PyMOOSE

Python module for MOOSE Simulation
Version 1.3

Subhasis Ray (subhasis at ncbs dot res dot in)

This manual is for MOOSE (version 1.3), The Multiscale Object-Oriented Simulation Environment.

Table of Contents

Executive Summary

It contains an overview of how to use MOOSE as a Python module to setup and run simulations. It also supplements as a reference manual describing the classes of objects in MOOSE.

1 Copying this document

This manual is for MOOSE (version 1.3), The Multiscale Object-Oriented Simulation Environment.

2 Introduction to MOOSE and PyMOOSE

MOOSE is a general purpose simulation environment, and PyMOOSE its incarnation as a Python module. MOOSE stands for Multiscale Object-Oriented Simulation Environment.

The 'multiscale' comes from the fact that one can simulate systems spanning multiple scales: from a few molecules reacting with each other (Gillespie simulations) to large networks of neurons to simulate areas of the brain.

It is 'object-oriented' in the sense that it represents the simulation entities as instances of particular classes and these classes are organized in a hierarchy where a child inherits the properties of its parent.

2.1 Modeling for simulation

Computers are limited by the available memory and the speed of the electronics, whereas anything from reality has infinite complexity (if you have any doubt, see the beautiful description of the complexity of a glass of wine by Richard Feynman)[1]. Hence, when simulating any system from reality, we have to consider carefully what details are important and what to leave out.

This leads us to abstraction. We use the existing scientific knowledge to make abstract representations of entities from reality. These are called `objects`. We use `objects` as the building blocks for more complex models. An `object` in a model may be represent a real-life object or some abstract concept. How do we model the interaction between the objects in real life? Through `messages` between interacting `objects`. Also, the pieces information relevant to the situation are attached to the `objects` as `fields`. When we set up these abstractions and interactions, we have a model of the system.

As an illustration, consider modeling a simple chemical reaction, say the oxidation of Nitric oxide(NO) to Nitrogen dioxide(NO2). There are some NO molecules and some O2 molecules and depending on the concentrations they react at certain rate and form NO2. If you go too deep into reality, there is a mind boggling amount of detail: NO molecules and O2 molecules roaming around randomly inside a container at very high speed, hitting each other and the walls of the container and once in a while there will be some exchange of electrons and an atom of oxygen will stick with the NO to share them.

But we also know that as long as we are worrying only about the amount of the end product NO2 at any given time, all these details can be forgotten and a more simple law can be applied if we have some empirical information, viz., the initial concentration of each of the gases, the rate constants of the forward and backward reactions and how many molecules of each reactant gas react to produce how many molecules of the product. Given these, we can invoke the laws of chemistry to calculate the concentration of each molecule at any given time. When we compute the series of values these molecular concentrations take at a given sequence of points in time, we call that a simulation.

All this can be modeled and simulated in MOOSE as follows:

The gases will be contained in some closed volume: we represent this with a `KinCompt` object (the name is a programmers' abbreviation for COMPartmenT for chemical KINetics). We set the `volume` field of the `KinCompt` object.

[1] Feynman Lectures on Physics, Volume I, pp 3-10, The relation of Physics to other sciences

The pool of NO molecules will be modeled as *a* `Molecule` object and we set the number of molecules by setting the the field `n` of the Molecule object. Similarly, the pools of O2 and NO2 molecules will be modeled as `Molecule` objects (If initially there is no NO2, we set `n = 0` for it). When we set the number of molecules, the concentration is computed automatically from the volume of the `KinCompt` object. Alternatively, we can set the concentrations directly in the `conc` field of the `Molecule` objects.

Now the reaction itself is determined by the forward and backward rate constants. The `Reaction` class is precisely for this. We create a `Reaction` object and set its `Kf` and `Kb` fields to the forward and backward rate constants.

Now to implement the interaction between these, we set some messages between particular fields of different objects. We have to setup the substrate and product relationships of the `Molecules` with the `Reaction`. This is done by connecting the `'reac'` field of the O2 and NO `Molecule` objects with `'sub'` field of the `Reaction` object and connecting the `'reac'` field of the NO2 `Molecule` with the `'prd'` field of the `Reaction` object.

