920544 mM",
    scaling_volt="16.129 mV",
)

nml_doc.blocking_plastic_synapses.append(bpSyn)

nml_file = "tmp/synapses.xml"
writers.NeuroMLWriter.write(nml_doc, nml_file)
print("Saved to: " + nml_file)

##### Validate the NeuroML #####

from neuroml.utils import validate_neuroml2
validate_neuroml2(nml_file)

1.4.8 Working with JSON serialization

One thing to note is that the JSONWriter, unlike NeuroMLWriter, will serializing using array-based (Arraymorph) representation if this has been used.

1.4.9 Working with arraymorphs

""
Example of connecting segments together to create a multicompartamental model of a cell.

In this case ArrayMorphology will be used rather than Morphology - demonstrating its similarity and ability to save in HDF5 format
""

import neuroml
import neuroml.writers as writers
import neuroml.arraymorph as am

p = neuroml.Point3DWithDiam(x=0, y=0, z=0, diameter=50)
d = neuroml.Point3DWithDiam(x=50, y=0, z=0, diameter=50)
soma = neuroml.Segment(proximal=p, distal=d)
soma.name = "Soma"
soma.id = 0

# now make an axon with 100 compartments:
parent = neuroml.SegmentParent(segments=soma.id)
parent_segment = soma
axon_segments = []
seg_id = 1
for i in range(100):
    p = neuroml.Point3DWithDiam(
        x=parent_segment.distal.x,
        y=parent_segment.distal.y,
        z=parent_segment.distal.z,
        diameter=0.1,
    )
    d = neuroml.Point3DWithDiam(
        x=parent_segment.distal.x + 10,
        y=parent_segment.distal.y,
        z=parent_segment.distal.z,
        diameter=0.1,
    )
    axon_segment = neuroml.Segment(proximal=p, distal=d, parent=parent)
    axon_segment.id = seg_id
    axon_segment.name = "axon_segment_" + str(axon_segment.id)

    # now reset everything:
    parent = neuroml.SegmentParent(segments=axon_segment.id)
    parent_segment = axon_segment
    seg_id += 1

    axon_segments.append(axon_segment)

test_morphology = am.ArrayMorphology()
test_morphology.segments.append(soma)
test_morphology.segments += axon_segments
test_morphology.id = "TestMorphology"

cell = neuroml.Cell()
cell.name = "TestCell"
cell.id = "TestCell"
cell.morphology = test_morphology

doc = neuroml.NeuroMLDocument()
# doc.name = "Test neuroML document"
doc.cells.append(cell)
doc.id = "TestNeuroMLDocument"

nml_file = "tmp/arraymorph.nml"
writers.NeuroMLWriter.write(doc, nml_file)
print("Written morphology file to: " + nml_file)

##### Validate the NeuroML ######

from neuroml.utils import validate_neuroml2
validate_neuroml2(nml_file)

## 1.4.10 Working with Izhikevich Cells

These examples were kindly contributed by Steve Marsh

```python
# from neuroml import NeuroMLDocument
from neuroml import IzhikevichCell
from neuroml.loaders import NeuroMLLoader
from neuroml.utils import validate_neuroml2

def load_izhikevich(filename="./test_files/SingleIzhikevich.nml"):
    nml_filename = filename
    validate_neuroml2(nml_filename)
    nml_doc = NeuroMLLoader.load(nml_filename)

    iz_cells = nml_doc.izhikevich_cells
    for i, iz in enumerate(iz_cells):
        if isinstance(iz, IzhikevichCell):
            neuron_string = "%d %s %s %s %s %s (\%s)" % (i,
                                                 iz.v0,
                                                 iz.a,
                                                 iz.b,
                                                 iz.c,
                                                 iz.d,
                                                 iz.id,
                                                 )
            print(neuron_string)
        else:
            print("Error: Cell %d is not an IzhikevichCell" % i)

load_izhikevich()
```

from neuroml import NeuroMLDocument
from neuroml import IzhikevichCell
from neuroml.loaders import NeuroMLLoader
from neuroml.writers import NeuroMLWriter
from neuroml.utils import validate_neuroml2
def write_izhikevich(filename="./tmp/SingleIzhikevich_test.nml"):
    nml_doc = NeuroMLDocument(id="SingleIzhikevich")
    nml_filename = filename

    iz0 = IzhikevichCell("iz0", v0="-70mV", thresh="30mV", a="0.02", b="0.2", c="-65.0", d="6"
    )

    nml_doc.izhikevich_cells.append(iz0)

    NeuroMLWriter.write(nml_doc, nml_filename)
    validate_neuroml2(nml_filename)

write_izhikevich()

1.5 References
2.1 How to contribute

libNeuroML development happens on GitHub, so you will need a GitHub account to contribute to the repository. Contributions are made using the standard Pull Request workflow.

