Chapter 14
Adding Voltage-Activated Channels

DAVID BEEMAN

14.1 Review

In the previous tutorial, we created a neuron soma compartment having the same physiological properties as those of the squid giant axon studied by Hodgkin and Huxley (1952d). In this tutorial, we will add voltage activated sodium and potassium channels to the soma by modifying a copy of the script that you produced in the previous tutorial. Before continuing with this tutorial, you may wish to review the discussion of the Hodgkin-Huxley channel models in Chapter 4. In other tutorials, we will build upon this to produce multicompartmental models of neurons and networks of these neurons. Your script from the previous tutorial should look something like the listing of *tutorial2.g* in Appendix B.

A number of conventions have been used in this example script in order to illustrate some principles of good GENESIS programming style. Any declarations and assignments of global variables are performed at the beginning of the script, followed by function definitions, and then the main body of the script. Although functions to create XODUS forms and their associated widgets often tend to be rather specific, it is best to make generally applicable functions whenever possible, as we have done with the *makecompartment* function. If a variable is not likely to be used outside a particular function, it should be declared locally to the function. Of course, the liberal use of comments will make it easier for you or others to understand your scripts. Your own script is unlikely to be exactly like this example. However, if it has diverged too much from this general style, now might be a good time for a rewrite and "cleanup." During the course of development of a simulation, hindsight can

be valuable. It is best to frequently go back and rewrite early code, rather than continuing to build upon programs that could have been written more cleanly.

14.2 More Fun With XODUS

In previous chapters, we have learned how to use three XODUS graphical objects, or *widgets* that are available within GENESIS. These objects (**xform**, **xgraph** and **xbutton**) and others are well documented in the GENESIS Reference Manual. As you develop more sophisticated simulations and need to exercise more control over the placement, size and labeling of graphical elements, you will want to read about the many options that can be set when these elements are created.

For example, the default title appearing on the graph (`graph`) may be changed by using the `-title` option when the graph is created. The labels on the buttons were set by default to the name of the **xbutton** element. Usually you can choose an appropriate name to correspond to the label you want, but you may gain additional flexibility by setting the *onlabel* and *offlabel* fields. Typing "`showobject xbutton | more`" will give you a list of the fields that you can set to modify the appearance of the buttons. You may also wish to change the size and location of the form and the elements contained within it. For example, the `RESET` and `RUN` buttons could be placed side by side. It would also be a good idea to isolate the control buttons to a "control" form like those appearing in many of the simulations we used in Part I of the book. If we remove the control buttons from the *Vmgraph* form, the *make_Vmgraph* function will be more versatile and can be used in other simulations that we create.

The following statements illustrate some ways in which we could change the layout of a form called */control* that contains the control buttons we have used so far.

```
function make_control
    create xform /control [10,50,250,145]
    create xlabel /control/label -hgeom 50 -bg cyan -label "CONTROL PANEL"
    create xbutton /control/RESET -wgeom 33%       -script reset
    create xbutton /control/RUN  -xgeom 0:RESET -ygeom 0:label -wgeom 33% \
         -script step_tmax
    create xbutton /control/QUIT -xgeom 0:RUN -ygeom 0:label -wgeom 34% \
        -script quit
    xshow /control
end
```

The horizontal and vertical positions and the width and height of an XODUS widget are specified by the four *geometry* fields *xgeom*, *ygeom*, *wgeom* and *hgeom*. These fields may be set at any time with the *setfield* command. You may also set them when the widget is created, by using the options `-xgeom`, `-ygeom`, `-wgeom` and `-hgeom`. The first line

illustrates a shorthand notation for specifying these four options. The four numbers in the square brackets represent the four geometry fields for the form, measured in pixels. For **xform** elements, positions are measured relative to the upper left-hand corner of the screen. In this case, we've given the form some extra height to save room for widgets that we might want to add later. You may want to experiment a bit to find the best size for your forms. One way to do this is to use the mouse to resize a form that you have created (usually by clicking and dragging the mouse on the icon at the right end of the form's title bar). Then use *showfield* to find the resulting geometry.

For other widgets, positions are measured relative to the upper left-hand corner of the form that contains them. If these aren't specified, they are set by default to be at the left-hand side of the parent form and just below the previously created widget. The default width *wgeom* is the width of the parent form. Each type of widget has its own default height *hgeom*.

The second line creates another type of widget, the **xlabel**. It is simply a rectangular box containing a string of text. It performs no action when you click on it. As only the *hgeom* field was specified when the label was created, it will have the width of the form and will be just below the top of the form. It will be 50 pixels high and will have the label "`CONTROL PANEL`". The option "`-bg cyan`" sets the background of the label to a color that will clash nicely with the default colors of the other widgets. You might want to experiment with other color schemes! You may do this interactively for the various widgets by using *setfield* to set the *bg* field.