There are bunch of `Clock` objects that tick at some specified intervals. Each object is associated with some clock which controls the update interval of its state variables. Once the model is set up we can do a 'reset' to bring everything to an initial condition and then 'run' the simulation. Running the simulation causes the clocks to start. As each clock progresses, the `conc` field of the `Molecules` will be updated with the computed concentration for that time.

3 Installing (Py)MOOSE

MOOSE is available both as source code and platform-specific installer packages. However easy the platform-specific installers are to use, there may be issues with dependencies. Hence the most universal way of installing MOOSE is to build it from the sources.

3.1 Prerequisites

- GNU Make
- GNU C and C++ compiler

  gcc and g++ are required for compiling the C++ source code of MOOSE.
- Python development headers

  Python.h should be located somewhere in the include search directories. If it is in a non standard directory, you can edit pymoose/Makefile to add this directory in the include search path with −I{your_python_include_directory} argument.
- GNU Scientific Library

  GSL provides integration methods for some classes in MOOSE.
- libsbml

  The default build requires libsbml to be installed. You can download it from SBML website (`http://sbml.org`).

3.2 Configuration

Right now all configuration is done by editing the `Makefile` of MOOSE. You can have some control over what all are included in the compilation by passing commandline arguments to `make`. These options are described in the next section (Build/install).

3.3 Build/install

To build **pymoose** run make in the moose source code's top-level directory with pymoose as the targte:

make pymoose

This will do the default build. There are several options you can pass to make for customizing the build. Pass the options as key-value pairs in the make commandline as follows:

make pymoose OPTION1=value1 OPTION2=value2 OPTION3=value3

Most of the options are for enabling or disabling some feature. They are listed below:

- BUILD=release,debug (default value: release)

  BUILDTYPE can be either `debug` or `release`. Passing `debug` will build an unoptimized version with debugging symbols for gdb and will enable the built-in C++ unit tests.

  `release` will build an optimized version without debugging symbols or unit-tests.

  If unspecified, BUILD defaults to `release`.
- USE_GSL=1,0 (default value: 1) Use GNU Scientific Library for integration in kinetic simulations.

- USE_SBML=1,0 (default value: 1) Compile with support for the Systems Biology Markup Language (SBML). This allows you to read and write chemical kinetic models in the simulator-indpendent SBML format.
- USE_NEUROML=1,0 (default value: 0) Compile with support for the NeuroML. This allows you to read neuronal models in the NeuroML format. Look in external/neuroML_src/README for the extra steps needed to add the libraries & headers.
- USE_READLINE=1,0 (default value: 1) Use the readline library which provides command history and better command line editing capabilities
- USE_MPI=1,0 (default value: 0) Compile with support for parallel computing through MPICH library
- USE_MUSIC=1,0 (default value: 0) Compile with MUSIC support. The MUSIC library allows runtime exchange of information between simulators.
- USE_CURSES=1,0 (default value: 0) To compile with curses support (terminal aware printing)
- USE_GL=1,0 (default value: 0) To compile with OpenSceneGraph support to enable the MOOSE elements 'GLcell', 'GLview'.
- GENERATE_WRAPPERS=1,0 (default value: 0) Useful for python interface developers. The binary created with this option looks for a directory named 'generated' in the working directory and creates a wrapper class (one .h file and a .cpp file) for each moose class and partial code for the swig interface file (pymoose.i). These files with some modification can be used for generating the python interface using swig.

The build process for pymoose generates two final output files in the top level directory: `moose.py` and `_moose.so`. Copy these to any location in your PYTHONPATH environment variable (for Python 2.6 you can copy them to '`dist-packages`' directory inside your python installation directory (for UNIX-like systems it is: '`/usr/lib/python{version}`' or '`/usr/local/lib/python{version}`' Additionally, there is a file with some utility functions '`pymoose/pymoose.py`'. It has functions like `showmsg`, `printtree`, etc. You may copy this file along with moose.py to use these functions.

3.4 Loading

Once you have built and installed PyMOOSE, you can load it in the Python interpreter by the statement: `import moose` You should see a few informational outputs, ending with:

```
SIMPATH = .:your_home_directory
nnnn FuncVecs built for the first time
```

where *your_home_directory* stands for the full path of the user's home directory and *nnnn* stands for some number (2200 in my case).

4 Quick Start

5 In-Depth Guide to PyMOOSE

In this chapter we go through the internal details of PyMOOSE.