2.1.1 Setting up

Please take a look at the GitHub documentation here: http://help.github.com/fork-a-repo/

To begin, please fork the repo on the GitHub website. You should now have a libNeuroML under you username. Next, we clone our fork to get a local copy on our computer:

```bash
git clone git@github.com:_username_/libNeuroML.git
```

While not necessary, it is good practice to add the upstream repository as a remote that you will follow:

```bash
cd libNeuroML
git remote add upstream https://github.com/NeuralEnsemble/libNeuroML.git
git fetch upstream
```

You can check which branch are you following doing:

```bash
git branch -a
```

You should have something like:

```
git branch -a
  * master
    remotes/origin/HEAD -> origin/master
    remotes/origin/master
    remotes/upstream/master
```
2.1.2 Sync with upstream

Before starting to do some work, please check to see that you have the latest copy of the sources in your local repository:

```
git fetch upstream
git checkout development
git merge upstream/development
```

2.1.3 Working locally on a dedicated branch

Now that we have a fork, we can start making our changes to the source code. The best way to do it is to create a branch with a descriptive name to indicate what you are working on. Generally, your will branch off from the upstream development branch, which will contain the latest code.

For example, just for the sake of this guide, I’m going to work on issue #2.

```
git checkout development
git checkout -b fix-2
```

We can work in this branch, and make as many commits as we need to:

```
# hack hack hack
git commit -am "some decent commit message here"
```

Once we have finished working, we can push the branch online to our fork:

```
git push origin fix-2
```

We can then open a pull-request to merge our fix-2 branch into upstream/development. If your code is not ready to be included, you can update the code on your branch and any more commits you add there will be added to the Pull Request. Members of the libNeuroML development team will then discuss your changes with you, perhaps suggest tweaks, and then merge it when ready.

2.1.4 Continuous integration

libNeuroML uses continuous integration (Wikipedia). Each commit to the master or development branches is tested, along with all commits to pull requests. The latest status of the continuous integration tests can be seen here on GitHub Actions.

2.1.5 Release process

libNeuroML is part of the official NeuroML release cycle. When a new libNeuroML release is ready the following needs to happen:

- Update version number in setup.py
- update version number in doc/conf.py
- update release number in doc/conf.py (same as version number)
- update changelog in README.md
- merge development branch with master (This should happen via pull request - do not do the merge yourself even if you are an owner of the repository.)
• push latest release to PyPi

More information on the NeuroML release process can be found on the NeuroML documentation page.

### 2.2 Regenerating documentation

Please create a virtual environment and use the `requirements.txt` file to install the necessary bits.

In most cases, running `make html` should be sufficient to regenerate the documentation. However, if any changes to `nml.py` have been made, the `nml-core-docs.py` file in the `helpers` directory will also need to be run. This script manually adds each class from `nml.py` to the documentation as a sub-section using the `autoclass` sphinx directive instead of the `automodule` directive which does not allow us to do this.

### 2.3 Implementation of XML bindings for libNeuroML

The GenerateDS Python package is used to automatically generate the NeuroML XML-bindings in libNeuroML from the NeuroML Schema. This technique can be utilized for any XML Schema and is outlined in this section. The addition of helper methods and enforcement of correct naming conventions is also described. For more detail on how Python bindings for XML are generated, the reader is directed to the GenerateDS and libNeuroML documentation. In the following subsections it is assumed that all commands are executed in a top level directory `nml` and that GenerateDS is installed. It should be noted that enforcement of naming conventions and addition of helper methods are not required by GenerateDS and default values may be used.

#### 2.3.1 Correct naming conventions

A module named `generateds_config.py` is placed in the `nml` directory. This module contains a Python dictionary called `NameTable` which maps the original names specified in the XML Schema to user-specified ones. The `NameTable` dictionary can be defined explicitly or generated programmatically, for example using regular expressions.

#### 2.3.2 Addition of helper methods

Helper methods associated with a class can be added to a Python module as string objects. In the case of libNeuroML the module is called `helper_methods.py`. The precise implementation details are esoteric and the user is referred to the GenerateDS documentation for details of how this functionality is implemented.

#### 2.3.3 Generation of bindings

Once `generateds_config.py` and a helper methods module are present in the `nml` directory a valid XML Schema is required by GenerateDS. The following command generates the `nml.py` module which contains the XML-bindings:

```
$ generateDS.py -o nml.py --use-getter-setter=none --user-methods=helper_methods NeuroML_v2beta1.xsd
```

The `-o` flag sets the file which the module containing the bindings is to be written to. The `--use-getter-setter=none` option disables getters and setters for class attributes. The `--user-methods` flag indicates the name of the helper methods module (See section “Addition of helper methods”). The final parameter (`NeuroML_v2beta1.xsd`) is the name of the XML Schema used for generating the bindings.
2.4 Multicompartmental Python API Meeting

2.4.1 Organisation

Dates: 25 & 26 June 2012
Location: Room 336, Rockefeller building, UCL, London
Attendees: Sandra Berger, Andrew Davison, Padraig Gleeson, Mike Hull, Steve Marsh, Michele Mattioni, Eugenio Piasini, Mike Vella
Sponsors: This meeting was generously supported by the INCF Multi Scale Modelling Program.

