# Additional Setup Information for Configuration/Build files

## General Information

1. In the directory where you ran setup there should be several new files and folders.
   1. *run\_bionet.py*
      1. Used from the command line to start a simulation
   2. *circuit\_config.json*
      1. Specifies where BMTK will look for files related to:
         1. Cell morphologies
         2. Synaptic models
         3. mechanisms (compiled mod files)
         4. biophysical neuron models
         5. point neuron models
         6. Network nodes and edges
      2. This file will remain relatively unchanged through simulations
   3. *simulation\_config.json*
      1. You can create as many of these files as you want as it’s specified when running each simulation using run\_bionet.py. You can specify separate simulations with different
         1. run configurations (stop time, spike threshold, etc)
         2. conditions (temperature, v\_init)
         3. inputs (current clamp, artificial)
         4. output (spikes from different cell types)
         5. reports
   4. *network/*
      1. When you add nodes and build the network (later) the csv and h5 files reside here until you run the network
   5. *biophys\_components/biophysical\_neuron\_templates/*
      1. Holds fit **cell json files** defining cell properties/parameters like conductance
      2. When creating a network and specifying new cells using the ***add\_cells()*** method, the parameter ***dynamics\_params*** looks here for a json file
   6. *biophys\_components/mechanisms/*
      1. Neuron mkdll or nrnivmodl **compiled code or dll file must** be placed here
   7. *biophys\_components/mechanisms/modfiles/*
      1. Location of **mod files**
   8. *biophys\_components/morphologies/*
      1. **SWC files** for biologically realistic cells placed here.
   9. *biophys\_components/synaptic\_models/*
      1. **Synapse files**

## Network Build Script for SWC/JSON

1. To begin, **you’ll need to create a new script file,** **build\_network.py,** responsible for specifying the cells to be used in your network. You can use the following as a template:

|  |  |
| --- | --- |
|  | **build\_network.py** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | from bmtk.builder.networks import NetworkBuilder  net = NetworkBuilder('mcortex')  net.add\_nodes(cell\_name='Scnn1a\_473845048',  potental='exc',  model\_type='biophysical',  model\_template='ctdb:Biophys1.hoc',  model\_processing='aibs\_perisomatic',  dynamics\_params='472363762\_fit.json',  morphology='Scnn1a\_473845048\_m.swc')  net.build()  net.save\_nodes(output\_dir='network')  from pprint import pprint  for node in net.nodes():  pprint(node) |

* 1. ***Line 5 -*** cell\_name (optional) – record keeping and arbitrary to your model
  2. ***Line 6 -*** potential (optional) – Indicate it’s an excitatory cell (‘exc’,’inh’)
  3. ***Line 7 -*** model\_type – can be (‘biophysical’, point\_process’, ‘point\_soma’, ‘virtual’)
     1. Reference: <https://github.com/AllenInstitute/bmtk/blob/develop/bmtk/simulator/bionet/bionetwork.py#L54>
     2. At this point (11/18) ‘point\_soma’ is not implemented
  4. ***Line 8 -*** model\_template – signals we’re using a biophysical cell
     1. Biophys1.hoc is located in BMTK installation at /bmtk/simulator/bionet/default\_templates/Biophys1.hoc
     2. (BioAxonStub.hoc, Biophys1.hoc, advance.hoc) are also available.
  5. ***Line 9 -*** model\_processing – internal method for processing Allen institute files. See bmtk\simulator\bionet\default\_setters\cell\_models.py for all methods you can use.
  6. ***Line 10 -*** dynamics\_parameters – **parameters json file** used to specify parameters of a cell located in (circuit\_config.json -> biophysical\_neuro\_models\_dir) typically (*./biophys\_components/biophysical\_neuron\_templates/*)
  7. ***Line 11 -*** morphology – **swc morphology file** located in
  8. ***Line 14 –*** output\_dir was the location specified in “Installation/Setup” step 3.

