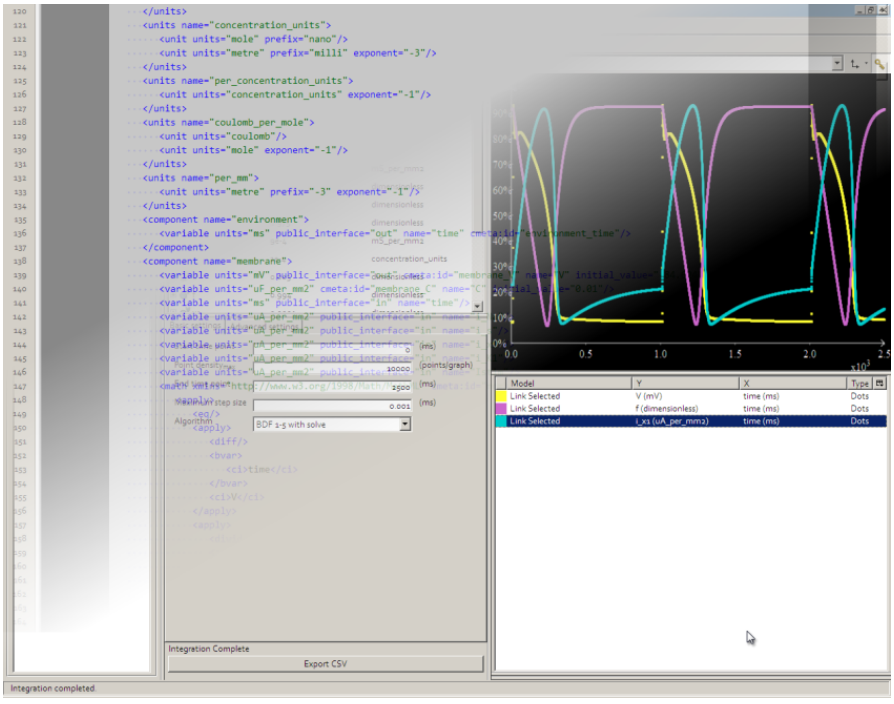
Auckland Bioengineering Institute
An Institute of The University of Auckland

cellML

Auckland Bioengineering Institute CellML tutorial



Virtual Physiological Human
network of excellence

1 Contents

Auckland Bioengineering Institute CellML OpenCell Tutorial

2 About this Tutorial

This tutorial is divided into three parts. The first is designed to acquaint you with the CellML software OpenCell, by guiding you through the process of creating a very simple model of a single chemical reaction.

The second and third parts of the tutorial emphasize the modularity, reuse, and exchange features that are available when developing models in a CellML 1.1 framework. The focus of these parts of the tutorial is the application of a CellML based modelling framework in the area of cardiac cellular electrophysiology.

The reference description can be browsed from the models.cellml.org a1 workspace:
http://models.cellml.org/workspace/a1/

For this tutorial, you will be provided with a folder containing the files required to complete the exercises.

3 Creating a basic model in OpenCell

This part of the tutorial will teach you how to create a simple, functional CellML model from scratch using OpenCell, and how to display the results of simulating this model on a graph.

3.1 Starting a model

We'll begin by defining some equations and parameters that we would like to use CellML to describe. Let's say we want to create a CellML model the simple chemical reaction shown below:

$$A + B \underset{k_b}{\overset{k^f}{\leftrightarrow}} C$$

Using mass action kinetics, we might model this using the following equations:

$$J = k_f[A].[B] - k_b.[C]$$

$$\frac{d[A]}{dt} = -J$$

$$\frac{d[B]}{dt} = -J$$

$$\frac{d[C]}{dt} = J$$

Where *[A]*, *[B]* and *[C]* are the concentrations of the reaction species, k_f and k_b are rate constants and *J* is a flux.