A percent sign may be used after a number to indicate that it represents a percentage of the screen or form dimension. This is used to create a `RESET` button that is 33% of the width of the form. The statement that creates the `RUN` button illustrates another way to specify a position. The notation "`-xgeom 0:RESET`" indicates that the button is to be placed 0 pixels to the right of the `RESET` button. The option "`-ygeom 0:label`" means that it will be placed just below the label. What would happen if this specification were omitted? Note that as we suggested in the previous chapter, we have used a *step_tmax* function to run the simulation for a time *tmax*.

After removing the lines in *make_Vmgraph* that create these buttons and adding some extra "bells and whistles," your function might look like

```
function make_Vmgraph
    float vmin = -0.100
    float vmax = 0.05
    create xform /data [260,50,350,350]
    create xlabel /data/label -hgeom 10% \
        -label "Soma with Na and K Channels"
    create xgraph /data/voltage -hgeom 90% -title "Membrane Potential"
    setfield ^ XUnits sec YUnits Volts
    setfield ^ xmax {tmax} ymin {vmin} ymax {vmax}
```

```
    xshow /data
end
```

In this example, the default title has been replaced with a more descriptive one. In addition, the internal fields of the **xgraph** element, *XUnits* and *YUnits*, have been set to provide meaningful horizontal and vertical axis labels. Try modifying your simulation script to use these new *make_control* and *make_Vmgraph* functions and observe the effect that they have.

The GENESIS Reference Manual describes some other widget fields that you may set to specify colors, font sizes and alternate labels. Further examples of the use of XODUS widgets may be found in the various *forms.g* scripts used in the tutorial simulations from Part I of this book. The modularity of GENESIS makes it easy to "steal" useful functions or bits of code from these simulations. For example, the *Cable, Neuron,* and *MultiCell* simulations all contain a script *xtools.g*. This contains a useful function (*makegraphscale*) to pop up a menu for changing the scale of the graphs that are created in their *forms.g* scripts. The *Squid* simulation uses a slightly different function to accomplish the same thing.

If you prefer, you may leave these details of "prettifying your display" until after you have learned to add channels to the model. However, we should now spend a few minutes learning to use another useful XODUS object, the *dialog widget*. The following statement creates an element called *Injection* within the */control* form, using the **xdialog** object.

```
create xdialog /control/Injection -label "Injection (amperes)" \
            -value 0.3e-9 -script "set_inject <widget>"
```

This element consists of a label adjacent to a dialog box in which the user can edit or enter text. The `-label` option is followed by the string used for the label in place of the default, which would be the name of the element. The backslash "\" indicates a continuation of the *create* statement to another line. The argument of the `-value` option is the initial value that appears in the dialog box and is stored in the *value* field of the element. As with any other element, this field may be inspected and set with the *showfield* and *setfield* commands. In addition, the value may be edited within the dialog box. When the user hits "Return" while the cursor is within the dialog box, the *value* field is set. In addition, this causes the function used as the argument to `-script` to be executed. Here, *set_inject* is a function that we will define. (NOTE: it is a common mistake to change the contents of the dialog box without hitting "Return". In this case, the old value is still being used.)

The argument of `-script` in our dialog box example employs another useful bit of shorthand. The notation `<widget>` stands for the widget that is being defined. Thus,

```
-script  "set_inject <widget>"
```

is equivalent to

```
-script  "set_inject /control/Injection"
```

In our example, we would like to define a function that gets the variable stored in the *value* field of the **xdialog** element and uses it to set the *inject* field of our soma compartment. The following function takes the name of the **xdialog** element as its argument and uses the GENESIS *getfield* function to return the value of the *value* field.

```
function set_inject(dialog)
    str dialog
    setfield /cell/soma inject {getfield {dialog} value}
end
```

The *getfield* function is another useful command for dealing with the internal data fields of an element. It takes the name of the element and the name of the field as arguments and returns the value of the field. Thus, the statement

```
echo {getfield /control/Injection value}
```

would print out the value of the dialog box. Likewise,

```
set_inject /control/Injection
```

will use *getfield* to retrieve this value and *setfield* to set the *inject* field of */cell/soma*. If you make these changes to your script, you will now be able to change the soma injection current by using the dialog box.