5.1 Overview of PyMOOSE internals

The entities in a MOOSE model are called `elements`. An `element` is an object of some MOOSE class. Elements are the basic building blocks of a model. You set the properties of the individual elements and connect them via messages to build a model.

5.1.1 Id - the unique identifier of each MOOSE object

Each element has a unique identifier, called its `id`. PyMOOSE provides Pythonic wrappers around this id. For every MOOSE class, we have a corresponding Python class provided by PyMOOSE. This class essentially creates an illusion of mirrorring the MOOSE class via a set of properties which correspond to the fields in the MOOSE class. The main data contained in instances of the PyMOOSE classes is only a copy of the `id` of the original MOOSE object. All the fields are accessed on-demand via function calls using this `id` as the handle.

5.1.2 Class hierarchy of MOOSE

MOOSE provides a class hierarchy which is closely imitated by the PyMOOSE classes. `Neutral` is at the top of the class hierarchy. So instance of any other MOOSE class is also an instance of `Neutral` class. A Neutral object is most often used as a container of other objects.

Major areas of focus in MOOSE are neuronal simulations and biochemical simulations. So many of the classes are for modeling these scenarios. Some of the commonly used classes in Neuronal models are:

- Compartment - models an isopotential fragment of a neuronal cable.
- HHChannel - Hodgkin-Huxley-type ion channel.
- SynChan - Synaptic channel.
- SpikeGen - Spike generator. This models the presynaptic terminal and is connected to the SynChan on the postsynaptic side.
- Cell - Represents a single neuron. Practically it is a container for connected compartments that constitute a neuron.

Commonly used classes in biochemistry/chemical kinetics models are:

- Molecule - A pool of molecules.
- Enzyme - An abstraction of enzymatic mechanism.
- Reaction - A general chemical reaction with forward and backward rate constants.
- KinCompt - A volume in space in which molecules exist/reactions take place.

Some classes are there to provide utilities and infrastructure for doing simulation experiments:

- Table - A versatile list-like structure which allows interpolation, data-recording, sending pre-assigned sequence of data at each time step tp some target object and saving data to file.

- AscFile - A general text(ascii) file handle.
- Random number generators - a whole bunch of them to generate samples from various distributions.

TODO: incorporate class-herarchy diagram

5.1.3 Messaging

Any complex model is composed of multiple components called `elements` which represent some well defined biological concept, like an ion-channel or an enzyme. The components of a model are connected to each other via messages. These messages allow state variables of one element to be seen by the other. For example, a synaptic channel has a conductance `g` which depends on the membrane potential `Vm` of a neuronal compartment. `Vm` on the other hand changes with change in `g`. Thus, we need a message sending `Vm` from compartment to synapse and a message sending `g` from synapse to compartment. These two messages are combined in another message, `channel`.

> Take, for example, a simple reaction: A + B <---> C. By convention, the rate of the forward reaction is represented by *Kf* and that of the backward reaction is represented by *Kb*.
>
> Now, **A**, **B** and **C** are molecular species and MOOSE provides the class Molecule to model them. The main property of a Molecule element is *n*, the number of molecules.
>
> As you will immediately recognize, just knowing the number of molecules does not help in calculating the progress of a chemical reaction with time. It is concentration that matters. But to obtain concentration from number of molecules, you need the volume of the container. This container is called a kinetic compartment, which need not be a real container, but any volume in space whithin which the molecules are homogeneously distributed. Kinetic compartments are represented by the class KinCompt in MOOSE.
>
> TODO: to be completed with an walk-through to developing the simulation.

5.2 Element Tree

All elements in MOOSE are part of a tree structure. We call this `Element Tree` or `Model Tree`. This is similar to the folder structure in the file system of your computer. Each entry in this tree is a moose object and we call it an **element**. The top level element is called the `root` element (represented as '`/`'). Every element other than `root` has a parent element. There are some predefined special elements that are used for management of the system. They are created when you startup moose (or `import` moose in Python). The following diagram shows this structure:

```
root
|
|__ shell
|  |
|  |__ sli
|  |
|  |__ BaseContext
|
|__ sched
|  |
|  |__ cj
|     |
|     |__ t0
|     |
|     |__ t1
|
|__ library
|
|__ proto
```

5.3 Id

The unique identifier for each MOOSE object is an Id. This is accessible as `id` field of pymoose objects. An Id object has two components,

- id an unsigned integer. The method `id` returns this value. The root element of the moose element tree always has the id 0.
- index an unsigned integer giving the index number of array-elements. For simple elements, it is 0. It is returned by the method `index`.

