Chapter 16
Automating Cell Construction with the Cell Reader

DAVID BEEMAN

16.1 Introduction

In the previous chapter, we created a simple multi-compartmental neuron with a dendrite compartment, a soma, and an axon. The dendrite contained a synaptically activated channel and the soma contained voltage-activated Hodgkin-Huxley sodium and potassium channels. The script *tutorial4.g*, listed in Appendix B, contains the function definitions and commands used to construct this neuron, provide synaptic input to the cell, and plot the results of the simulation. In this tutorial, we will modify the script in order to construct the same neuron from a data file, using the GENESIS cell reader. The cell reader, which is implemented in the *readcell* command, allows one to build multi-compartmental neurons by reading cell parameters from a *cell descriptor file*. This file contains the names and dimensions of the compartments that will be used in the cell, along with the names of the channels and other elements that are contained within each compartment. The cell reader then uses this information to put together a cell and to establish the necessary messages between the various elements. This can significantly reduce the effort needed to construct a complex cell with many compartments and channels.

Before reading further, you may wish to review Chapter 15 and the listing of *tutorial4.g* in order to remind yourself of the purpose of the various functions that were defined in the simulation script. In this tutorial, we will use seven scripts to create the simulation. The following section describes the use of a script that we will write (*protodefs.g*) in order to create a library of the basic components to be used in building our cell. This script may use *include* to bring in other scripts that define the particular channels or other elements which we need. In our case, we will use the *hhchan.g* script that we used in the previous chapters, plus three other scripts from the *neurokit/prototypes* directory (*compartments.g, synchans.g* and *protospike.g*). Then, we describe the listing of the cell descriptor file *cell.p*. Finally, we will construct a main simulation script *tutorial5.g*. This will create the various XODUS elements needed to control and display the simulation, include *protodefs.g* in order to create the raw materials for building the cell, process *cell.p* with *readcell* to create the cell, and create any needed external objects and their messages in order to interact with the cell.

16.2 Creating a Library of Prototype Elements

The cell reader builds the cell by making copies of "prototypes" of the various elements that will be used, replacing the default values of parameter fields with values taken from the cell descriptor file. For example, when constructing a soma with several attached dendrite compartments, it will make multiple copies of a generic compartment prototype and then set the data fields in each compartment to the appropriate values. Likewise, a cell having Hodgkin-Huxley Na channels in several compartments will get these channels from copies of the prototype. These channels will be identical, but the cell reader will provide the right value of the maximum channel conductance *Gbar* for each copy.

The cell reader expects to find this library of prototype elements as a set of subelements of the **neutral** element */library*. Thus, we need to write a script that will create */library* and fill it with a prototype compartment, one copy of each of the different channel types we will use, and a spike generator. Although the statements that are needed to set up the prototype library could go into your main simulation script, it is customary to make a separate script for this, and to then use *include* to bring it into the simulation. Following tradition, we will call this script *protodefs.g*, although you may give it any name you like.

16.2.1 Future Changes in the Cell Reader

The cell reader is continually evolving in order to provide more flexibility. In future releases of GENESIS, more options will be available for the cell descriptor file and new types of GENESIS objects may be used as components for the construction of cells. In order to be aware of these new developments, you should consult the current version of the *readcell* documentation in the GENESIS Reference Manual.

It is likely that future versions of the cell reader will construct cells from extended objects instead of the */library* prototype elements we have used here. As we have demonstrated in Sec. 14.5, extended objects can take over many of the duties that are performed by script commands or by the cell reader, such as establishing messages with their parent compartments and scaling various fields according to the compartment dimensions. This will mean some changes in the way that prototypes are specified in the *protodefs.g* file. The scripts that are included will typically create extended objects for prototype compartments, channels and other cell components, rather than providing functions that create elements from the basic predefined GENESIS objects. As these changes are incorporated into future releases of GENESIS, they will be accompanied by documentation and example scripts.

Instead of directly creating prototype elements from the basic **compartment**, **synchan** and **spikegen** objects, we will create them from functions defined in the scripts *compartments.g*, *synchans.g* and *protospike.g* in the *neurokit/prototypes* directory. Under GENESIS version 2, this is not strictly necessary. However, by including these scripts, we can maintain compatibility with future releases of GENESIS that may have modified versions of these scripts to go along with changes in *readcell*.