## Cell SWC and JSON Config Files

1. Preparing Dynamics parameters and SWC Files
   1. Initial notes:
      1. SWC Files are 3d representation files of real neuron cells
         1. See <http://www.neuromorpho.org/myfaq.jsp> for more
      2. **Swc files** have to be in biophys\_components/morphologies directory
      3. **Cell json files** have to be in biophys\_components/biophysical\_neuron\_templates
      4. Names of these files need to be the same as what you specified in the **build\_network.py** script
      5. You can download a base set of cells to play with from:
         1. <http://celltypes.brain-map.org/neuronal_model/download/482934212>
         2. These cells fit in the example provided in <https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/01_single_cell_clamped.ipynb> and referenced through this tutorial.
         3. Info: <https://www.neuron.yale.edu/phpBB/viewtopic.php?t=3477>
   2. **SWC Files**
      1. Format

|  |
| --- |
| **http://research.mssm.edu/cnic/swc.html** |
| **n T x y z R P**  **n** is an integer label that identifies the current point and increments by one from one line to the next.  **T** is an integer representing the type of neuronal **segment you’re starting**, such as soma, axon, apical dendrite, etc. The standard accepted integer values are given below.   * 0 = undefined * 1 = soma * 2 = axon * 3 = dendrite * 4 = apical dendrite * 5 = fork point * 6 = end point * 7 = custom   **x**, **y**, **z** gives the Cartesian coordinates of each node.  **R** is the radius at that node.  **P** indicates the parent (the integer label) of the current point or -1 to indicate an origin (soma). |

* + 1. The cells MUST contain an axon component (2) I’m not sure if this will be a problem for some models. There must be 3 items identified.

|  |
| --- |
| **biophys\_components/morphologies/simple.swc** |
| #Sample cell morphology  1 1 0 0 0 5 -1  2 2 1000 0 0 5 1  3 2 1000 0 0 0.1 2 |

* + 1. SWC files are required
       1. The above cell is a soma with length 1000 and radius 5 (diam=10) with an unrealistic required axon (cm will be 0 later) due to neuron 3d limitations.
    2. I used NeuTube <http://www.neutracing.com> for editing manually initially.
    3. Allen Institute references **VAA3D** for viewing
       1. Here: <https://alleninstitute.org/bigneuron/algorithms/>
       2. See homepage: <http://home.penglab.com/proj/vaa3d/home/index.html>
       3. Downloaded from <https://github.com/Vaa3D/release/releases/>
  1. **Cell JSON files**
     1. The cell properties like axial resistance, capacitance and channel conductance are all editable from here
        1. For a full example to be used in your model, download <http://celltypes.brain-map.org/neuronal_model/download/482934212>
     2. A json file to go with the **simple.swc** specified above would look like:

|  |
| --- |
| **biophys\_components/biophysical\_neuron\_templates/simple.json** |
| {  "passive": [  {  "ra": 1,  "cm": [  {  "section": "soma",  "cm": 1.0  },  {  "section": "axon",  "cm": 0.0  }  ],  "e\_pas": -50  }  ],  "conditions": [  {  "erev": [  {  "ena": 50.0,  "section": "soma",  "ek": -80.0,  "eleak":-50  }  ]  }  ],  "genome": [  {  "section": "soma",  "name": "gbar\_leak",  "value": 0.00003,  "mechanism": "leak"  },  {  "section": "soma",  "name": "gbar\_na",  "value": 0.3,  "mechanism": "na"  },  {  "section": "soma",  "name": "gbar\_kdr",  "value": 0.3,  "mechanism": "kdr"  }  ]  } |

* + 1. Soma and Axon must be defined

## Simulation Environment (Compilation and Reports)

1. Setting up the simulation environment
   1. **Windows**
      1. Methods specified in the official documentation don’t work because nrnivmodl doesn’t exist in Windows. Use **mknrndll gui** instead.
      2. Edit the simulation\_config.json file to specify the reports you want to run

|  |  |
| --- | --- |
|  | **simulaton\_config.json** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | "reports": {  "membrane\_report": {  "module": "membrane\_report",  "cells": "all",  "variable\_name": [  "v"  ],  "file\_name": "cell\_vars.h5",  "sections": "soma"  }  }, |

* + 1. Use **mknrndll** to compile the components/mechanisms/modfiles directory
    2. Move the nrnmech.dll file up one directory to components/mechanisms/
  1. **Linux/Mac**
     1. You can either:
        1. Compile manually and edit a config file

cd components/mechanisms

nrnivmodl modfiles

|  |  |
| --- | --- |
|  | **simulaton\_config.json** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | "reports": {  "membrane\_report": {  "module": "membrane\_report",  "cells": "all",  "variable\_name": [  "cai",  "v"  ],  "file\_name": "cell\_vars.h5",  "sections": "soma"  }  }, |

* + - 1. Build from the command line:

python -m bmtk.utils.sim\_setup -n network --membrane\_report-vars v,cai --membrane\_report-sections soma --tstop 2000.0 --dt 0.1 bionet