When you print an `Id` object, it is printed in the form: `id[index]`

Example:

```
>>> foo = moose.Neutral('foo') # create a Neutral object called 'foo'
>>> foo.id
<moose.Id; proxy of <Swig Object of type 'Id *' at 0x248c6f8> >
>>> print foo.id
470[0]
>>> print foo.id.id()
470
>>> print foo.id.index()
0
```

Even if you create multiple python objects wrapping the same moose element, you can always verify if the underlying moose element is the same by comparing their `id`s.

Example:

```
>>> a = moose.Neutral('my_test_object')
>>> b = moose.Neutral('my_test_object')
>>> a.id == b.id
True
```

Note that `id` is a more fundamental property than the path string of an object. The path string changes when you change the name of the object, but the `id` remains unchanged.

Example:

```
>>> a = moose.Neutral('my_test_object')
>>> print a.path
/my_test_object
>>> ii = a.id
>>> a.name = 'your_test_object'
>>> print a.path
/your_test_object
>>> ii == a.id
True
```

In addition, any `Id` object is hashable. The hash is computed as the hash of its string representation. Thus you can use them as keys in a `dict`.

5.4 Neutral

`Neutral` captures the fundamental properties of all the MOOSE elements. If you are familiar with Object Oriented Programming, you can recognize that this is the base class of all moose classes whose instances can be in the element tree.[1]

Being the base class of all other MOOSE classes, properties of `Neutral` class is common to all MOOSE classes. A `Neutral` object can wrap any valid moose object.

You can construct a Neutral object in many ways.

'`From a path string:`'

First, you can just give a path-string as the parameter to the constructor. If there is already an object with the given path, then you get a wrapper around the existing object. On the other hand, if no such object exists, it will try to create a new `Neutral` object with the given path.

[1] Not all classes in MOOSE are element-classes. The `Id` class, for example.

```
>>> foo_neutral = moose.Neutral('/foo')
>>> bar_neutral = moose.Neutral('/foo')
>>> foo_neutral.path
'/foo'
>>> bar_neutral.path
'/foo'
>>> bar_neutral.name = 'bar'
>>> foo_neutral.name
'bar'
```

As you can see in the above example, `bar_neutral` is just a wrapper around `foo_neutral` and thus changing a field in one of them will be reflected in the other.

Note that when you specify a path to the constructor, it has to exist up to the parent object.

'`From an Id:`'

You can wrap the Id of any existing MOOSE object inside a `Neutral`. The following example shows a common idiom used for looping through the list of children of an object.

```
>>> root = moose.Neutral('/')
>>> for child_id in root.childList:
...     child_obj = moose.Neutral(child_id)
...     print child_obj.path, 'is actually of class', child_obj.className
...
/shell is actually of class Shell
/sched is actually of class Neutral
/library is actually of class Neutral
/proto is actually of class Neutral
```

'`Specifying a parent and name:`'

You can also give the name of the objct to be created and specify the parent. You can call the constructor like this:

`>>> foo = moose.Neutral('foo', parent)` Here `parent` can be another PyMoose object or an `Id`. Thus, if the path of the parent is `'/bar'`, then path of `foo` will be: `'/bar/foo'`

'`Copy an existing object:`'

You can also make a copy of an existing object. The constructor is called like: `foo = moose.Neutral(src, new_name, parent)` or `foo = moose.Neutral(src, path)`

Here `src` can be another `Neutral` object or the `Id` thereof. `new_name` is a string specifying the name of the duplicate and parent is any PyMoose object or an Id. `path` is a string specifying the path of the duplicate object.

The following fields are available in Neutral class and hence in all element classes:

'`className`'
: The name of the MOOSE class this object belongs to. The object-oriented design of MOOSE enables you to work with an element as if it was an instance of its superclass. You can wrap any element in a `Neutral` object but of course later you may need to find out the actual MOOSE class it belongs to. `className` is the way to go.

  What about the `__class__` property in Python? The reason for having a separate `className` field is that MOOSE has its own system for class hierarchy. This is not necessarily visible to Python. Similarly, if you extend a MOOSE class in Python, it does not reach the underlying MOOSE class system. Extending MOOSE classes in Python is only useful for attaching additional information to it, but not for changing the underlying behaviour. For that you have to edit the C++ source code of MOOSE and recompile it, a task suited for the brave.