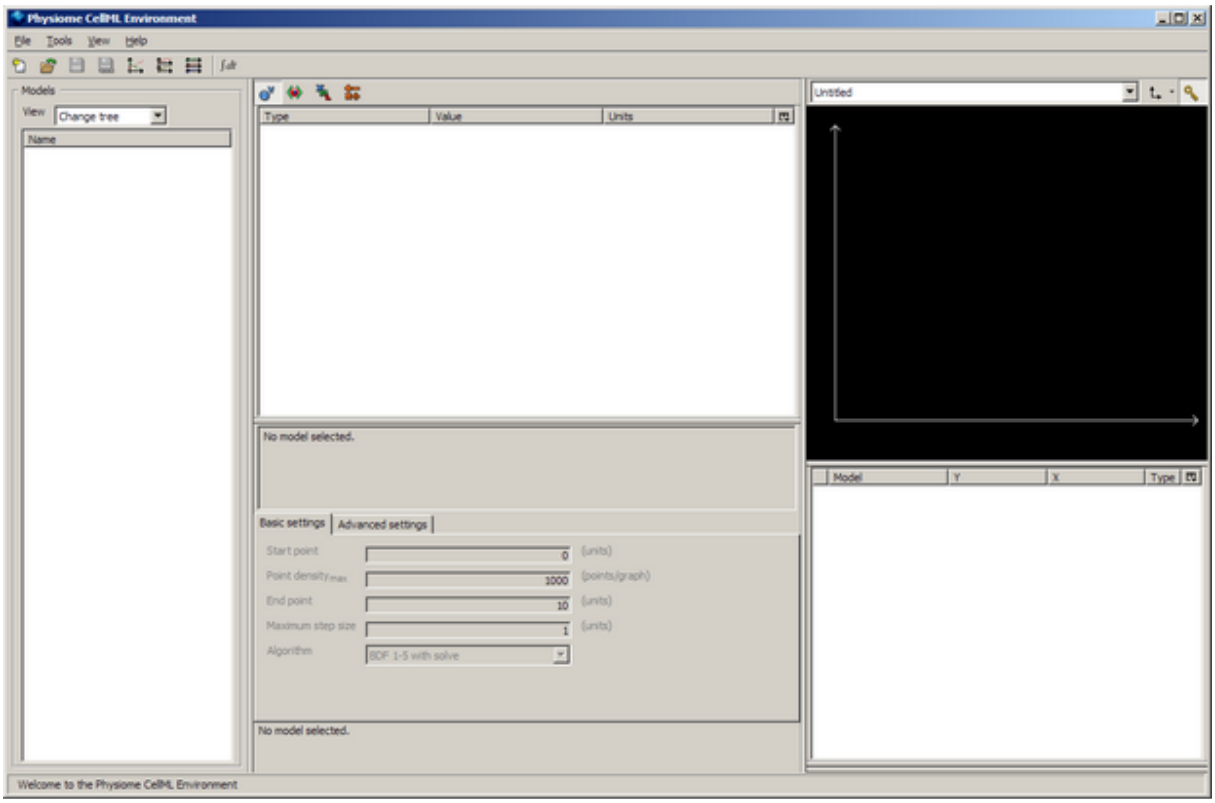
To create this model in CellML, we will use two *components*: an *environment* component to contain the *time* variable, and a *flux* component to contain the reaction. We will use seven variables: *time*, the reaction species *A, B,* and *C*, the rate constants k_f and k_b, and the flux *J*. There will be a connection between the two components to pass the *time* variable.

Although the best description of elements within a CellML model can be found within the CellML 1.1 Specification, a brief comment on components and connections is in order here.

Every variable in a CellML model is defined within a component. These components allow a modeller to organize and group variables and equations into modules. There is no ordained way to do this, so a modeller may in fact choose to create only one component and put everything in the model within it, but usually the components represent discrete processes or entities within the model, such as protein interactions in a signal transduction model. This is useful both because it keeps the model neat and aids conceptualization, and because it facilitates the creation of reusable modules.

Connections allow the modeller to map variables within different components to each other. This means that a variable may be defined by an equation in one component and mapped to another component to be used. Because the derivative of each ODE in this model needs to be taken with respect to time, the variable that tracks time needs to be defined once and mapped to each component.

When you first load OpenCell, it will look something like this (depending on your operating system and monitor resolution):

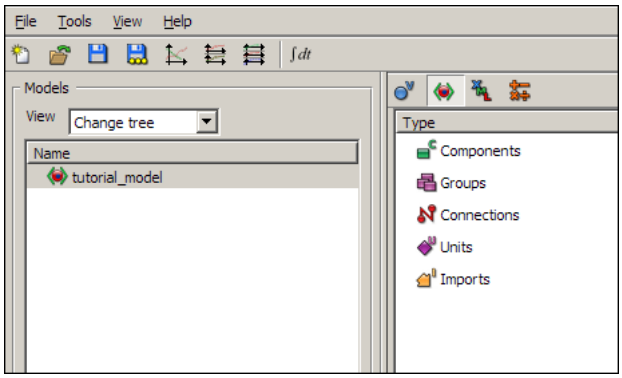


The window is divided into four sections; the model selector pane, tree view pane, interaction pane, and the graph pane. For more information about the OpenCell interface, refer to the manual - available from the *Help* menu from within OpenCell.