2.4.2 Minutes

Agreeing on terminology (segments, etc.) & scope

A discussion on the definitions of the key terms Node, Segment and Section is here, and was the basis for discussions on these definitions at the meeting:

*Nodes, Segments and Sections*

**Agreements**

The Python libNeuroML API will use Node as a key building block for morphologies.

Segment is agreed on as the basis for defining morphologies in NeuroML and will be a top level object in libNeuroML, where it will be the part of a neurite between two Nodes (proximal & distal).

Segment Group will be the basis for the grouping of these, and will be used to define dendrites, axons, etc.

Section is a term for the cable-like building block in NEURON, and will not be formally used in NeuroML or libNeuroML.

There was a discussion on whether it would be useful to be able to include this concept “by the back door” to enable lossless import & export of morphologies from NEURON. Padraig’s proposal was to add an attribute (e.g. primary) to the segmentGroup element to flag a core set of non overlapping segmentGroups, which are continuous (all children are connected to distal point of parent) which would correspond to the old “cable” concept in NeuroML v1.x.

There was much discussion on the usefulness of this concept and whether it should be a different element/object in the API from segmentGroup. The outcome was not fully resolved, but as a first test of this concept, Padraig will add the new attribute to NeuroML, Mike V will add a flag (boolean?) to the API, and at a later point, when the API begins to interact with native simulators, we can reevaluate the usefulness of the term.

**Mike Vella’s current implementation**

This is under development at: https://github.com/NeuralEnsemble/libNeuroML/tree/master/neuroml

Mike will continue on this (almost) full time for the next 2 months.

Following the meeting, he will perform a refactoring operation on the code base to better reflect the names used in NeuroML, e.g.

```python
    neuroml_doc
cells
    morphology # not entirely sure how this works- contains segment groups and is itself a segment group?
```
It was also decided that certain SegmentGroup names should have reserved names in libNeuroML, the exact implementation of this is undecided:

**Segment groups with reserved names:**

- soma_group
- axon_group
- apical_dendrite_group
- basal_dendrite_group

It was also decided that a segment should only be able to connect to the root of a morphology, the syntax should be something along the lines of:

**Segment can only connect to root of a morphology**

**Connect syntax examples:**

```python
morph2.attach(2,cell2,0.5) (default frac along = None)
```

and:

```python
morph[2].attach(cell2,0.5)
```

Mike V was asked to add a clone method to a morphology.

It was decided that fraction_along should be a property of segment.

The syntax for segment groups should be as follows: `group=morph.segment_groups['axon_group']` (in connect merge groups should be false by default - throw an exception, tell the user setting merge_groups = True or rename group will fix this)

This was a subject of great debate and has not been completely settled.
**Morphforge latest developments**

Mike Hull gave a brief overview of the latest developments with Morphforge:

https://github.com/mikehulluk/morphforge

He pointed out that it’s still undergoing refactoring, but it can be used by other interested parties, and there is detailed documentation online regarding installation, examples, etc.

**Neuronvisio latest developments**

Michele Mattioni gave a status update on Neuronvisio:

http://neuronvisio.org

The application has been closely linked to the NEURON simulator but hopefully use of libNeuroML will allow it to be used independently of NEURON.

Michele showed Neuronvisio’s native HDF5 format as just one possible way to encode model structure + simulation results: https://github.com/NeuralEnsemble/libNeuroML/blob/master/hdf5Examples/Neuronvisio_medium_cell_example_10ms.h5

**Current Python & NeuroML support in MOOSE**

A Skype call/Google Hangout was held on Tues at 9:30 to get an update from Bangalore.

The slides from this discussion are here:


As outlined there are a number of areas in which MOOSE and Moogli import/export NeuroML version 1.x. A number of issues and desired features missing in v1.x were highlighted, most of which are implemented or planned for NeuroML v2.0.

There was general enthusiasm about the libNeuroML project, and it was felt that MOOSE should eventually transition to using libNeuroML to import NeuroML models. This will happen in parallel with updating of the MOOSE PyNN implementation.

The MOOSE developers were also keen to see how the new ComponentTypes in NeuroML 2 will map to inbuilt objects in MOOSE (e.g. Integrate-and-Fire neurons, Markov channel, Izhikevich). They will add simple examples to the latest MOOSE code to demonstrate their current implementation and discussion can continue on the mailing lists.

**Saving to & loading from XML**

There was not any detailed discussion on the various strategies for reading/saving XML in Python.

Padraig’s suggestion based on generateDS.py: https://github.com/NeuralEnsemble/libNeuroML/tree/master/ideas/padraig/generatedFromV2Schema produces a very big file, which while usable as an API, e.g. see:

https://github.com/NeuralEnsemble/libNeuroML/blob/master/hhExample/hh_NEUROML2.py

could do a lot of refactoring. It was felt that a version of this with a very efficient description of morphologies (and network instances) based on the current work of Mike V is the way forward.
Storing simulation data as HDF5

The examples at: https://github.com/NeuralEnsemble/libNeuroML/tree/master/hdf5Examples have been updated.

