libNeuroML Documentation

Release 0.2.45

libNeuroML authors and contributors

Oct 01, 2018

Contents

CHAPTER 1

Introduction

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

NOTE: libNeuroML targets NeuroML v2.0 (described in Cannon et al, 2014) not NeuroML v1.8.1 (Gleeson et al. 2010).

For a detailed description of libNeuroML see:

Michael Vella, Robert C. Cannon, Sharon Crook, Andrew P. Davison, Gautham Ganapathy, Hugh P. C. Robinson, R. Angus Silver and Padraig Gleeson
libNeuroML and PyLEMS: using Python to combine procedural and declarative modeling approaches in computational neuroscience
Frontiers in Neuroinformatics 2014, doi: 10.3389/fninf.2014.00038

PLEASE CITE THE PAPER ABOVE IF YOU USE libNeuroML!

1.1 NeuroML

NeuroML provides an object model for describing neuronal morphologies, ion channels, synapses and 3D network structure.

Any dynamical components (channels, synapses, abstract cell models) in NeuroML v2.0 will have a definition "behind the scenes" in LEMS. However, all NeuroML files specify is that "element segment will contain element distal with attributes x, y, z, diameter…" or "element izhikevichCell will have attributes a, b, c…".

For more on NeuroML 2 and LEMS see here.

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). Note: LEMS definitions of NeuroML ComponentTypes (defining what izhikevichCell does with a, b, c...) and this XSD file (only saying the izhikevichCell element requires a, b, c...) are currently manually kept in line.

Other serialisations have been developed (HDF5, JSON, SWC). See Vella et al. 2014 for more details.

CHAPTER 2

Installation

2.1 Requirements

For the default XML serialization (saving NeuroML to XML files), only *lxml* is required:

```
sudo pip install lxml
```

Alternatively, on Linux you can use:

```
sudo apt-get install python-lxml
```

To use the other serializations (e.g. HDF5, JSON, see Vella et al. 2014) the following will also be required:

```
sudo apt-get install libhdf5-serial-dev
sudo pip install numpy
sudo pip install numexpr
sudo pip install jsonpickle
sudo pip install pymongo
sudo pip install simplejson
sudo pip install tables
```

See .travis.yml for the latest requirements on libraries etc.

2.2 Install libNeuroML via pip

```
pip install libNeuroML
```

This is always the latest stable branch from GitHub.

2.3 Install using a local copy of libNeuroML source

Install git and type:

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

More details about the git repository and making your own branch/fork are here.

Use the standard install method for Python packages:

```
sudo python setup.py install
```

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

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

2.4 Run an example

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

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

The standard examples can also be found here

2.5 Unit tests

To run unit tests cd to the directory '/neuoroml/test' and use the python unittest module discover method:

```
python -m unittest discover
```

If everything worked your output should look something like this:

```
.......
----------------------------------------------------------------------
Ran 55 tests in 40.1s

OK
```

Alternatively install and use nosetests:

```
nosetests -v
```

CHAPTER 3

Examples

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

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

3.1 Creating a NeuroML morphology

```
"""
Example of connecting segments together to create a
```

(continues on next page)

(continued from previous page)

```
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)

    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")
```

(continues on next page)

(continued from previous page)

```
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)
```

3.2 Loading and modifying a file

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

(continues on next page)

(continued from previous page)

```
    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 ######

from neuroml.utils import validate_neuroml2

validate_neuroml2(nml_file)
```

3.3 Building a network

```
"""

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

"""

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")
```

(continues on next page)

(continued from previous page)

```
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")

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),
```

(continues on next page)

(continued from previous page)

```
                                        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 ######

from neuroml.utils import validate_neuroml2

validate_neuroml2(nml_file)
```

3.4 Building a 3D network

```
"""

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

"""

from neuroml import NeuroMLDocument
from neuroml import Network
from neuroml import ExpOneSynapse
from neuroml import Population
from neuroml import Annotation
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()
```

(continues on next page)

(continued from previous page)

```
    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

    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)
```

(continues on next page)

(continued from previous page)

```
        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()
```

3.5 Ion channels

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

import neuroml
```

(continues on next page)

(continued from previous page)