* + - 1. Or create a python file with build instructions and run

|  |  |
| --- | --- |
|  | **build\_network.py** |
| 1  2  3  4  5  6  7  8  9  10  11  12 | from bmtk.utils.sim\_setup import build\_env\_bionet  build\_env\_bionet(network\_dir='network', tstop=2000.0, dt=0.1,  reports={'membrane\_report': {  "module": "membrane\_report",  'cells': 'all',  'variable\_name': ['cai', 'v'],  "file\_name": "cell\_vars.h5",  "sections": "soma",  }}) |

## Current Clamp Input

1. Current clamp input (**OPTIONAL**)
   1. In the **simulation\_config.json** file place the following code block in the input section

|  |  |
| --- | --- |
|  | **simulaton\_config.json** |
| 1  2  3  4  5  6  7  8  9  10  11  12 | "inputs": {  "current\_clamp": {  "input\_type": "current\_clamp",  "module": "IClamp",  "node\_set": "all",  "amp": 0.120,  "delay": 500.0,  "duration": 1000.0  }  }, |

* + 1. “node\_set” can be set to a list. Using [0] would reference the soma for all models
  1. It’s possible to have multiple clamps at different times and amps through a single simulation. Just duplicate the “current\_clamp” section, separated by a comma. Ex:

|  |  |
| --- | --- |
|  | **simulaton\_config.json** |
| 1  2  3  4  5  6  7  8  9  10  11  12 | "inputs": {  "cclamp1": {  "input\_type": "current\_clamp",  "module": "IClamp",  "node\_set": "all",  "amp": 0.150,  "delay": 0.0,  "duration": 400.0  },  "cclamp2": {  "input\_type": "current\_clamp",  "module": "IClamp",  "node\_set": "all",  "amp": 0.300,  "delay": 500.0,  "duration": 400.0  }  }, |

## Virtual Input

1. Virtual spike input (**OPTIONAL**)

# <https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/02_single_cell_syn.ipynb>

## Networked Cells

<https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/03_single_pop.ipynb>

<https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/04_multi_pop.ipynb>

# Running the simulation and Output

## Running Bionet

1. To **Run** your network execute:

python run\_bionet.py simulation\_config.json

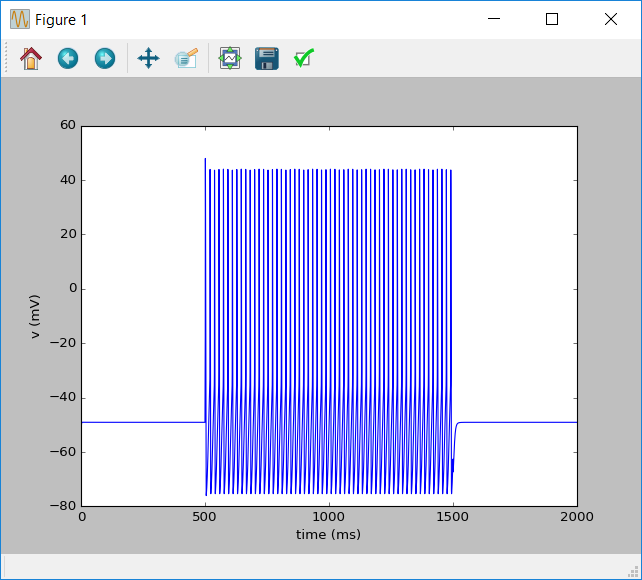
## Viewing Results

1. View spike times:

|  |  |
| --- | --- |
|  | **spikes.py** |
| 1  2  3  4  5 | from bmtk.analyzer.spike\_trains import to\_dataframe  to\_dataframe(config\_file='simulation\_config.json') |

1. View output graphs specified in the “reports” section

|  |  |
| --- | --- |
|  | **graphs.py** |
| 1  2  3  4  5 | from bmtk.analyzer.cell\_vars import plot\_report  plot\_report(config\_file='simulation\_config.json') |



1. View raster plot for population of cells

|  |  |
| --- | --- |
|  | **raster.py** |
| 1  2  3  4  5  6  7 | from bmtk.analyzer.spike\_trains import raster\_plot  raster\_plot(cells\_file='network/mcortex\_nodes.h5', cell\_models\_file='network/mcortex\_node\_types.csv', spikes\_file='output/spikes.h5') |

# Additional References

<https://github.com/AllenInstitute/bmtk/blob/develop/docs/tutorial/01_single_cell_clamped.ipynb>