'`name`'
: Name of the element. Two different objects may have the same name but siblings in the element tree should have different names. You can change the name of an object by assigning a new string value to this field.

'`index`'
: MOOSE has two flavours of element: `simple element` and `array element`. A single entity is represented by a `simple element` whereas an `array element` represents a bunch of elements of the same kind. The `index` field indicates the position of this `Neutral` object in an `array element`. For `simple element`s it is `0`.

'`parent`'
: The `Id` of the parent element of this object in the element tree.

'`node`'
: This is the CPU node no. of on which this element is located. This is relevant only for parallel computers and PyMOOSE is yet to be adapted to such systems.

'`fieldList`'
: Vector listing the fields in the MOOSE object. You can traverse it like a Python `list`. If you add a Python attribute to the object later, that will not be visible in this. Nor will that visible to MOOSE. The way to add a field to the underlying MOOSE object is to use the `addField` function of the `PyMooseContext`.

'`cpu`'
: Reports the cost of one clock tick, very roughly # of FLOPs.

'`dataMem`'
: Memory used by data part of object

'`msgMem`'
: Memory used by messaging (Element) part of object.

'`childList`'
: Vector of `Id`s of the children of this object. This is also available via `children()` method, which is retained for backward compatibility.

5.5 PyMooseContext

One global object that provides access to global functions in PyMOOSE is the instance of PyMooseContext. It is available from the abstract base class of all PyMOOSE classes, PyMooseBase (which, being abstract, cannot be instantiated, but whose public properties are inherited throughout the class hierarchy).

PyMooseContext should be considered a singleton. There should be only one instance of it at any given time. And usually this instamce is created at startup (when moose is imported for the first time).

```
The standard way to access the context is:
context = PyMooseBase.getContext()
```

The context object provides quite low level access to MOOSE. So you should be discriminate about using the functions available. There are some functions to control simulation which are very common and must accessed via the context object.

'`setClock(clockNo, dt, stage=0)`'
: set the clock no. specified by `clockNo` with time step `dt` and stage `stage`.

'`useClock(clockNo, path, func='process')`'
: use clock specified by `clockNo` on `path`, which can be a wildcard path, calling the internal function `func` of the element class at each time step. The internal function is by default 'process', which calculates the state of the object at the end of the timestep.

'`reset()`'
: reset the simulation. This clears all recording Table objects that have been scheduled on some clock, and usually sets things back to initial values.

'`step(time)`'
: Run the simulation. If `time` is a `float`, it is taken as the length of time to be simulated. Each clock in the simulation proceeds by its `dt` until its `currentTime` reaches or exceeds `time`.

  If `time` is an integer, then it is taken as the number of steps for the fastest clock in the system. Thus, if the fastest clock in the system has `dt = 0.01` and `time = 10`, then this will run the simulation for 0.1 time units.

'`readCell`'
'`readSBML,`'
'`readNeuroML,`'
'`getCurrentTime,`'
'`addField.`'
'`loadG`'

'`runG`'

'`move`'

'`exists`'

The following are more advanced methods and should be used with caution. Usually for each of them there is a more accessible counterpart as a member of the PyMooseBase object or its derivative. The methods in PyMooseContext are required only when there is no such counterpart.

'`getField`'
'`setField`'
'`getParent`'
'`getPath`'

'`getName`'

‘getChildren’
‘copy’

5.6 Messaging

5.6.1 Source and Destination fields

Elements in a MOOSE model communicate with each other at runtime via messages. For example, to set up a constant current injection into a neuronal compartment, you want to connect a `PulseGen` object `pulsegen` to the `Compartment` object `comp`. In the definition of `PulseGen` class you have predefined source fields which tell what are the fields that can be transmitted out of the `pulsegen` object. Similarly, `Compartment` has a destination field where it can receive a current injection value. By connecting the source and the destination fields we set up communication between these two components.

The syntax for connecting source and destination messages is simple: `{source-object}.connect({source-field}, {target-object}, {target-field})`

With the current example, it will be:

```
pulsegen.connect('outputSrc', comp, 'injectMsg')
```

A very common yet confusing scenario is when you want to record the value of some state variable periodically. This is achieved by using `Table` objects. The confusing bit here is that the table object is used as a source, not a destination. The relevant source-field is `inputRequest` and the destination-field is the field to be recorded. This is a special case as the fields to be recorded are not destination fields, but value fields, which do not usually participate in messaging. Also, to use the Table object as an output buffer, you have to set the stepMode field to `TAB_BUF`.