The long term aim would be to arrive at a common format here that can be saved by simulators and that visualisation packages like Moogli and Neuronvisio can read and display. This may be based on Neo: http://packages.python.org/neo/, but that package’s current lack of ability to deal with data with nonuniform time points (e.g. produced by variable time step simulations) may be a limiting factor.

General PyNN & NeuroML v2.0 interoperability

There was agreement that libNeuroML will form the basis of the multicompartmental neuron support in PyNN. The extra functionality needed to interact with simulators is currently termed “Pyramidal”, but this will eventually be fully merged into PyNN.


2.5 Nodes, Segments and Sections

An attempt to clarify these interrelated terms used in describing morphologies. Names in **bold type** are used for elements of the NeuroML object model.

2.5.1 Nodes

A node is a 3D point with diameter information which forms the basis for 3D morphological reconstructions.

These nodes (or points) are the fundamental building blocks in the SWC and Neurolucida formats. This method of description is based on the assumption that each node is physically connected to another node.

2.5.2 Segments

A **segment** (according to NeuroML v1&2) is a part of a neuronal tree between two 3D points with diameters (**proximal** & **distal**). The term node isn’t used in NeuroML but the above description describes perfectly well the **proximal** & **distal** points. Cell **morphology** elements consist of lists of **segments** (each with unique integer id, and optional name).

All segments, apart from the root segment, have a **parent** segment. If the **proximal** point of the segment is not specified, the **distal** point of the parent segment is used for the **proximal** point of the child.

A special case is defined where **proximal == distal**, and the **segment** is assumed to be a sphere at that location with the specified diameter.

Segments can be grouped into **segmentGroups** in NeuroML v2.0. These can be used to specify “apical_dendrites”, “axon_group”, etc., which in turn can be used for placing channels on the cell.

An example of a NeuroML v2.0 cell is here.

libNeuroML will allow low level access to create and modify morphologies by handling nodes. Segments will also be top level objects in the API. The XML serialisation will only specify **segments** with **proximal** & **distal** points, but the HDF5 version may have an efficient serialisation of nodes & segments.
2.5.3 Sections

The concept of section is fundamentally important in NEURON. A section in this simulator is an unbranched cable which can have multiple 3D points outlining the structure of a neurite in 3D. These points are used to determine the surface area along the section. NEURON can vary the spatial discretisation of the neurite by varying the “nseg” value of the section, e.g. a section with 20 3D points and nseg =4 will be split into 4 parts of equal length for simulating (as isopotential compartments), with the surface area (and so total channel conductance) of each determined by the set of 3D points in that part.

There was a similar concept to this in NeuroML v1.x, the cable. Each segment had an attribute for the cable id, and these were used for mapping to and from NEURON. Cables were unbranched, and so all segments after the first in the cable only had distal points, see this example.

The cable concept was removed in NeuroML v2.0, as this is was seen as imposing concepts from compartmental modelling on the basic morphological descriptions of cells. There is only a segmentGroup element for grouping segments, though a segment can belong to multiple segmentGroups, which don’t need to be unbranched (unlike cables). There may need to be a new attribute in segmentGroup (e.g. primary or unbranched or cable=”true”) which defines a nonoverlapping set of unbranched segmentGroups, which can be used as the basis for sections in any parsing application which is interested in them, or be ignored by any other application.

In libNeuroML, a section-like concept can be added at API level, to facilitate building cells, to facilitate import/export to/from simulators supporting this concept, and to serve as a basis for recompartmentalisation of cells.

2.5.4 Issues

Dendrites in space

One major issue to address is that in many neuronal reconstructions, the soma is not included (or perhaps just an outline of the soma is given), only the dendrites are. These dendrites’ 3D start points are on the edge of the soma membrane “floating in space”. Normal procedure for a modeller in this case is to create a spherical soma at this central point and electrically attach the dendrites to the centre of this.

In this case (and many others) the physical location of the start of the child segments do not correspond to the electrical (or logical) connection point on the parent. This has advantages and disadvantages:

(+ ) It allows the real 3D points of the neuronal reconstruction to be retained (useful for visualisation)
(- ) This is not unambiguously captured in the simplest morphological formats like SWC, which assume physical connectivity between nodes/points

This scenario is supported in NeuroML v1&2, where a child segment has the option to redefine its start point (by adding a proximal) with the child <-> parent relationship defining the electrical connection. This allows lossless import & export from NEURON and removes the ambiguity of more compact formats like SWC and Neurolucida.

Connections mid segment

Another option for electrical connections (also influences by NEURON sections) is the ability for segments to (electrically/logically) connect to a point inside a segment. This is specified by adding a fractionAlong attribute to the parent element, i.e.

```
<parent segment="2" fractionAlong="0.5"/>
```

This is not possible in a node based format, but represents a logically consistent description of what the modeller wants.
What to do?