3.2 Creating a new CellML model

Click on the first button on the toolbar (*new model*) or go to the *File* menu and select *New*. A window titled *Create new model* will appear, asking for a name and a CellML version for your new model. Type in `tutorial_model`, and select *CellML 1.1* for the *CellML version*. Press enter, or click on *OK*. You will now have a model called *tutorial_model* visible in the model selector pane.

Now switch to the *show complete model structure* view in the *tree view pane*, using the second button on the tree view pane toolbar. This view is the main view you will use to edit models in OpenCell.



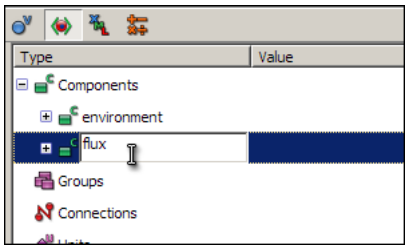
You will see a list of items in the tree view pane; *Components, Groups, Connections, Units,* and *Imports*.

Click on the *Save* button (the diskette icon) or select *Save* from the *File* menu. Give your file the name `tutorial_model.cellml` and save it in your chosen folder. Save regularly during this tutorial.

3.2.1 Making new components

You are going to create two components; one called *environment* to contain the global variable *time*, and another called *flux* to contain the variables and equations for the reaction.

To create a new component, right click on the green icon labelled *Components*, and select *New* from the context menu. A component called *New_component* will appear below *Components* - select the name of this component, type in `environment` (in place of `New_component`) and press enter. Now create another component, and rename it to `flux`.

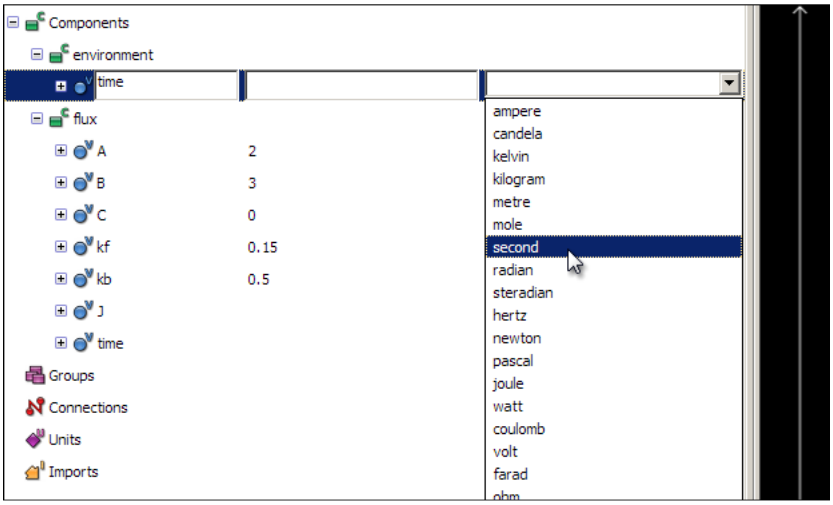


3.2.2 Creating variables

You will now create the variables for the model, within the appropriate components. First create the *time* variable: right click on the *environment* component's icon in the tree view (the green rectangle with a C, labelled *environment*) and select *New variable* from the context menu. Re-name the variable from *New_variable* to `time`.

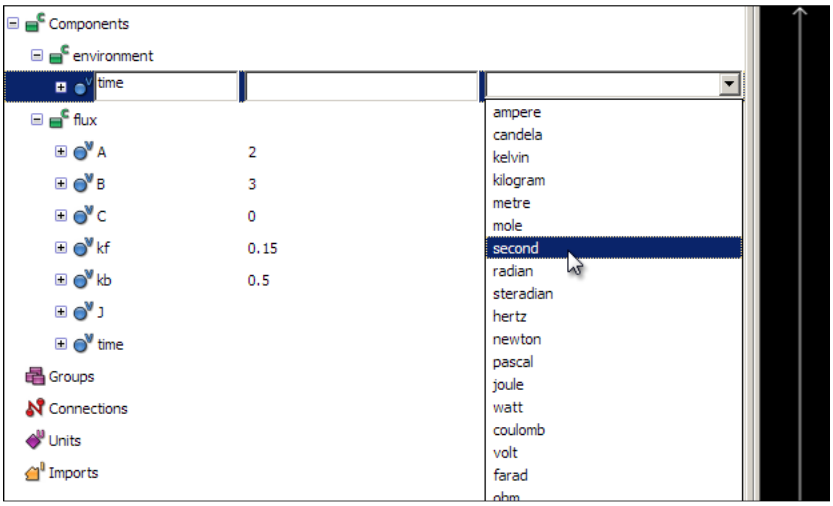
The other six variables should be created in the flux component. Create the variables *A, B, C, kf, kb* and *J* under the *flux* component. Some of these variables will require *initial values*. These can be entered in the *Value* field of the tree view. Enter values of `2`, `3`, and `0` for the variables *A, B*, and *C* respectively. These values are the initial concentrations of these three species. The rate constant variables *kf* and *kb* should be given values of `0.15` and `0.5`.