Thus, to record the membrane potential Vm from our compartment, we can issue the following statements:

```
table = moose.Table('/Vm_tbl')
table.stepMode = 3
table.connect('inputRequest', comp, 'Vm')
```

5.7 Running GENESIS scripts in Python

There are two commands in PyMOOSE that allows you to run legacy GENESIS commands inside Python. These are `loadG` and `runG`. If you have a file `script.g` containing a GENESIS script, then you can execute the contents of it by invoking:

```
context.loadG(script.g)
```

where context is the singleton PyMooseContext object (can be obtained via: `context = moose.PyMooseBase.getContext()`).

Sometimes it is useful to access the MOOSE objects via the GENESIS commands. This is because for some classes, the Python wrappers may not be up to date, or it may just be ease of use. In those cases, just to carry out a single command, you can use `context.runG('{genesis-command}')`.

For example, to display the messages on an element specified by the path `/my_element`, you can use:

```
context.runG('showmsg /my_element')
```

6 Backward Compatibility with GENESIS

MOOSE is backward compatible with GENESIS 2. You can run many GENESIS scripts in MOOSE and there are ways to use GENESIS scripts and GENESIS commands from Python via PyMOOSE.

6.1 Loading a GENESIS script

You can load (and run) and existing GENESIS script using the loadG command in pymoose.

```
moose.context.loadG(path_to_your_genesis_script)
```

Here path_to_your_genesis_script is a python string with the file-path to be loaded.

If the GENESIS script has something that MOOSE is not compatibile with, then you will get error messages. the loadG function passes the file to the GENESIS parser in MOOSE and thus, whatever is there in the script gets exeeceuted outside the control of pymoose. Thus, if your GENESIS script has `reset` and `step` commands, then the simulation will have to be finished before control returns to the Python process.

6.1.1 Accessing elements created in a GENESIS script

Whatever elements are created in a GENESIS script, are available in memory, but not directly visible in Python. In order to access such elements, you need to create reference to the elements.

In GENESIS, every element is uniquely identified by its path, which similar to UNIX filesystem paths. You can use this path string to create a reference to the element in PyMOOSE.

```
Suppose you have created a table in your GENESIS script as follows:

// FILE: my_script.g
// ... ...
create table /vm_table
setfield /vm_table step_mode 3
// ... ...


Then you have loaded my_script.g in moose:

moose.context.loadG('my_script.g')



Now, you can access the table as follows:

my_table = moose.Table('/vm_table')


And the field values you set in the GENESIS script remain intact:
```

```
print my_table.stepMode
>>> 3
```

6.2 Running a GENESIS command

Sometimes it happens that some functions available in the GENESIS parser are not yet ported to PyMOOSE, or you may just find it more convenient to use GENESIS commands. In such cases, you directly run the GENESIS commands using `runG` function of PyMooseContext. You have to pass the GENESIS command string as parameter to the function.

```
moose.context.runG('setfield /vm_table xdivs 1000 xmin -1.0 xmax 1.0')
```

6.3 PyMOOSE equivalents of GENESIS classes

Many GENESIS classes have been renamed in PyMOOSE. We have tried to follow CamelCase for class names in PyMOOSE. For fields and method names, we use camelCase starting with lowercase letter.

The following table gives a list of MOOSE classes replacing GENESIS classes. The ones followed by an asterix (*) are not exactly compatible but can be used with a little bit of modification discussed later. If there is a '-' in the MOOSE column, that means the class has not been implemented in MOOSE yet.