Two options are available then for a serialisation format or API: should it try to support all of these scenarios, or try to enforce “best practice”?

PG: I’d argue for the first approach, as it retains as much as possible of what the original reconstructor/simulator specified. An API which enforces a policy when it encounters a non optimal morphology (e.g. moving all dendrites to connection points, inserting new nodes) will alter the original data in perhaps unintended ways, and that information will be lost by subsequent readers. It should be up to each parsing application to decide what to do with the extra information when it reads in a file.
CHAPTER
THREE

INDICES AND TABLES

• genindex
• modindex
• search
neuroml.loaders, 70
neuroml.utils, 71
neuroml.writers, 71
INDEX

A
add() (neuroml.nml.nml.BaseWithoutId method), 11
add_all_to_document() (in module neuroml.utils), 71
AdExIaFCell (class in neuroml.nml.nml), 6
AlphaCondSynapse (class in neuroml.nml.nml), 6
AlphaCurrentSynapse (class in neuroml.nml.nml), 7
AlphaCurrSynapse (class in neuroml.nml.nml), 7
AlphaSynapse (class in neuroml.nml.nml), 7
Annotation (class in neuroml.nml.nml), 8
append() (neuroml.nml.nml.NeuroMLDocument method), 52
append_to_element() (in module neuroml.utils), 71
ArrayMorphLoader (class in neuroml.loaders), 70
ArrayMorphWriter (class in neuroml.writers), 71

B
Base (class in neuroml.nml.nml), 8
BaseCell (class in neuroml.nml.nml), 8
BaseCellMembPotCap (class in neuroml.nml.nml), 8
BaseConductanceBasedSynapse (class in neuroml.nml.nml), 8
BaseConductanceBasedSynapseTwo (class in neuroml.nml.nml), 9
BaseConnection (class in neuroml.nml.nml), 9
BaseConnectionNewFormat (class in neuroml.nml.nml), 9
BaseConnectionOldFormat (class in neuroml.nml.nml), 10
BaseCurrentBasedSynapse (class in neuroml.nml.nml), 10
BaseNonNegativeIntegerId (class in neuroml.nml.nml), 10
BaseProjection (class in neuroml.nml.nml), 10
basePyNNCell (class in neuroml.nml.nml), 68
basePyNNIaFCell (class in neuroml.nml.nml), 69
basePyNNIaFCondCell (class in neuroml.nml.nml), 69
BasePyNNSynapse (class in neuroml.nml.nml), 10
BaseSynapse (class in neuroml.nml.nml), 11
BaseVoltageDepSynapse (class in neuroml.nml.nml), 11
BaseWithoutId (class in neuroml.nml.nml), 11

BiophysicalProperties (class in neuroml.nml.nml), 12
BiophysicalProperties2CaPools (class in neuroml.nml.nml), 12
BlockingPlasticSynapse (class in neuroml.nml.nml), 13
BlockMechanism (class in neuroml.nml.nml), 13

C
Case (class in neuroml.nml.nml), 13
Cell (class in neuroml.nml.nml), 13
Cell2CaPools (class in neuroml.nml.nml), 16
CellSet (class in neuroml.nml.nml), 16
ChannelDensity (class in neuroml.nml.nml), 16
ChannelDensityGHK (class in neuroml.nml.nml), 17
ChannelDensityGHK2 (class in neuroml.nml.nml), 17
ChannelDensityNernst (class in neuroml.nml.nml), 17
ChannelDensityNernstCa2 (class in neuroml.nml.nml), 17
ChannelDensityNonUniform (class in neuroml.nml.nml), 18
ChannelDensityNonUniformGHK (class in neuroml.nml.nml), 18
ChannelDensityNonUniformNernst (class in neuroml.nml.nml), 18
ChannelDensityNonUniformNernstCa2 (class in neuroml.nml.nml), 18
ChannelDensityVShift (class in neuroml.nml.nml), 19
ChannelPopulation (class in neuroml.nml.nml), 19
ClosedState (class in neuroml.nml.nml), 19
ComponentType (class in neuroml.nml.nml), 20
CompoundInput (class in neuroml.nml.nml), 20
CompoundInputDL (class in neuroml.nml.nml), 20
ConcentrationModel_D (class in neuroml.nml.nml), 20
ConditionalDerivedVariable (class in neuroml.nml.nml), 21
Connection (class in neuroml.nml.nml), 21
ConnectionWD (class in neuroml.nml.nml), 22
Constant (class in neuroml.nml.nml), 23
ContinuousConnection (class in neuroml.nml.nml), 23
ContinuousConnectionInstance (class in neuroml.nml.nml), 24
ContinuousConnectionInstanceW (class in neuroml.nml.nml), 25
ContinuousProjection (class in neuroml.nml), 25
DecayingPoolConcentrationModel (class in neuroml.nml), 25
DerivedVariable (class in neuroml.nml), 26
DistalDetails (class in neuroml.nml), 26
distance_to() (neuroml.nml.Point3DWithDiam method), 55
DoubleSynapse (class in neuroml.nml), 26
Dynamics (class in neuroml.nml), 26