```
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="HHSigmoidRate",
                                     rate="1per_ms",
                                     midpoint="-35mV",
                                     scale="10mV")

h_gate.forward_rate = neuroml.HHRate(type="HHExpLinearRate",
                                     rate="0.1per_ms",
                                     midpoint="-55mV",
                                     scale="10mV")

h_gate.reverse_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

validate_neuroml2(nml_file)
```

3.6 PyNN models

```
"""
```

(continues on next page)

(continued from previous page)

```
Example to build a PyNN based network

"""

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", 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)
```

3.7 Synapses

```
"""

Example to create a file with multiple synapse types

"""

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

(continues on next page)

(continued from previous page)

```
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")
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)
```

3.8 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.

```
"""
In this example an axon is built, a morphology is loaded, the axon is
then connected to the loadeed morphology. The whole thing is serialized
in JSON format, reloaded and validated.
"""

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)
```

(continues on next page)

(continued from previous page)

```
#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

json_file = './tmp/modified_morphology.json'

writers.JSONWriter.write(doc,json_file)

print("Saved modified morphology in JSON format to: " + json_file)


##### load it again, this time write it to a normal neuroml file ###

neuroml_document_from_json = loaders.JSONLoader.load(json_file)

print("Re-loaded neuroml document in JSON format to NeuroMLDocument object")

nml_file = './tmp/modified_morphology_from_json.nml'

writers.NeuroMLWriter.write(neuroml_document_from_json,nml_file)

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

from neuroml.utils import validate_neuroml2
```

(continues on next page)

(continued from previous page)

```
validate_neuroml2(nml_file)
```

3.9 Working with arraymorphs

```
"""
Example of connecting segments together to create a
multicompartmental 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)
```

(continues on next page)

(continued from previous page)

```
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)
```

3.10 Working with Izhikevich Cells

These examples were kindly contributed by Steve Marsh

```
#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)
```

(continues on next page)

(continued from previous page)

```
load_izhikevich()
```

```
from neuroml import NeuroMLDocument
from neuroml import IzhikevichCell
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(id="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()
```

CHAPTER 4

Useful tools

Below is a list of tools which are built in or use Python and which would benefit from a standard library to access, modify and save detailed neuronal morphologies. Developers from most of these initiatives are involved with the libNeuroML project.

4.1 Neuronvisio

Neuronvisio is a Graphical User Interface for NEURON simulator environment with 3D capabilities.

http://neuronvisio.org (GitHub: https://github.com/mattions/neuronvisio)

4.2 PyNN

PyNN is a is a simulator-independent language for building neuronal network models.

http://neuralensemble.org/trac/PyNN

4.3 Morphforge

A Python library for simulating small networks of multicompartmental neurons

https://github.com/mikehulluk/morphforge

4.4 CATMAID

We reconstruct neuronal circuits (morphology in 3D, synaptic connectivity) as skeletons, surfaces, volumes in CATMAID. We want to be able to export the data into an object model (data format), complement it with ion channel

distribution of several types & synaptic mechanisms, and simulate the membrane voltage time series and do virtual current injection etc. on standard simulators. All of this with a easy-to-use, intuitive Python API in a few lines of code.

http://www.catmaid.org

4.5 NEURON

A widely used simulation platform for biophysically detailed neurons and networks which has recently added a Python interface.

http://www.neuron.yale.edu/neuron

For more information on Python & NEURON, see Andrew Davison's guide here: http://www.davison.webfactional.com/notes/installation-neuron-python/

4.6 MOOSE & Moogli

MOOSE is the Multiscale Object-Oriented Simulation Environment. It is the base and numerical core for large, detailed simulations including Computational Neuroscience and Systems Biology.

http://moose.sourceforge.net

PyMOOSE

The latest version of MOOSE with a Python interface can be installed as follows:

```
svn co http://moose.svn.sourceforge.net/svnroot/moose/moose/branches/dh_branch moose
cd moose
make pymoose
sudo cp -r python/moose /usr/lib/python2.7/dist-packages
```

replacing */usr/lib/python2.7/dist-packages* with the appropriate location for your Python packages. More details can be found here.