You will also need to create the variable *time* in the flux component, which will later be connected to the *time* variable in the *environment* component. Variables that are connected must appear in both of the connected components.



3.2.3 Defining units

All variables and numbers in a CellML model must be associated with units. OpenCell has a library of pre-defined units, which includes all base and derived SI units. First, associate the unit second with the *time* variable - click on the *Units* field of the time variable in the tree view, and select *second* from the list of pre-defined units.

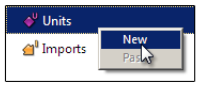


The other units used in this model will have to be defined, as they do not appear in the list of pre-defined units. The concentrations of the three reaction species are in units of micromoles per litre, or micromolar; the flux will be in units of micromolar per second. The two reaction rates will have units of per micromolar per second (for *kf*) and per second (for *kb*).

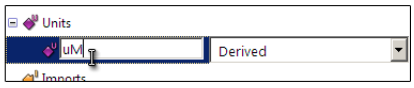
These units are made up of combinations of SI units, modified by *prefixes*, *multipliers*, *offsets*, and *exponents*. For example, the unit micromolar is made up of *mole* with a *prefix* of -6 (micro) combined with *litre* with an exponent of -1; micromoles per litre.

To define the unit micromolar, right click on the *Units* icon in the tree view and select *New* from the context menu (1). Rename this unit `uM`, and make sure that the type of unit is *Derived* (2). Now right click on the icon for *uM* and select *New unit* again from the context menu. Select *mole* from the list of units next to the yellow triangular unit node icon (3). To make these micromoles, enter a value of `-6` into the *Prefix* text field, or select *micro* from the list of prefixes. Right click again on the *uM* icon, and select *New unit* again. For this new unit, select *litre* from the list of pre-defined units. To make this per litre, enter a value of `-1` into the *exponent* field of the litre unit (4). When you now select the main unit node in the tree view, the unit definition will be displayed in the information pane (5).