14.3 Voltage-Activated Channel Objects

GENESIS provides several different types of objects for implementing voltage dependent ion channels. The **hh_channel** object, which we will use in this tutorial, provides the simplest way to implement the equations used by Hodgkin and Huxley to model sodium and potassium channels in the squid giant axon. Another object, the **vdep_channel**, may be linked to **vdep_gate** objects to create channels that have a more general form of the equations. This latter approach is somewhat slower because messages must be passed between the separate channel and gate objects. For maximum generality in channel modeling, one may use the **tabchannel** or **tab2Dchannel** object, or a combination of **tabgate** objects used with **vdep_channel** objects. The **tabchannel**, **tab2Dchannel** and **tabgate**, which are described in Chapter 19, allow the use of tables with interpolation rather than equations to represent the voltage dependencies. This makes it possible to model channels using experimental data that do not fit the Hodgkin-Huxley form. For serious modeling, we strongly recommend the use of **tabchannel** or **tab2Dchannel** objects. As we will see in Chapters 19 and 20, this is particularly important for accurate simulation of cells containing a large number of compartments. The GENESIS directory *Scripts/neurokit/prototypes* contains scripts that illustrate the use of all of these ways to implement voltage-dependent channels.

14.3.1 The hh_channel Object

In the Hodgkin-Huxley model, the general form for the channel conductance is represented as being proportional to an activation state variable raised to an integer power, times an inactivation state variable raised to another integer power. The **hh_channel** object calculates the channel conductance from the equation

$$Gk = Gbar \cdot X^{X\,power} Y^{Y\,power}. \tag{14.1}$$

The **hh_channel** fields in Eq. 14.1 correspond to the variables in Eqs. 4.8 and 4.9. In the usual Hodgkin-Huxley notation for the Na channel, the activation state variable m corresponds to our variable X, with $X\,power = 3$, and the inactivation variable h corresponds to Y with $Y\,power = 1$. The Hodgkin-Huxley K channel activation variable n is represented by X in our notation, with $X\,power = 4$. The K channel has no inactivation state variable, so we use $Y\,power = 0$. As we are using SI units, the constant of proportionality $Gbar$ ($\overline{g}$) should be expressed in siemens (1/ohms).

GENESIS uses the convention that a positive current represents a flow of positive charge into the compartment, so the current through the channel is given by

$$Ik = Gk(Ek - Vm). \tag{14.2}$$

Here, Vm is the membrane potential of the compartment that contains the channel, and the other variables used in these equations are the names of fields of the **hh_channel** object. These correspond to the subscripted variables in Eq. 4.2. The field Ek represents the value of the ionic equilibrium potential, which we will express in volts. Channel elements that are created from the **hh_channel** object calculate both X and Y by solving differential equations of the form

$$\frac{dX}{dt} = \alpha(1 - X) - \beta X, \tag{14.3}$$

corresponding to Eqs. 4.11–4.13. The voltage-dependent rate constants α and β can each assume one of the three functional forms:

FORM 1 (EXPONENTIAL):

$$\alpha(V_m) = A \exp(\frac{V_m - V_0}{B}) \tag{14.4}$$

FORM 2 (SIGMOID):

$$\alpha(V_m) = A/(\exp(\frac{V_m - V_0}{B}) + 1) \tag{14.5}$$

FORM 3 (LINOID):

$$\alpha(V_m) = A(V_m - V_0)/(\exp(\frac{V_m - V_0}{B}) - 1). \tag{14.6}$$

Note that Eqs. 4.23, 4.24 and 4.26-4.29, used by Hodgkin and Huxley to fit the K and Na channel rate constants, each fall into one of these three forms. The form to be used and the constants *A*, *B*, and V_0 are specified for each rate constant by setting the corresponding fields in the **hh_channel** element. These are:

(α for X): *X_alpha_FORM, X_alpha_A, X_alpha_B, X_alpha_V0*
(β for X): *X_beta_FORM, X_beta_A, X_beta_B, X_beta_V0*
(α for Y): *Y_alpha_FORM, Y_alpha_A, Y_alpha_B, Y_alpha_V0*
(β for Y): *Y_beta_FORM, Y_beta_A, Y_beta_B, Y_beta_V0*

14.3.2 Adding Hodgkin-Huxley Na and K Channels to the Soma

In order to simplify the process of setting all these internal fields, we will borrow a script from the *Neurokit* library of channel prototypes. The file *hhchan.g*, which is listed in Appendix B, defines two functions, *make_Na_squid_hh* and *make_K_squid_hh*. These create the two channels from **hh_channel** objects and set the internal fields to the proper values. In order to understand how we will use these functions, you should now take a look at the *hhchan.g* script.

The fields *X_alpha_FORM, X_beta_FORM*, etc. take integer values (1, 2 or 3) to specify which form of the rate constant equations to use. For clarity, it is more desirable to refer to these forms by name, as is done in *hhchan.g*. In order to allow this, *hhchan.g* contains the statements

```
int EXPONENTIAL =   1
int SIGMOID     =   2
int LINOID      =   3
```

at the beginning of the script. The contents of the *hhchan.g* script can be made part of our simulation by using the GENESIS *include* command "`include hhchan`" after these definitions. This script is found in the GENESIS *Scripts/neurokit/prototypes* directory. Normally, the *.simrc* file in your home directory will set the GENESIS environment variable "SIMPATH" so that the *prototypes* directory will be searched when a file is specified for inclusion. Thus, you may use this or one of the other channel "prototype" scripts without having a copy in the directory that contains your simulation scripts.