E
EIF_cond_alpha_isfa_ista (class in neuroml.nml), 27
EIF_cond_exp_isfa_ista (class in neuroml.nml), 28
ElectricalConnection (class in neuroml.nml), 29
ElectricalConnectionInstance (class in neuroml.nml), 30
ElectricalConnectionInstanceW (class in neuroml.nml), 30
ElectricalProjection (class in neuroml.nml), 30
ExpCondSynapse (class in neuroml.nml), 31
ExpCurrSynapse (class in neuroml.nml), 31
ExplicitInput (class in neuroml.nml), 32
ExpOneSynapse (class in neuroml.nml), 31
exportHdf5() (neuroml.nml.ContinuousProjection method), 25

eexportHdf5() (neuroml.nml.ElectricalProjection method), 30
exportHdf5() (neuroml.nml.InputList method), 45
exportHdf5() (neuroml.nml.Population method), 55
exportHdf5() (neuroml.nml.Projection method), 56
Exposure (class in neuroml.nml), 33
ExpThreeSynapse (class in neuroml.nml), 32
ExpTwoSynapse (class in neuroml.nml), 32
ExtracellularProperties (class in neuroml.nml), 33
ExtracellularPropertiesLocal (class in neuroml.nml), 33

F
F FitzHughNagumo1969Cell (class in neuroml.nml), 33
F FitzHughNagumoCell (class in neuroml.nml), 34
FixedFactorConcentrationModel (class in neuroml.nml), 34
ForwardTransition (class in neuroml.nml), 35
G
GapJunction (class in neuroml.nml), 35

GateFractional (class in neuroml.nml), 35
GateFractionalSubgate (class in neuroml.nml), 35
GateHHInstantaneous (class in neuroml.nml), 36
GateHHRates (class in neuroml.nml), 36
GateHHRatesInf (class in neuroml.nml), 36
GateHHRatesTau (class in neuroml.nml), 36
GateHHRatesTauInf (class in neuroml.nml), 36
GateHHTauInf (class in neuroml.nml), 37
GateHHUndetermined (class in neuroml.nml), 37
GateKS (class in neuroml.nml), 37
get_actual_proximal() (neuroml.nml.Cell method), 13
get_all_segments_in_group() (neuroml.nml.Cell method), 14
get_by_id() (neuroml.nml.Network method), 51
get_by_id() (neuroml.nml.NeuroMLDocument method), 52
get_delay_in_ms() (neuroml.nml.Connection method), 22
get_fraction_along() (neuroml.nml.ExplicitInput method), 32
get_fraction_along() (neuroml.nml.Input method), 44
get_members() (neuroml.nml.BaseWithoutId method), 11
get_ordered_segments_in_groups() (neuroml.nml.Cell method), 14
get_post_cell_id() (neuroml.nml.Connection method), 21
get_post_cell_id() (neuroml.nml.Connection method), 22
get_post_cell_id() (neuroml.nml.ContinuousConnection method), 23
get_post_cell_id() (neuroml.nml.ElectricalConnection method), 29
get_post_fraction_along() (neuroml.nml.Connection method), 21
get_post_fraction_along() (neuroml.nml.Connection method), 22
get_post_fraction_along() (neuroml.nml.ContinuousConnection method), 23
get_post_fraction_along() (neuroml.nml.ElectricalConnection method), 29
get_post_info() (neuroml.nml.Connection method), 21
get_post_info() (neuroml.nml.Connection method), 22
get_post_info() (neuroml.nml.ContinuousConnection method),
get_post_info() (neuroml.nml.nml.ElectricalConnection method), 29
get_post_segment_id() (neuroml.nml.nml.Connection method), 21
get_post_segment_id() (neuroml.nml.nml.ConnectionWD method), 22
get_post_segment_id() (neuroml.nml.nml.ContinuousConnection method), 24
get_post_segment_id() (neuroml.nml.nml.ElectricalConnection method), 29
get_pre_cell_id() (neuroml.nml.nml.Connection method), 21
get_pre_cell_id() (neuroml.nml.nml.ConnectionWD method), 23
get_pre_cell_id() (neuroml.nml.nml.ContinuousConnection method), 24
get_pre_cell_id() (neuroml.nml.nml.ElectricalConnection method), 29
get_pre_fraction_along() (neuroml.nml.nml.Connection method), 21
get_pre_fraction_along() (neuroml.nml.nml.ConnectionWD method), 23
get_pre_fraction_along() (neuroml.nml.nml.ContinuousConnection method), 24
get_pre_fraction_along() (neuroml.nml.nml.ElectricalConnection method), 29
get_pre_info() (neuroml.nml.nml.Connection method), 21
get_pre_info() (neuroml.nml.nml.ConnectionWD method), 23
get_pre_info() (neuroml.nml.nml.ContinuousConnection method), 24
get_pre_info() (neuroml.nml.nml.ElectricalConnection method), 29
get_pre_segment_id() (neuroml.nml.nml.Connection method), 22
get_pre_segment_id() (neuroml.nml.nml.ConnectionWD method), 23
get_pre_segment_id() (neuroml.nml.nml.ContinuousConnection method), 24
get_pre_segment_id() (neuroml.nml.nml.ElectricalConnection method), 29
get_segment() (neuroml.nml.nml.Cell method), 14
get_segment_group() (neuroml.nml.nml.Cell method), 14
get_segment_groups_by_substring() (neuroml.nml.nml.Cell method), 15
get_segment_id() (neuroml.nml.nml.ExplicitInput method), 33
get_segment_id() (neuroml.nml.nml.Input method), 44
get_segment_ids_vs_segments() (neuroml.nml.nml.Cell method), 15
get_segment_length() (neuroml.nml.nml.Cell method), 15
get_segment_surface_area() (neuroml.nml.nml.Cell method), 15
get_segment_volume() (neuroml.nml.nml.Cell method), 15
get_segments_by_substring() (neuroml.nml.nml.Cell method), 15
get_size() (neuroml.nml.nml.Population method), 55
get_summary() (in module neuroml.utils), 71
get_target_cell_id() (neuroml.nml.nml.ExplicitInput method), 33
get_target_cell_id() (neuroml.nml.nml.Input method), 45
get_target_population() (neuroml.nml.nml.ExplicitInput method), 33
get_weight() (neuroml.nml.nml.ContinuousConnectionInstanceW method), 25
get_weight() (neuroml.nml.nml.ElectricalConnectionInstanceW method), 30
get_weight() (neuroml.nml.nml.InputW method), 45
GradedSynapse (class in neuroml.nml.nml), 38
GridLayout (class in neuroml.nml.nml), 38