16.2.2 The *protodefs.g* Script

We begin by including the script *compartments.g*, which can create a variety of compartments. As in *tutorial4.g*, we can include the file *hhchan.g* in order to provide the functions to create the prototype Hodgkin-Huxley channels *Na_squid_hh* and *K_squid_hh*. We will also include the scripts *synchans.g*, which contains a function to create a glutamate-activated excitatory channel like the *Ex_channel* element we have used before, and *protospike.g*, which can create a **spikegen** element. If the SIMPATH in your *.simrc* file is properly set to include the *Scripts/neurokit/prototypes* directory, these files need not exist in your own directory. After including the *hhchan.g* file, be sure to replace its values for *EREST_ACT*, *ENA*, and *EK* with your own, as we did in *tutorial4.g*. As the cell reader will calculate the compartment area from dimensions given in the cell descriptor file, you will not need to specify these quantities here. At this stage, your *protodefs.g* might look something like

```
    include compartments
    include hhchan

    EREST_ACT = -0.07
    ENA   = 0.045
    EK    = -0.082

    include synchans
    include protospike
```

Of course, your version will contain many illuminating comments!

After the relevant scripts have been included, we can create the library with the statements

```
  create neutral /library
```

```
disable /library
pushe /library    // Make these elements in the library

make_cylind_compartment         /* makes "compartment" */

// Create prototype H-H channels "Na_squid_hh" and "K_squid_hh"
make_Na_squid_hh
make_K_squid_hh

// Make a prototype excitatory channel, "Ex_channel"
make_Ex_channel     /* synchan with Ek = 0.045, tau1 = tau2 = 3 msec */

make_spike      /* Make a spike generator element "spike"*/

pope    // Return to the previous working element
```

You may notice that we have used a new GENESIS command, *disable*. We don't want to waste time having the elements in the library calculate anything, since they exist only to be copied into the elements that will actually be used in the simulation. When an element is disabled, the simulator will not attempt to update the fields of it or any of the child elements in its hierarchy. It may be re-enabled with the *enable* command.

The function *make_cylind_compartment* not only makes the element *compartment*, but it sets the default values of its fields and adds a new field *Shape*, which is initialized to "`cylinder`". As with *hhchan.g*, the script *synchans.g* uses global variables for the channel equilibrium potentials. The value for the glutamate excitatory channel *Ex_channel* is set with the statement "`EGlu = 0.045`". As this is the same as the value we used for the excitatory channel in Chapter 15, we can leave it as is. However, the two channel time constants, *tau1* and *tau2*, are "hard-coded" at 3 *msec* within the *make_Ex_channel* function. Fortunately, these are also the values that we want. If we were to require different values, we could set them to the correct values right after we create the channel. Finally, we need to create a spike generator, which will be linked to the soma by the cell reader. As the *make_spike* function in *protospike.g* creates exactly what we want (an element named *spike* with a unit amplitude and an absolute refractory period of 10 *msec*), we will use it here.

Before going on to construct the rest of the simulation, you may wish to test your *protodefs.g* file by executing it as a GENESIS script. If you have made no errors, it should execute without much happening on the screen. However, you may use the *le* and *showfield* commands to satisfy yourself that the proper prototype elements have been created in the library.

16.3 The Format of the Cell Descriptor File

Our cell will be assembled from the prototype elements in the library according to the specifications in the cell descriptor file. For our simulation we will use the example file *cell.p*, shown in Fig. 16.1. (This is often referred to as a *cell parameter file*, hence the required extension, "*.p*".) We can best understand the format of this file by going through the file line by line.

```
// cell.p - Cell parameter file used in Tutorial #5

//    Format of file :
// x,y,z,dia are in microns, all other units are SI
// In polar mode 'r' is in microns, theta and phi in degrees
// Control line options start with a '*'
// The format for each compartment parameter line is :
//name  parent  r  theta  phi  d  ch  dens  ...
//in polar mode, and in cartesian mode :
//name  parent  x  y  z  d  ch  dens  ...
// For channels, "dens" =  max conductance per compartment unit area
// For spike elements, "dens" is the spike threshold
// Coordinate mode
*relative
*cartesian
*asymmetric

// Specifying constants
*set_compt_param    RM    0.33333
*set_compt_param    RA    0.3
*set_compt_param    CM    0.01
*set_compt_param    EREST_ACT    -0.07

// For soma, use the leakage potential (-0.07 + 0.0106) for EREST_ACT
*set_compt_param    ELEAK    -0.0594
soma  none  30  0  0  30  Na_squid_hh 1200  K_squid_hh 360  spike 0.0

*set_compt_param    ELEAK    -0.07
dend  soma 100  0  0   2  Ex_channel 0.795775
```

Figure 16.1 The cell descriptor fi le for the simple neuron model.