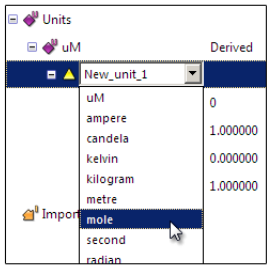
1



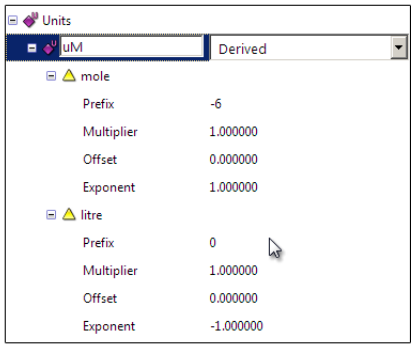
2



3



4



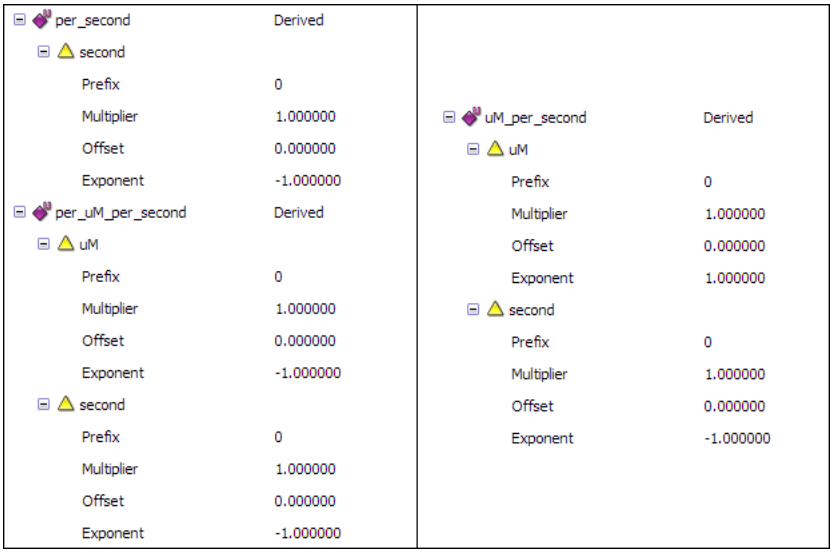
5

$$\mathrm{uM} = \mu\mathrm{mol} \times L^{-1}$$

To define the remaining units follow the procedure above, using the components of each unit.

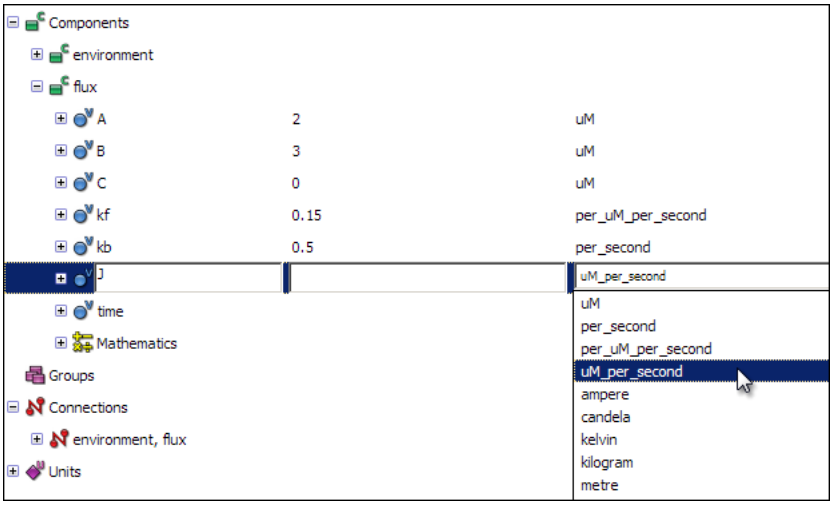
- The *per_second* unit is defined as a second unit with an exponent of -1 .
- *uM_per_second* is defined using *uM* combined with the second unit with an exponent of -1.
- *per_uM_per_second* is defined using *uM* with an exponent of -1, combined with the second unit with an exponent of -1.

This will give you the following unit definitions:



Now that you have defined all the required units, you can associate them with the variables. Click on the units field of each variable, and select the appropriate unit for each of the variables.

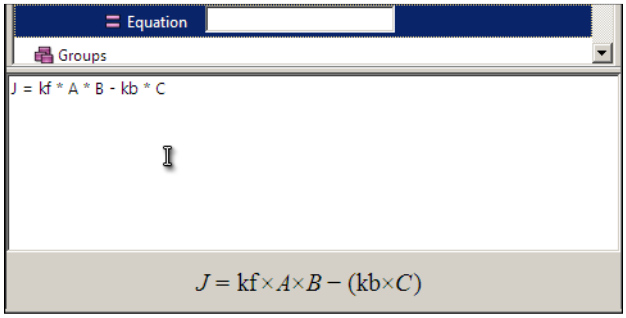
A: uM
B: uM
C: uM
kf: per_uM_per_second
kb: per_second
J: uM_per_second



3.2.4 Creating equations

Now you will enter the model equations. These are created within the appropriate component, in this case the flux component. Right click on the icon for the component in the tree view, and select *New maths* from the context menu. An icon labelled *Mathematics* will appear - right click on it and select *New equation* from the context menu. Click on the equation element, and in the text box (which will initially read *<no operator yet>*) enter the following equation:

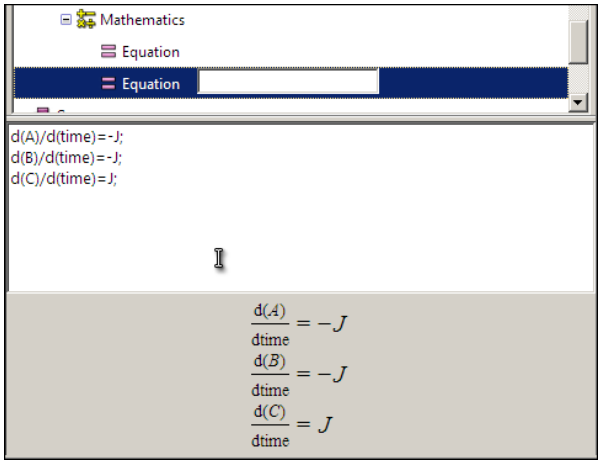
```
J=kf*A*B-kb*C
```



For the second, third, and fourth equations, create another equation and enter the following in the text box:

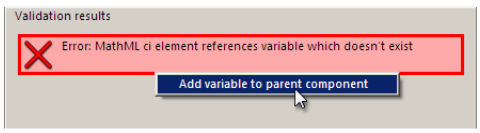
```
d(A)/d(time)=-J;
d(B)/d(time)=-J;
d(C)/d(time)=J;
```

OpenCell allows you to create multiple equations at once by entering a number of equations separated by semicolons. When you press enter, the multiple equations will appear as separate elements in the tree view. You may also enter your equations directly into the *Mathematics* tree-view element text box.



After you have entered the equations, selecting them will cause a formatted version of the equation to appear in the interaction pane below the tree view. This will only work if you have entered an equation in a way that OpenCell is able to interpret. The input format is an editable form of MathML and is similar to the format for a basic command-line calculator tool.

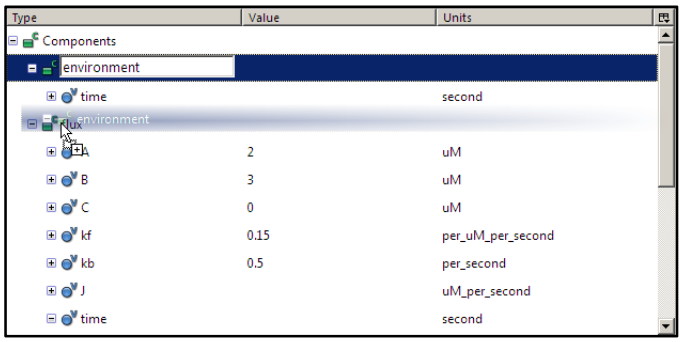
It is worth noting that if you create equations without having first created the appropriate variables, OpenCell will show errors when the model is validated. You can right click on these errors and select *Add variable to parent component* to automatically fix these errors, creating the required variable elements within the component that the equation is in. These automatically created variables will have no initial value and their units will be set to *dimensionless*. You will have to edit these fields if required.



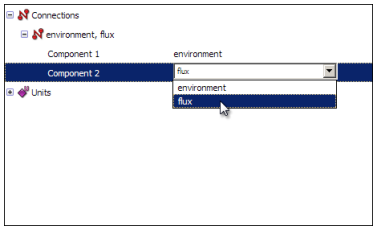
3.2.5 Creating connections

Finally, you will need to create a connection between the two components, to pass the *time* variable from the *environment* component to the *flux* component. As we mentioned previously, connections allow variables which have been defined in one component to be used within another component.

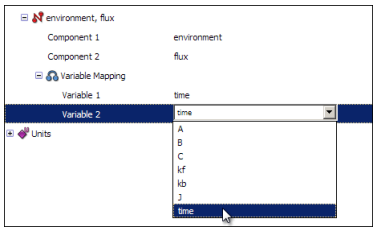
There are two ways to create connections in OpenCell. The easiest is to use the drag & drop method. To connect time in environment to time in flux, simply hold down the control key, drag the *environment* component, and drop it on the *flux* component. All of the common variables will be connected, which in this case is only *time*.



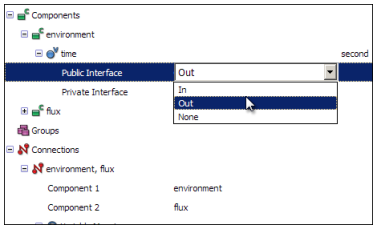
Alternatively, you can create connections using the context menus in the tree view. To create a connection, right click on the red *Connections* icon in the tree view, and select New from the context menu. Expanding the newly created connection will show you two items named *Component 1* and *Component 2*. Select *environment* and *flux* respectively from the drop-down menus.



Now, right click on the connection labelled *environment*, *flux* and select *New variable mapping* from the context menu. Expanding the new variable mapping will show you two items labelled *Variable 1* and *Variable 2*. You want to map the variable time from the environment component onto time from the *flux* component, so select *time* in both of the fields.