As with any modern computer language, you can make your GENESIS simulations much easier to understand by breaking them into modules that are combined by the use of *include*. For example, large GENESIS simulations often put the graphics functions together in a script called *forms.g*, so that it may be included if graphics are to be used, and may be omitted if the simulation is to be performed in the background with output to files. Although GENESIS makes no distinction between constants and variables, it is often useful to include

a file with a name like *constants.g* at the beginning of the main simulation script. This will contain assignments of global variables, such as ionic equilibrium potentials, which are not expected to change during the course of a simulation. The *MultiCell* demonstration in the *Scripts* directory illustrates this structure.

As our simulation script will be relatively short, we will only use the one included script *hhchan.g*, but will try to keep our global variables and constants together at the beginning of the script. As we add more graphics-related function definitions, it will be useful to keep them together in one part of the script, rather than interspersing them with non-graphics functions. However, looking at the listing for *hhchan.g*, we see that it makes a necessary exception to the guideline of keeping global variables together in one place.

After some initial comments, the script defines and initializes some variables (*EREST_ACT*, *ENA*, *EK*, and *SOMA_A*) for the membrane resting membrane potential, channel equilibrium potentials, and the area of the soma. These are needed by the following function definitions. As this script and the other channel prototype scripts were intended to be available for inclusion in many different simulations, it makes sense to define the needed "constants" in these scripts. In the case of *hhchan.g*, the values of these constants were chosen for a granule cell simulation using Hodgkin-Huxley channels, but with potentials that are slightly shifted from the squid potentials which we defined in the last tutorial. Thus, it is important to place the "`include hhchan`" statement before our own definitions of these constants, so that ours will prevail. It is also a good idea to put in comments that explain the necessary order of these statements.

We can understand how these constants and the *SOMA_A* constant are used by looking at the definition of the function *make_Na_squid_hh*, which follows some extended comments describing the **hh_channel** object. The function begins by checking for the existence of an element called *Na_squid_hh*. If this channel doesn't already exist, one is created and the internal fields are set with a lengthy *setfield* command that begins with

```
setfield Na_squid_hh \
        Ek              {ENA} \                 // V
        Gbar            { 1.2e3 * SOMA_A } \    // S
```

The field *Ek* is the "battery" that is in series with the sodium conductance. It will be set to the value (in volts) that we have defined for the sodium equilibrium potential, *ENA*. In their paper, Hodgkin and Huxley (1952d) fit *Gbar* to a value of 120 milli-mhos (*mS*) per square centimeter of membrane area. You should verify that if we calculate *SOMA_A* in square meters, the expression above will correspond to the Hodgkin-Huxley result in SI units. In your script, use the values that we have given for the soma dimensions, *soma_l* and *soma_d*, to calculate *SOMA_A* for our cylindrical soma compartment. In doing so, you are letting this single Na channel represent the composite behavior of the many sodium channels that would be found in a patch of membrane with area *SOMA_A*. If you have the

time (and inclination) you may wish to verify that the values used to set the remaining fields correspond to those used by Hodgkin and Huxley. You should note, however, that the sign convention used for voltages by Hodgkin and Huxley is reversed from our modern usage.

The function *make_K_squid_hh* works in a similar manner to create a potassium channel called *K_squid_hh*. These two channels will be created at the current position in the hierarchical element tree. For a convenient grouping of elements, we would like to refer to these channels as */cell/soma/Na_squid_hh* and */cell/soma/K_squid_hh*. The easiest way to accomplish this is to use the *pushe* and *pope* commands to temporarily make */cell/soma* the current working element. We can create the two channels in the proper location by putting the statements

```
pushe /cell/soma
make_Na_squid_hh
make_K_squid_hh
pope
```

in the main part of our script, after the statements that create the soma compartment.

At this stage, we now have the two channels, but we have not yet linked them to the soma. The soma needs to know the value of the channel conductance and its equilibrium potential in order to calculate the current through the channel. The soma will use this current as part of its calculation to update the soma membrane potential. The channel calculates its voltage- and time-dependent conductance using the current value of the soma membrane potential. As usual, these communication links are established by setting up messages between the elements. The soma may be linked to the sodium channel with the statements

```
addmsg /cell/soma/Na_squid_hh /cell/soma CHANNEL Gk Ek
addmsg /cell/soma /cell/soma/Na_squid_hh VOLTAGE Vm
```

You should now add these to your script, along with the corresponding messages for the potassium channel. Before running the simulation, ask yourself if there is anything else that needs to be done.