An example of the HH squid mode can be run with:

```
cd Demos/squid/
python squid_demo.py
```

Moogli

Moogli (a sister project of MOOSE) is a simulator independent OpenGL based visualization tool for neural simulations. Moogli can visualize morphology of single/multiple neurons or network of neurons, and can also visualize activity in these cells.

http://moose.ncbs.res.in/moogli/

4.7 neuroConstruct

neuroConstruct generates native simulator code for NEURON, MOOSE and other simulators. It would be a great benefit to be able to generate pure NeuroML descriptions of the model components and run (nearly) identical Python code on these simulators to load the NeuroML and execute the simulations. This scenario is implemented already for a limited number of model types by generating PyNN based scripts which can run on NEURON, Brian and NEST.

http://www.neuroConstruct.org

CHAPTER 5

Developer documentation

5.1 How to contribute

To contribute to libNeuroML you need a github account then you should fork the repository at https://github.com/NeuralEnsemble/libNeuroML

Note: Fork is not a bad thing on a Github workflow. Fork basically means you have your own repository which is connected with upstream (the main repository from which official releases will be made). You can contribute back to upstream using Pull Request

5.1.1 Setting up

Have a quick view at the doc: http://help.github.com/fork-a-repo/

1. Fork the repo (done on github website). Now you should have a libNeuroML under you username (mine for example sits happily here: https://github.com/mattions/libNeuroML)
2. Clone your repo locally (This is done once!)

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

3. Add upstream as remote branch which you follow

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

You can check which branch are you following doing:

```
git branch -a
```

you should have something like:

```
mattions@triton:libNeuroML(master*)$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/upstream/master
```

This means you are currently on branch master and there are two remotes branches `origin/master` which is your origin master (the branch where you master gets pushed automatically and `upstream/master` which is the upstream master (the NeuroEnsemble one).

5.1.2 Sync with upstream

Before starting to do some work, I'll suggest to get the latest development going on the upstream repo

```
git fetch upstream
git merge upstream/master
```

If there are no conflict, you are all set, if there are some you can solve them with

```
git mergetool
```

which will fire up your favourite merger to do a 3-ways merge.

3-ways means you will have your *local* file on your left, the *remote* file on your right, and the file in the middle is the conflicted one, which you need to solve.

A nice 3-ways merger makes this process very easy, and merging could be fun. To see what you have currently installed just do `git mergetool`

This is my response

```
mattions@triton:libNeuroML(master*)$ git mergetool
merge tool candidates: meld opendiff kdiff3 tkdiff xxdiff tortoisemerge gvimdiff␣
↪diffuse ecmerge p4merge araxis bc3 emerge vimdiff
No files need merging
```

[Meld] (http://meldmerge.org/) is the first of the list and would be automatically picked up by `git mergetool`. Chose your favourite.

Well done, now you are all set to do some cool work!

5.1.3 Working locally on a dedicated branch

Now you can work on your repo. The best way to do it is to create a branch with a descriptive name which indicate what are you working on.

For example, just for the sake of this guide, I'm going to close #2

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

Now, I'm working on a different branch from master which is fix-2 This will come handy in a minute.

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

Now that I found how to fix this issue, I just want to push my branch online and open a pull request.

1. Push the branch online

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

2. Open the pull request

Here I want to open a pull-request to integrate `fix-2` into `upstream/master`

To do that I click Pull-Request and automatically a new Issue #3 is created where it is possible to comment.

If your code is not ready to be include, you can update the code on your branch and automatically the Pull Request will sync to the latest commit, so it is possible to change it after the Pull Request is started. Don't be scare to open one.

5.1.4 Release process

libNeuroML is part of the official NeuroML release cycle. As of 1/09/13 we are still ironing out the proecure. When a new libNueroML 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

5.1.5 Miscellaneous

- Nice guide about git
- Quick reference for git
- Remember to tell git your name, so we know who contributes!
- Always known in which branch you are using this bash function

5.2 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.

5.2.1 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.

5.2.2 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.

5.2.3 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.

modules

5.3 Indices and tables

- genindex
- modindex
- search