<https://github.com/AllenInstitute/sonata/blob/master/docs/SONATA_DEVELOPER_GUIDE.md#nodes---required-attributes>

<https://github.com/AllenInstitute/bmtk/tree/develop/docs/tutorial>

Things to remember

1. Compilation needs to be done manually for Windows systems
2. Nrnmech.dll has to be in the directory above modfiles – biophys\_components/mechanisms
3. Swc files have to be in the morphologies directory
4. Fit\_json files containing cell parameters have to be in biophys\_components/biophysical\_neuron\_templates

Other notes:

Bmtk/simulator/bionet/default\_templates/BioAxonStub.hoc is missing load\_file("import3d.hoc") at begining (<https://www.neuron.yale.edu/phpBB/viewtopic.php?t=3257>)

Useful: SWC Viewer from different databases. No downloading

[https://neuroinformatics.nl/HBP/morphology-viewer/#](https://neuroinformatics.nl/HBP/morphology-viewer/)

# Questions for Allen Institute

1. Single compartment cells - a couple of our models are very small and the morphology/electrophysiology is entirely defined in hoc. How can BMTK handle something simple like a single hoc file? Much of BMTK is centered around JSON and SWC for the Allen Institute's cell database.

2. Is there any way around using SWC files for cell morphology? I noticed NeuroML is an acceptable format, which may make question 1 possible through hoc to nml conversion but the morphology reader in BMTK hasn't been implemented. Are there any SWC alternatives?

1 & 2)  
In theory bmtk should be able to read in any NEURON template file. In the nodes.csv file update model\_template to "hoc:MyNRNTemplate.hoc", where MyNRNTemplate.hoc is the name of your template file (if the path is not absolute or in your working directory it will look under "components/templates" in the config.json). The template must have a "soma" section, but it will ignore the SWC file and assume the morphology is being built into the template  
  
However it that doesn't work you can override the way bmtk loads each individual cell. To do so add the following to the run\_bionet.py script (before the \_\_main\_\_ section:

def loadHOC(cell, template\_name, dynamics\_params):  
 param1 = cell['param1'] # Use cell dict to get each cell's attributes,  
 tau = dynamics\_params['tau'] # dynamics params values store in json file (is not None)  
 hobj = ... # Create and return a NEURON HOCObject  
 return hobj  
bionet.pyfunction\_cache.add\_cell\_model(loadHOC, directive='hoc', model\_type='biophysical')  
  
Inside the loadHOC method you can call another NEURON hoc file, or build the cell directly using neuron's python api. It will ignore the swc files unless you want it too. BMTK will execute this function for every cell with a model\_template parameter that starts with "hoc:". If you run into any issues overriding the loadHOC method please feel free to make or github issue or just email me directly (kaeld@alleninstititute.org) and we can try to fix the issue.

3. If we were to implement the previously mentioned morphology reader for NML files would this be a complete solution for the previous questions? Is this something that has been considered for near future development there?

You can use the same method as above to update the way NML templates are loaded. Here is a link to the current way bmtk loads NML based cells.  
  
There is no immediate plans to read and load the morphology from the NML method. And unfortunately, last time I checked, there doesn't seem to be an easy way in python to convert a NML <cell> into a HOC object (I had to write a primitive parser just to read the membrane properties). But if you have a method to do so you can override the NMLLoad function - and if you do let me know because I think it would be a good feature to add to the main branch.

4. How customizable are the synapse JSON files? Specifically, for neuromodulation effects for short/long term plasticity.

You can use any synapse mechanism built into NEURON or that you've created and compiled yourself. The json files just have to match up with the parameters for that given synapse.  
  
We actually have our own custom stp synapses, and are used in the bio\_stp\_models examples. If you look under network/ext\_to\_slice\_edge\_types.csv you'll see for a certain number of connections the model\_tempate is set stp2syn which uses different json file than the standard exp2syn.

5. Is it possible to record LFPs from a set of cells?

You can turn on LFP recordings in the config.json file, as part of the "reports" section. Here is an example config with lfpy recordings of the first 10 cells.  
  
You will likely need to adjust the electrode.csv to coordinates that are more appropriate for your network. After the simulation is finished it will record and sum up the lfp's from all "cells" as an hdf5 into file\_name specified (which is just a table where each row is a given time step and each column a different electrode). It will also store each's cell's individual lfp contribution into the "contributions\_dir", unless that value is null or removed from the config.