14.4 Final Additions and Improvements

At the end of the previous tutorial, we discussed some guidelines for choosing an appropriate integration step to be specified with the GENESIS *setclock* command. With no active channels, this simulation produced a smooth asymptotic increase to a constant value of V_m. Now that we have added the channels, we expect to see action potentials with a fairly short rise time. You should therefore use either your knowledge of neurobiology or trial and error to set the step size to a suitable value.

You should now be able to click on the RUN button to produce a sequence of realistic looking action potentials. With the injection current set at 0.3 *nA*, they should occur at intervals of roughly 14 *msec*. You may notice something a little strange about the first few steps of the simulation, however. Instead of starting at a resting potential of -70 *mV*, the membrane potential starts at about -59 *mV* and then becomes more negative. This effect becomes more obvious if you set the injection current to zero. You would expect to see a constant *Vm* of -70 *mV*, but the first 10 *msec* of the simulation show a different behavior. In general, the initial steps of a neural simulation will not begin with the system in a "natural" state. One solution to this problem is simply to run the simulation until it reaches a steady-state behavior before taking data. In our case, it is possible to perform a more realistic initialization if we understand a few details of the GENESIS *reset* function and its effect upon **compartment** objects and channels.

14.4.1 Use of the Compartment *initVm* Field

In Sec. 12.5, we have discussed some of the actions performed by GENESIS objects. The *reset* command causes each element to perform its own RESET action. For example, this will cause an **xgraph** object to clear the graph, unless the *overlay* field flag is set. If the element is a **compartment** object, *reset* means that the membrane potential *Vm* will be initialized to the value of the *initVm* field of the compartment. We haven't mentioned this field before, because it normally contains the same value as the *Em* field and follows any changes that are made to *Em*. Thus, the default behavior is that *Vm* gets intialized to *Em* after a reset. (Remember that *Em* is the "leakage" battery that is in series with the membrane resistance *Rm* in Figure 12.1.) The RESET action for the **hh_channnel** object causes it to get *Vm* from any incoming messages and to use it to calculate initial values for the Hodgkin-Huxley rate constants and the channel conductance *Gk*. If there were no passive channels or sources of current other than the channels explicitly added to our model, we would set the soma *Em* field to *EREST_ACT* (-70 *mV*). As this would indirectly set the *initVm* field to the same value, this initial value of *Vm* would be used to calculate the initial channel conductances after a *reset*. These conductances, with the channel equilibrium potentials, would result in no net current flow into the soma. Thus *Vm* would remain at *EREST_ACT* in the absence of any current injection.

In the Hodgkin-Huxley model, the Na and K channels produce a net current flow at *EREST_ACT*. In order to offset this current, *Em* is set to *Eleak*, a leakage potential that differs from *EREST_ACT*. With the proper value of *Eleak*, there will be no net current flow, and *Vm* will remain at the steady-state value of *EREST_ACT*. However, after a *reset*, we want to start the simulation with *Vm* initialized to *EREST_ACT*, rather than *Eleak*. This may easily be accomplished by setting the soma *initVm* field to *EREST_ACT* after the soma compartment has been created and *Em* has been set to *Eleak*. Once *initVm* has been set to a value which is different from that of *Em*, future changes to *Em* will no longer affect

initVm. (If you later want to have *initVm* follow changes to *Em*, just set it to the same value as *Em*.) With this final addition to the script, everything will be properly initialized. Your simulation results should now look like those shown in Fig. 14.1.

You might notice that, even with this change, the first action potential is larger than subsequent ones. This is not a bug in the simulator or your script, but is a legitimate result of the Hodgkin-Huxley model. You may wish to experiment with different injection currents and see if you can explain this behavior.