For any newly created connection to function (created using either method), the direction of the *Public Interface* of the *time* variables must be set correctly for each component. Do this by first expanding the *time* variable in the *environment* component, and then selecting *Out* from the *Public Interface* drop-down menu. Setting up the interfaces can be done before connections are created, at the time when the variables are created.

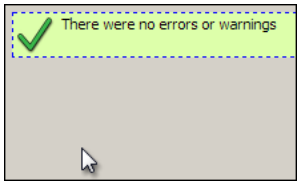


In the same way, change the *Public interface* of the *time* variable in the *flux* component to *In*. This sets the direction of the connection, as shown in the information pane when a correctly created connection is selected.

environment		flux
time (second)	→	time (second)

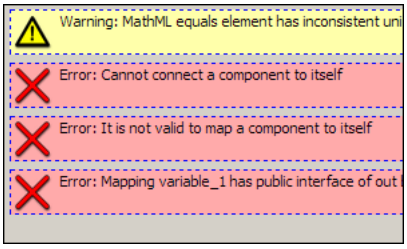
3.2.6 Validating the model

Validating the model will help you to identify and locate any errors you might have made in building your model.



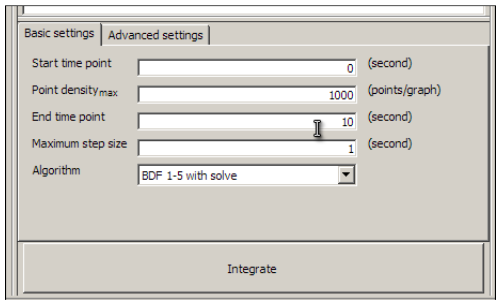
To validate the model, right click on the model name in the model selector panel on the left, and select *Validate* from the context menu. If there are no errors, a message will appear in the far right hand pane of the OpenCell window, the graph pane.

If there were errors, messages will appear describing the errors found. Double-clicking on the error messages will take you to the part of the model where the error was detected. Take note that you must be in the *complete model structure view* for this error highlighting function to work correctly. Shown on the left is an example of what these error messages might look like:



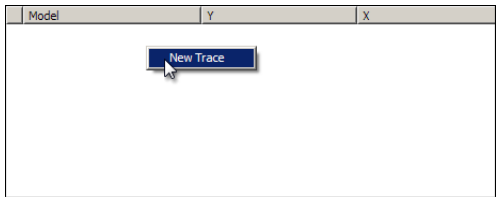
3.2.7 Running a simulation

Our model is now validated and ready to simulate. Save the model now, and then click on the Display integration pane button (the last button on the toolbar, with the letters *dt*,) to make the integration pane visible. You have to close and re-load the model before the integration pane becomes available. The integration pane should look something like this:

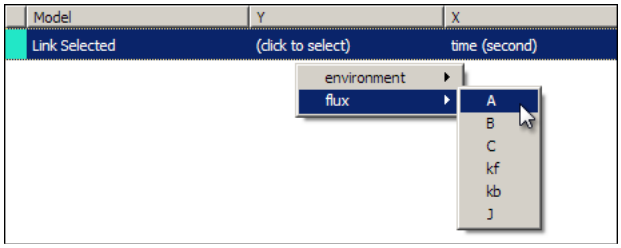


Click on the large *Integrate* button, and wait until the message *Integration complete* appears in the integration pane. The *Integrate* button will change to the *Export CSV button*. This button allows you to save the results of the simulation as comma separated values, for loading into other applications. You now have results for the model which you can display in the graph pane.

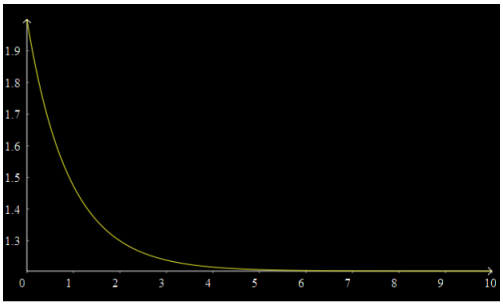
Right click in the graph trace control area in the graph pane, and select *New Trace* from the context menu.



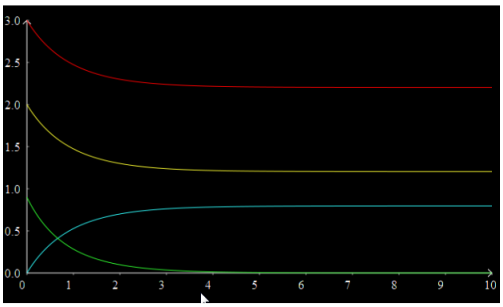
A new trace entry will appear in the listing. Click on the X column where it says *(click to select)* and select *environment*, *time* from the cascading menu. Do the same for the Y column and select *flux, A*.