H

has_segment_fraction_info() (in module neuroml.utils), 72
HH_cond_exp (class in neuroml.nml.nml), 39
HHRate (class in neuroml.nml.nml), 38
HHTime (class in neuroml.nml.nml), 38
HHVariable (class in neuroml.nml.nml), 38

I

IafCell (class in neuroml.nml.nml), 42
IafRefCell (class in neuroml.nml.nml), 42
IafTauCell (class in neuroml.nml.nml), 43
IafTauRefCell (class in neuroml.nml.nml), 43
IF_cond_alpha (class in neuroml.nml.nml), 39
IF_cond_exp (class in neuroml.nml.nml), 40
IF_curr_alpha (class in neuroml.nml.nml), 41
IF_curr_exp (class in neuroml.nml.nml), 41
Include (class in neuroml.nml.nml), 43
IncludeType (class in neuroml.nml.nml), 44
info() (neuroml.nml.nml.Base WithoutId method), 12
InhomogeneousParameter (class in neuroml.nml.nml), 44
InhomogeneousValue (class in neuroml.nml.nml), 44
InitMembPotential (class in neuroml.nml.nml), 44
Input (class in neuroml.nml.nml), 44
InputList (class in neuroml.nml.nml), 45
InputW (class in neuroml.nml.nml), 45
Instance (class in neuroml.nml.nml), 45
InstanceRequirement (class in neuroml.nml.nml), 45
IntracellularProperties (class in neuroml.nml.nml), 46
IntracellularProperties2CaPools (class in neuroml.nml.nml), 46
IonChannel (class in neuroml.nml.nml), 46
IonChannelHH (class in neuroml.nml.nml), 46
IonChannelIKS (class in neuroml.nml.nml), 47
IonChannelScalable (class in neuroml.nml.nml), 47
IonChannelVShift (class in neuroml.nml.nml), 47
is_valid_neuroml2() (in module neuroml.utils), 72
Izhikevich2007Cell (class in neuroml.nml.nml), 48
IzhikevichCell (class in neuroml.nml.nml), 48

L
Layout (class in neuroml.nml.nml), 49
LEMS_Property (class in neuroml.nml.nml), 49
length (neuroml.nml.nml.Segment property), 60
LinearGradedSynapse (class in neuroml.nml.nml), 49
load() (neuroml.loaders.ArrayMorphLoader class method), 70
load() (neuroml.loaders.NeuroMLHdf5Loader class method), 70
load() (neuroml.loaders.NeuroMLLoader class method), 70
load_swc_single() (neuroml.loaders.SWCLoader class method), 70
Location (class in neuroml.nml.nml), 49

M
main() (in module neuroml.utils), 72
Member (class in neuroml.nml.nml), 49
MembraneProperties (class in neuroml.nml.nml), 50
MembraneProperties2CaPools (class in neuroml.nml.nml), 50
module
euroml.loaders, 70
euroml.utils, 71
euroml.writers, 71
Morphology (class in neuroml.nml.nml), 50