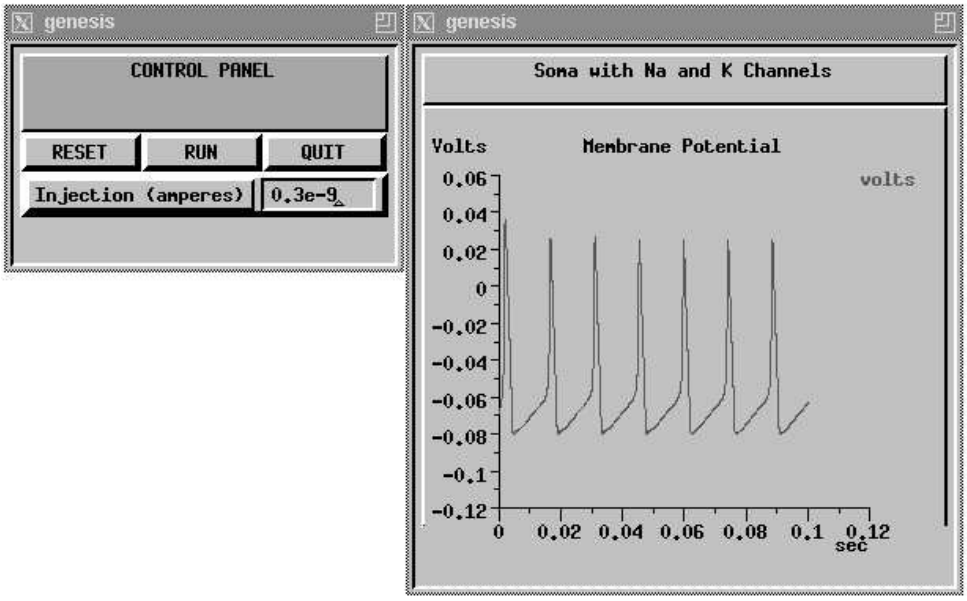

Figure 14.1 Typical results for the GENESIS simulation of a soma with Hodgkin-Huxley sodium and potassium channels.

14.4.2 Overlaying GENESIS Plots

There is yet another addition that you may make to your simulation if you choose. The command "`showfield /data/voltage -all`" reveals that your voltage graph has a field called *overlay* that was initialized to zero. As mentioned above, by setting it to a non-zero value, you may suppress the clearing of the graph during reset. This will allow you to overplot results using different injection currents, step sizes or other parameters. Although you can set this field to different values from the GENESIS command line, it would be nice to toggle it back and forth between zero and one by clicking on a button. As this lesson has been long enough, we will save the discussion of the XODUS toggle widget for the end of Chapter 15. If you would like to add toggle buttons to your simulations, you may find the information you need there, or in the GENESIS Reference Manual.

14.5 Extended Objects

You may encounter GENESIS scripts that use *extended objects* to create voltage-activated channels, rather than functions such as those defined in *hhchan.g*. Extended objects are created by a newly added GENESIS feature that allows you to use the GENESIS script language to create your own objects. The starting point for an extended object is an element or hierarchy of elements created from existing GENESIS objects that have at least some of the properties of the new object which you would like to create. New fields, message definitions and actions may then be added to the root element of the hierarchy before it is converted to an extended object.

For example, you may wish to create a specialized version of the **hh_channel** object in order to implement a Hodgkin-Huxley potassium channel. Or, you might replace the function *make_Vmgraph* with an extended graphical object that is appropriate for plotting membrane potentials. Typically, this would consist of a form, graph and overlay toggle button linked by messages, and would have the various fields pre-set to appropriate values for plotting membrane potentials. It might also be convenient to define a compartment object that can calculate and set its own values of *Rm*, *Cm* and *Ra* from the global variables *RM*, *CM* and *RA* using the compartment dimensions that have been set in its own *len* and *dia* fields.

As a specific example, let's create an extended object to represent a "squid-like" Hodgkin-Huxley potassium channel. When a channel element is created from this object, we would like it to not only have the fields for the various Hodgkin-Huxley constants (*X_alpha_FORM*, etc.) initialized to the appropriate values, but we would like it to perform some of the actions that were previously defined in our simulation script. For example, it should be able to calculate the area of its parent compartment from the compartment's *len* and *dia* fields, and use these to calculate and set *Gbar*, without having to previously specify a global variable *SOMA_A* and variables for the soma dimensions. It should also be capable of establishing its own messages with the parent compartment, so that we don't have to do this ourselves each time we add a channel to a compartment. We illustrate some of these features with excerpts from the script *hhchan_K.g*, which is listed in Appendix B.

The script begins much like *hhchan.g*, with the definition of some global constants, the creation of an **hh_channel** element called *K_squid_hh*, and the setting of the various fields. One difference is that a function is not defined to create *K_squid_hh*. Instead, it will be created by this script, modified, and then converted into a new GENESIS object with the same name. Also, we will arbitrarily set the *Gbar* field to zero, as it will be initialized to the proper value when the channel is created as the child element of a compartment.

Next, we add a new field *gdens* to the element to hold the conductance density of the channel, and set it to the default value of 360 S/m^2. This is done with the statements

```
addfield        K_squid_hh      gdens
setfield        K_squid_hh      gdens      360.0
```

The *addfield* command is used not just with extended objects, but may be used any time we wish to add a new field to an element.