When cells are built "by hand," the [*x*, *y*, *z*] coordinate fields of the compartments are normally not used, except for graphical display of the cell geometry, or when the coordinates are needed to establish a pattern of connections in a network. However, the cell reader uses these coordinates in order to determine the lengths of the compartments, and they are NOT optional. For an asymmetric compartment like that shown in Fig. 2.3, these coordinates are measured at the end that has the potential *Vm*. Using the option "`*relative`" allows one to specify these coordinates relative to the parent compartment without keeping track of the absolute location relative to the origin of the cell. As we will be using Cartesian coordinates, the option "`*cartesian`" is also specified. The "`*asymmetric`" option need not have been specified, as it is the default. It tells the cell reader that the various compartments will be constructed from copies of the asymmetric compartment in the library named *compartment*. If we had specified "`*symmetric`", the compartments would have been constructed from the symmetric compartment prototype *symcompartment*.

Next, the values of the variables *RM*, *RA*, *CM*, and *EREST_ACT* are set, using the "`*set_compt_param`" option. These are internal variables that will be used by *readcell* for values of the specific membrane resistance and capacitance, specific axial resistance, and resting potential of the various compartments. The cell reader will use the values of *RM*, *RA* and *CM* to calculate and set the *Rm*, *Ra* and *Cm* fields from the compartment dimensions, ignoring any initial values set in the library prototype. Likewise, the *Em* and *initVm* fields are set from the *EREST_ACT* variable. However, we would like to set the soma *Em* voltage to the leakage potential rather than to the resting potential, as we have done in the previous tutorials. This is done by setting another internal variable *ELEAK* to $-0.0594\ V$. After the soma is created, *ELEAK* is set to -0.07, so that the dendrite *Em* will be given the proper resting potential. (Recall that the different value of *Em* was needed in the soma to compensate for the current flow through the Hodgkin-Huxley channels at the resting potential.)

A similar option, "`*set_global`" has the same effect on the internal compartment parameter variables as "`*set_compt_param`", but also changes the value of the global script variables of the same name, which must have been previously declared. In addition, "`*set_global`" may be used to set the values of other previously declared global script variables that are not directly used by *readcell*. In general, it is best to use the option "`*set_compt_param`" in order to avoid the use of unnecessary global variables.

The line describing the soma compartment starts out by giving its name ("`soma`") and its parent compartment ("`none`"). When reading the documentation for the cell reader, note that the terms "parent" and "children" are used somewhat differently than we have used them so far. If we have elements */cell/soma* and */cell/dend*, we would normally refer to the siblings *soma* and *dend* as children of the parent */cell*. When this same hierarchy of elements is created with the cell reader, the documentation refers to the dendrite as a child of the parent soma, because it is connected to the soma through the dendrite axial resistance. Thus, the line describing the dendrite (*dend*) lists *soma* as its parent, even though it will create */cell/dend*, not */cell/soma/dend*. To avoid confusion, we refer to *dend* as a *subelement* of */cell* whenever there is any ambiguity.

The soma that we created in the previous tutorials was 30 *μm* long and 30 *μm* in diameter. If we make our cell lie along the *x*-axis, that means that we should give (*x*, *y*, *z*, *d*) as (30, 0, 0, 30). After the coordinates and diameter of the compartment, we list the channels, giving the name of the prototype channel followed by the parameter "*dens*". For a channel, this parameter is the maximum conductance per unit area of the compartment. For example, the function *make_Na_squid_hh*, defined in *hhchan.g*, sets the channel field *Gbar* to "`1.2e3 * SOMA_A`" siemens. In our previous simulation we were forced to use this "density" value of 1200, and had to explicitly override the default value of *SOMA_A* with our own value. The cell reader provides us with more flexibility, however. The density of 1200 is stated explicitly in the "*.p*" file, after the name of the channel, and the compartment dimensions are used with this parameter to set the value of *Gbar* when the channel is created from its prototype in the library. Thus, we need say nothing about *SOMA_A* in the *protodefs.g* file.

The spike generator is listed in the same manner as the channels. However, the *dens* parameter is used to give the spike threshold. The cell reader will then create the element */cell/soma/spike*, set its *thresh* field to 0.0, and add the INPUT message from the soma. The *abs_refract* and *output_amp* fields of the **spikegen** object are not set from the cell descriptor file. They will take on the values assigned to the prototype, unless these fields are explicitly set after the cell has been built. As in the previous chapter, we have used values that will produce a minimum interval (refractory period) of 10 *msec* between the spikes and give them a unit amplitude.

Next, we change the value of *ELEAK* to the appropriate value for the dendrite compartment. If we had wished to use different values for the specific resistances and capacitance, these parameters could be changed here. Then we give the name of the compartment (*dend*) its "parent" (*soma*), along with its coordinates, diameter and any of its subelements that need to be linked with messages. The geometrical parameters for the dendrite should be (100, 0, 0, 2) if it is to be 100 *μm* long and 2 *μm* in diameter. If we were using absolute coordinates instead of relative coordinates, we would have given the dendrite x-coordinate as 130.