GENESIS	MOOSE
Ca_concen	*CaConc*
Kpores	-
Mg_block	*Mg_block*
neutral	*Neutral*
PID	*PIDController*
RC	*RC*
asc_file	*AscFile*
autocorr	-
calculator	-
compartment	*Compartment*
concchan	-
concpool	-
crosscorr	-
ddsyn	-
dif2buffer	-

difbuffer	-
diffamp	*DiffAmp*
difshell	*DifShell*
dis_kin	-
disk_out	-
diskio	-
efield	-
enz	*Enzyme*
event_tofile	-
facsynchan	-
fixbuffer	-
freqmonitor	-
funcgen	*Table**
fura2	-
ghk	*GHK*
hebbsynchan	-
hh_channel	*HHChannel**
hillpump	-
hsolve	*HSolve*
interspike	-
leakage	*Leakage*

TO BE COMPLETED

7 Frequently Asked Questions

7.1 How do I load a model?

There are various formats that a model can be in. MOOSE recognizes GENESIS cell prototypes ('.p' files), SBML and neuroML files. Moreover, a GENESIS script ('.g' file) can be loaded in PyMOOSE. Here is a brief description of these:

7.1.1 How do I load GENESIS prototype file?

There is a text file format used by GENESIS to define prototypes for neuronal models. The details of this format can be found in GENESIS documentation for `readcell`.

Briefly, the main content of files in this format is a sequence of rows corresponding to compartments that constitute the cell model. In each row, you have the compartment name, its parent compartment's name, position of the starting point the compartment (optionally the end point), diameter of the compartment (the spatial dimensions are in microns for position and diameter), followed by a list of channels and their conductance densities(in SI unit, Siemens/m^2).

The channels are recognized by their names and the actual channel prototype definitions must be loaded under `/lib` before loading the prototype file. Otherwise that channel will not be inserted on the compartments.

Once you have created prototypes for the channels, the `readCell` function in `PyMooseContext` can be called to load the cell prototype. For instance,

```
moose.context.readCell('/usr/share/doc/moose/DEMOS/axon/axon.p', '/axon')
```

will load read the prototype described in the file `/usr/share/doc/moose/DEMOS/axon/axon.p` as the cell `/axon`.

7.1.2 How do I load an SBML file?

SBML stands for systems biology markup language. This is an XML based file format for defining computational models in systems biology. More information can be found at the official website: http://sbml.org/.

You can load SBML models using the `readSBML(filepath, elementpath)` function in `PyMooseContext` class.

For example,

```
moose.context.readSBML('/usr/share/moose/DEMOS/sbml_Reader/acc88.xml', '/acc88')
```

will load the model defined in acc88.xml file under a container `/acc88`.

7.1.3 How do I load a neuroML file?

neuroML is an XML-based format for defining neuronal models. You can find out more about neuroML at its website: http://www.neuroml.org/. MOOSE is capable of reading some levels of neuroML model definition. You can use the `readNeuroML` function in PyMooseContext to load a neuroML model in MOOSE:

```
moose.context.readNeuroML('/usr/share/moose/DEMOS/NeuroML_Reader/CA1/Ca1.xml', '/ca1')
```

to load the model defined in Ca1.xml into a Cell object '/ca1'.

7.2 How do I record field 'xyz' from and object?

The `Table` class serves for recording data in MOOSE. `Table` is a multipurpose class, capable of acting as an interpolation table, function generator as well as a data recorder.

For recording the value of fields that represent continuous variables, during a simulation, you have to connect the `inputRequest` field of a `Table` object to the corresponding value field on the target object, and set the step mode of the table object to `TAB_BUF` (=3).

```
xyz_table = moose.Table('/data/xyz_table')
xyz_table.stepMode = moose.TAB_BUF
xyz_table.connect('inputRequest', target_object, 'xyz')
```

7.2.1 How do I record spike events?

If you only want to record some threshold crossing events, like neuronal spikes, then you can use `TAB_SPIKE` (=4) mode. In this case, the `stepSize` field of the table is used as the threshold for the target field to be recorded.

```
spike_table = moose.Table('/data/spike_table')
spike_table.step_mode = moose.TAB_SPIKE
spike_table.connect('inputRequest', soma, 'Vm')
spike_table.stepSize = 0.1
```

In the above example, soma is a neuronal compartment (instance of `Compartment` class). `spike_table` will have an entry of the current simulated time every time membrane potential `Vm` of `soma` crosses 0.1 Volt (assuming you are using SI system through out).

7.3 Can I do XYZ available in GENESIS using PyMOOSE?

See Chapter 6 [Backward Compatibility], page 21.

Appendix A GNU Free Documentation License.

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released

under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and

that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called

an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C)  year  your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts.  A copy of the license is included in the section entitled ``GNU
Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with. . . Texts." line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index

Variable Index

(Index is nonexistent)

Concept Index