As our new object will be a specific kind of channel, rather than a generalized Hodgkin-Huxley channel, it is best to protect the Hodgkin-Huxley channel constants from being inappropriately changed by the user. In fact, it would be a good idea to hide them from view, so that "`showfield -all`" will show only fields about which we care. We would like to be able to inspect the *Gbar* field, but it should only be indirectly settable by setting *gdens*. We set these two different levels of protection using the *setfieldprot* command with the statements

```
setfieldprot K_squid_hh -hidden  Xpower Ypower X_alpha_FORM \
    X_alpha_A X_alpha_B X_alpha_V0 X_beta_FORM X_beta_A \
    X_beta_B X_beta_V0 Y_alpha_FORM Y_alpha_A Y_alpha_B \
    Y_alpha_V0 Y_beta_FORM Y_beta_A Y_beta_B Y_beta_V0
setfieldprot K_squid_hh -readonly Gbar
```

The second statement protects the *Gbar* field, but uses the "`-readonly`" option instead of "`-hidden`", so that it will be visible with the *showfield* command, but will not be settable with *setfield*.

Now we need to define a function that extends the SET action of the **hh_channel** object to also set the *Gbar* field whenever the *gdens* field is set. This is accomplished with:

```
function K_squid_hh_SET(action, field, newvalue)
    float newvalue
    float PI = 3.14159
    if (field == "gdens")
        setfield . Gbar {PI*{getfield .. dia}*{getfield .. len}*newvalue}
    end
    return 0        // indicate that SET action isn't yet complete
end
```

When the *setfield* command is given, the three arguments *action*, *field* and *newvalue* are automatically passed as strings. As we would like *newvalue* (the value used to set the *gdens* field) to be a float, we redeclare it here.

This is the first time that we have mentioned the GENESIS *if* construct so far, although it also appears in *hhchan.g*. In the next chapter (Sec. 15.3) we will see an example of an *if-else*. In either of these constructs, the word "`if`" is followed by a space and then a pair of parentheses that contains a logical expression. If the expression is true, it evaluates to 1 and the statements preceding "`end`" will be executed. Note the use of the two equals signs

("==") to represent the logical operator for "equal to," as distinct from the single sign used to assign a value. The GENESIS logical operators are similar to those used in C, and are described in the GENESIS Reference Manual.

In our case, we want our new SET action to do something special if the field being set is our newly added *gdens* field. Otherwise, the default SET action of the **hh_channel** object will take over. Specifically, we want to multiply the new value of *gdens* by the area of the parent compartment and use this to set the *Gbar* field of the channel. Fortunately, the **compartment** object has two fields, *len* and *dia*, which may be used to hold the length and diameter of the compartment. Note that the "`.`" in the *setfield* expression refers to the working element (our channel) and that the "`..`" in the two *getfield* expressions refers to its parent element (the compartment containing the channel). When the SET action is called, the channel element becomes the working element during execution of the *K_squid_hh_SET* function. This change of working element takes place for all action functions. Also note that it is not necessary to name the working element with *setfield* and *getfield* commands, so the "`.`" could have been omitted. We also need to be sure that the *gdens* field itself gets set to *newvalue*. This will be done by the default SET action if we indicate that our SET action function did not set the field directly. We provide this information with the statement "`return 0`", which causes the function to return a value of zero.

Having defined a function to implement the new SET action, we next use the *addaction* command

```
addaction K_squid_hh SET K_squid_hh_SET
```

to specify the name of the element (*K_squid_hh*), the name of the action that it is to perform (SET) and the name of the function that implements the action (*K_squid_hh_SET*).

This will properly set *Gbar* when we change the value of *gdens*, but we would like to properly initialize *Gbar* when the element is first created. We would also like the messages that link the channel to its parent compartment to be automatically added when the element is created. Finally, we should put in some protection to ensure that our new kind of channel can only be created as a child element of a compartment.

We can accomplish these with a similar function that extends the CREATE action, and with an appropriate *addaction* command:

```
function K_squid_hh_CREATE(action, parent, object, elm)
    float PI = 3.14159
    if (!{isa compartment ..})
        echo K_squid_hh must be the child of a compartment
        return 0
    end
    setfield . Gbar \
        {PI*{getfield .. dia}*{getfield .. len}*{getfield . gdens}}
```

```
    addmsg . .. CHANNEL Gk Ek
    addmsg .. . VOLTAGE Vm
    return 1
end
addaction K_squid_hh CREATE K_squid_hh_CREATE
```

Here, we use the *isa* command to see if the parent element is derived from the **compartment** object. The negation operator "`!`" is then used so that if the channel's parent is not a compartment, an error message will be given and the function will return a zero, indicating that creation of the element failed. If the test is passed, we go on to calculate and set *Gbar* using the compartment dimensions and the default value that we previously established for *gdens*. The two *addmsg* commands are similar to those at the end of Sec. 14.3, adding a CHANNEL message from the channel to the compartment, and a VOLTAGE message from the compartment to the channel. The function ends with "`return 1`" to indicate that it was completed without error.