N
NamedDimensionalType (class in neuroml.nml.nml), 51
NamedDimensionalVariable (class in neuroml.nml.nml), 51
Network (class in neuroml.nml.nml), 51
neuroml.loaders module, 70
neuroml.utils module, 71
neuroml.writers module, 71
NeuroMLDocument (class in neuroml.nml.nml), 52
NeuroMLHdf5Loader (class in neuroml.loaders), 70
NeuroMLHdf5Writer (class in neuroml.writers), 71
NeuroMLLoader (class in neuroml.loaders), 70
NeuroMLWriter (class in neuroml.writers), 71
num_segments (neuroml.nml.Morphology property), 50

O
OpenState (class in neuroml.nml.nml), 53

P
Parameter (class in neuroml.nml.nml), 53
Path (class in neuroml.nml.nml), 53
PinskyRinzelCA3Cell (class in neuroml.nml.nml), 53
PlasticityMechanism (class in neuroml.nml.nml), 54
Point3DWithDiam (class in neuroml.nml.nml), 54
PoissonFiringSynapse (class in neuroml.nml.nml), 55
Population (class in neuroml.nml.nml), 55
print() (in module neuroml.loaders), 70
print_summary() (in module neuroml.utils), 72
Projection (class in neuroml.nml.nml), 56
Property (class in neuroml.nml.nml), 56
ProximalDetails (class in neuroml.nml.nml), 56
PulseGenerator (class in neuroml.nml.nml), 56
PulseGeneratorDL (class in neuroml.nml.nml), 57

Q
Q10ConductanceScaling (class in neuroml.nml.nml), 57
Q10Settings (class in neuroml.nml.nml), 57

R
RampGenerator (class in neuroml.nml.nml), 57
RampGeneratorDL (class in neuroml.nml.nml), 58
RandomLayout (class in neuroml.nml.nml), 58
ReactionScheme (class in neuroml.nml.nml), 58
read_neuroml2_file() (in module neuroml.loaders), 70
read_neuroml2_string() (in module neuroml.loaders), 70
Region (class in neuroml.nml.nml), 59
Requirement (class in neuroml.nml.nml), 59
Resistivity (class in neuroml.nml.nml), 59
ReverseTransition (class in neuroml.nml.nml), 59

S
Segment (class in neuroml.nml.nml), 59
SegmentEndPoint (class in neuroml.nml.nml), 60
SegmentGroup (class in neuroml.nml.nml), 60
SegmentParent (class in neuroml.nml.nml), 61
SilentSynapse (class in neuroml.nml.nml), 61
SineGenerator (class in neuroml.nml.nml), 61
SineGeneratorDL (class in neuroml.nml.nml), 61
Space (class in neuroml.nml.nml), 62
SpaceStructure (class in neuroml.nml.nml), 62
SpecificCapacitance (class in neuroml.nml.nml), 62
Spike (class in neuroml.nml.nml), 63
SpikeArray (class in neuroml.nml.nml), 63
SpikeGenerator (class in neuroml.nml.nml), 63
SpikeGeneratorPoisson (class in neuroml.nml.nml), 63
SpikeGeneratorRandom (class in neuroml.nml.nml), 64
SpikeGeneratorRefPoisson (class in neuroml.nml.nml), 64
SpikeSourcePoisson (class in neuroml.nml.nml), 64
SpikeThresh (class in neuroml.nml.nml), 65
Standalone (class in neuroml.nml.nml), 65
StateVariable (class in neuroml.nml.nml), 65
SubTree (class in neuroml.nml.nml), 65
summary() (neuroml.nml.nml.Cell method), 16
summary() (neuroml.nml.nml.NeuroMLDocument method), 52
surface_area (neuroml.nml.nml.Segment property), 60
SWCLoader (class in neuroml.loaders), 70
SynapticConnection (class in neuroml.nml.nml), 65

T
TauInfTransition (class in neuroml.nml.nml), 66
TimeDerivative (class in neuroml.nml.nml), 66
TimedSynapticInput (class in neuroml.nml.nml), 66
TransientPoissonFiringSynapse (class in neuroml.nml.nml), 66

U
UnstructuredLayout (class in neuroml.nml.nml), 67

V
validate_neuroml2() (in module neuroml.utils), 72
validate_Nml2Quantity_resistivity() (neuroml.nml.nml.Resistivity method), 59
validate_Nml2Quantity_resistivity_patterns_ (neuroml.nml.nml.Resistivity attribute), 59
VariableParameter (class in neuroml.nml.nml), 67
VoltageClamp (class in neuroml.nml.nml), 67
VoltageClampTriple (class in neuroml.nml.nml), 68
volume (neuroml.nml.nml.Segment property), 60

W
write() (neuroml.writers.ArrayMorphWriter class method), 71
write() (neuroml.writers.NeuroMLHdf5Writer class method), 71
write() (neuroml.writers.NeuroMLWriter class method), 71

Index