In the previous tutorial, we gave the synaptically activated dendrite channel a maximum conductance of 5.0×10^{-10} siemen. Whether or not the *gmax* field of the prototype channel *Ex_channel* was set to this value, the cell reader will set the field of */cell/dend/Ex_channel* to a value calculated from the channel density parameter given here. You should verify that for the compartment dimensions used here, this results in a density of 0.795775 S/m^2, as given in *cell.p*

The GENESIS Reference Manual and the help file for the *readcell* command describe several other useful options. The "`*compt`" option is particularly useful for large models. A detailed cell model may use many compartments that are identical, except for their dimensions. These may often contain the same conductances, with the same conductance densities. In order to avoid the repetition of many identical long strings of channel specifications in the cell descriptor file, you may use the "`*compt`" option followed by the name of a compartment in the library. Rather than being a "naked" compartment like the

/library/compartment element which we have used, this would typically consist of a compartment and its associated subtree of channels and other subelements, linked with the appropriate messages. Once this option is specified, all following compartments will be copies of this element tree, with their channel conductances appropriately scaled to the dimensions of the compartment.

16.4 Modifying the Main Script to Use the Cell Reader

By making use of the two files *cell.p* and *protodefs.g*, we can eliminate much of the complexity of the *tutorial4.g* script. Copy this file into a new one with a different name and carry out the changes described below.

The cell parameters are either determined from the prototype definitions in *protodefs.g* or are read in from the parameter file, *cell.p*. Therefore all of the statements preceding the function definitions, except for those that set *tmax*, *dt* and *gmax* should be removed and replaced by the statement "`include protodefs`". Likewise, the function definitions for *makecompartment*, *makechannel*, and *makeneuron* should be removed. The only function definitions remaining will then be those that involve the interaction of the cell with the outside world, the feedback connection and the graphics functions.

In the main script, the call to *makeneuron* should be removed and replaced by the statement

```
readcell cell.p /cell
```

Here, the two arguments specify the name of the cell descriptor file to be read and the path name of the resulting cell. As in the previous chapters, */cell* will be created from a **neutral** object.

At this point, you should have a script that is functionally equivalent to *tutorial4.g*. Try it out and verify that it works the same as *tutorial4.g*.

Using the cell reader to build this simple cell hasn't saved us all that much work, as we still had to create all the prototype channels in the library and to provide the graphical interface. For a more complicated multi-compartmental cell, the effort saved could be considerable. This is because the same prototypes can be used in many compartments. In addition, the use of cell descriptor files provides a convenient mechanism for the exchange of cell models with other users of GENESIS.

16.5 The Neurokit Simulation

In this simulation, we still had to write a fair amount of GENESIS script language code in order to provide for the display of our results. The *Neurokit* simulation, which is found in the GENESIS *Scripts/neurokit* directory, takes us a step further by providing a graphical interface to the cell reader. This makes it possible to build complex multi-compartmental neurons and to run elaborate single cell simulations without the necessity of writing GENESIS scripts. The *README* file that accompanies the simulation gives detailed instructions for using *Neurokit*. The next tutorial in this series shows how to implement this simulation within the *Neurokit* environment. In Chapter 19, we will learn how *Neurokit* and the cell reader are used with calcium concentration-dependent channels.

The subdirectory *Scripts/neurokit/prototypes* contains a large number of scripts, similar in format to *hhchan.g*, for creating a wide variety of channels. As with *hhchan.g*, these may be used independently of *Neurokit*. In order to encourage the exchange of new GENESIS simulation components, it is strongly recommended that you follow the conventions used in these scripts. The complete list of available channels may be found in the file *LIST* in this directory. In order to easily include these prototype scripts in your own simulations, include a statement like

```
setenv SIMPATH {getenv SIMPATH} /usr/genesis/Scripts/neurokit/prototypes
```

in either your *.simrc* file or your simulation scripts.

16.6 Exercises

1. In Chapter 7, we used *Neurokit* to experiment with a model of a burst firing molluscan neuron. The cell descriptor file *mollusc.p* in the *Scripts/burster* directory was used to create this cell. The file *ASTchan_tut.g* provides the functions needed to create the prototype channels and the file *userprefs.g* contains some statements that you can include in your *protodefs.g* file. (The statements at the end of *userprefs.g* are for use with *Neurokit*, and are described in the next chapter.) With these files and a modified version of *tutorial3.g*, duplicate the endogenous bursting behavior described in Sec. 7.5.2.

2. As a variation of the previous exercise, write a simple simulation to observe the soma membrane potential of the Traub model CA3 cell, with a current injection of 0.2 *nA* to the soma. Experiments with this model are also described in Chapter 7. The *Scripts/traub91* directory contains the files *CA3.p*, *traub91proto.g* and *userprefs.g*, which you may incorporate into your simulation. Your results should agree with those shown in the lower plot of Fig. 7.3.