Finally, we want to turn the *K_squid_hh* element into an object called **K_squid_hh**. This is done with the *addobject* command, which is followed by the name of the object to be created and then the name of the element from which it was made. This command also allows options for specifying the name of the object's author and a description of the object. These will then appear with *showobject*, just as with a built-in GENESIS object. In our example, we have:

```
addobject K_squid_hh K_squid_hh -author "J. R. Hacker" \
    -description "Hodgkin-Huxley Active K Squid Channel - SI units"
```

Now, let's test the script. Start GENESIS, change to the *Scripts/tutorials* directory where *hhchan_K.g* should be found, and give the following commands:

```
hhchan_K
le
listobjects
```

Note that the element *K_squid_hh* doesn't exist, but that there is a new object of this name. Of course, we could have used a different name for the basis element and the extended object that we created from it. Either way, the *addobject* command destroys the original element when the new object is created. Try the commands

```
showobject K_squid_hh | more
showobject hh_channel | more
```

What differences do you note? Now try

```
create compartment /compt
setfield /compt dia 30e-6 len 30e-6
create K_squid_hh /compt/K
showfield /compt/K -all
showmsg /compt/K
```

How do the results of the *showfield* and *showmsg* commands compare with those obtained for the */cell/soma/K_squid_hh* element in your tutorial script? (You can also load *tutorial3.g* without interfering with any of the commands given so far.)

Try setting the */compt/K gdens* field to a different value and inspecting the value of *Gbar*. What happens if you try to set *Gbar* directly? What happens if you try to create a *K_squid_hh* element as a child of an element that is not a compartment?

The GENESIS Reference Manual describes some other capabilities of extended objects that are useful when constructing a compound object from a hierarchy of elements. Normally, all access to elements of the new object are via the fields, messages and actions of the root element, and the child elements are not accessible. Although this is usually desirable, there are cases when we want to access some of the fields and messages of one of the child elements. For example, if the root element is the form that contains the graph for plotting membrane potential, we would want to be able to add PLOT messages to the graph and to set the fields that specifiy the ranges for its axes. This may be done with *message forwarding* and *indirect fields*.

The GENESIS Reference Manual also describes the procedure for defining new objects and GENESIS commands by programming them in C, compiling them and then linking them into a new version of the executable *genesis* file. If this procedure is used to create a new object that has a computationally intensive PROCESS action added, it will execute more rapidly than one that is created at the script level. Nevertheless, it may speed development to make initial tests using an extended object as a prototype. For the example we have given here, there is little speed disadvantage in using an extended object, because the script language functions are used only when an element is created, or when fields are set. The compiled code for the original **hh_channel** object is used during the PROCESS action.

14.6 Exercises

1. It would be useful to be able to modify both the time step *dt* and the run time *tmax* from dialog boxes. Add these features to your simulation and have the x-axis of the graph adjust to the new *tmax*.

2. The function *make_Ca_hip_traub91* in *neurokit/prototypes/traub91chan.g* will create a high threshold calcium channel *Ca_hip_traub91*. (Except for the more descriptive name, this is the same as the Ca channel used in the *traub91* tutorial of Chapter 7.)

Chapter 19 describes how this channel was created from a **tabchannel** object. As with the channels in *hhchan.g*, we may use it without having to be concerned with the details of the *traub91chan.g script*. Copy this file into your directory and *include* it in a copy of your simulation script. Then make the changes necessary to add the channel to your model. In order for it to have a significant effect, you will need to increase the value of the *Gbar* field from its default value. What value of *Gbar* gives an appreciable "shoulder" to the action potentials?

3. In Sec. 14.2, we mentioned the *makegraphscale* function. Use this function to add a `scale` button to each of your graphs, so that you may easily change the *xmin*, *xmax*, *ymin*, and *xmax* fields.

4. The *Squid* tutorial (Chapter 4) uses the script *squid_electronics.g* to simulate the circuitry that is used for voltage clamp experiments. This is accomplished with a combination of the **PID** (Proportional, Integral, Derivative controller), **diffamp** (differential amplifier), and **RC** (low pass filter) objects. The GENESIS Reference Manual and the on-line help provide additional documentation for these devices. Adapt this script to provide voltage clamp capability for your simulation.

5. Why is the first action potential of the series shown in Fig. 14.1 higher than the following ones? It may help to revisit the *Squid* tutorial (Chapter 4) in order to examine plots of some of the other variables.

6. Make a script analogous to *hhchan_K.g* to create an extended object for the Na channel. Modify your *tutorial3.g* script to use extended objects instead of the functions defined in *hhchan.g*. Don't forget to set the *len* and *dia* fields of the soma compartment.