A graph should now appear above, showing the concentration of *A* decaying over time to its equilibrium value.



Add more traces to the graph - another way to add graph traces is to simply drag variables from the tree view and drop them on the graph pane. Try doing this with the variables *B*, *C*, and *J* from the *flux* component. Additional new traces will automatically have time selected as the X axis variable, although you can of course change this if you wish by clicking on it and selecting a different variable. You should now have a graph that looks something like this:



You can change the colour of individual traces by clicking on the coloured block in the graph controls and selecting a colour from the pop up palette.

This concludes the first part of the tutorial. Please refer to the manual and other tutorials for more information.

4 Modifying Hodgkin & Huxley 1952 toward Noble 1962

This part of the tutorial is a detailed guide to the modification of an existing "published" model which is available on the internet. The example used is modifying the original Hodgkin & Huxley 1952 squid axon model to produce action potentials closer to those found in the Noble 1962 model of cardiac cells.

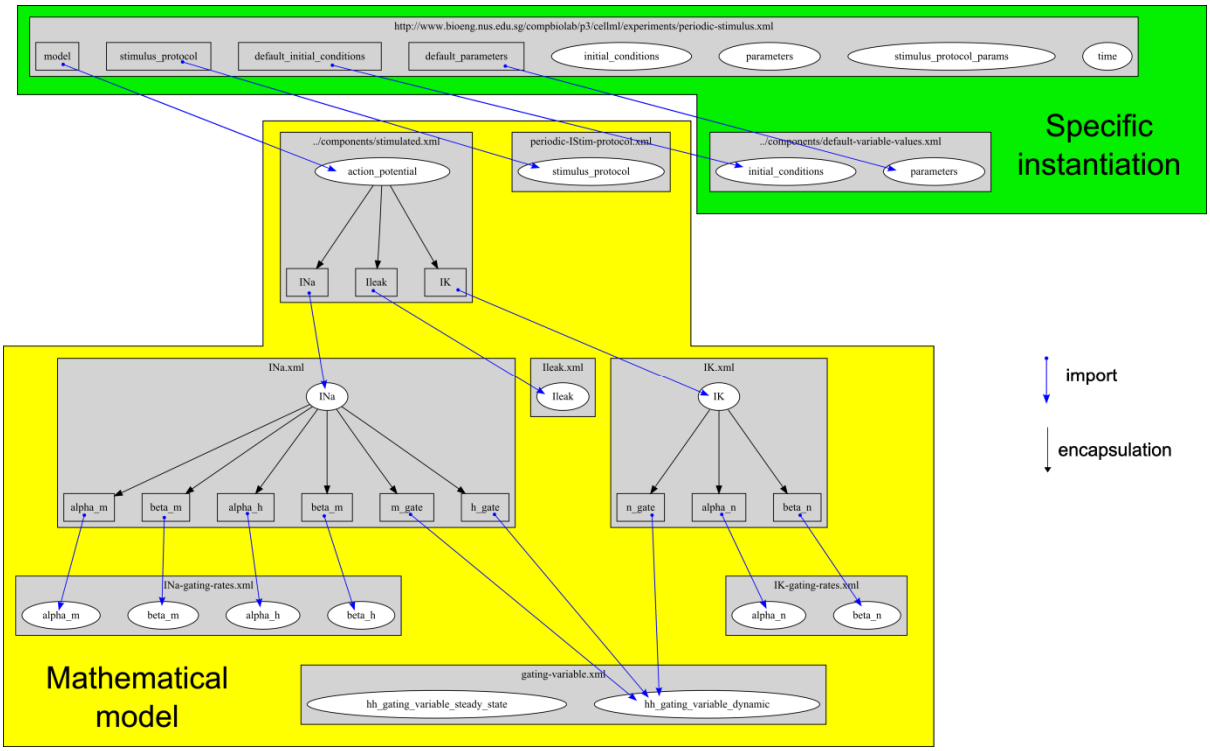
This section of the tutorial illustrates the modularity and reuse aspects of CellML 1.1. An accurate representation of the original Hodgkin & Huxley 1952 squid axon model has previously been published and made available on a web server at NUS in Singapore.

In this task we first validate this implementation of the model and then modify it through the replacement of the potassium current from a model available in the CellML model repository, and the slight modification of some parameters. These modifications are sufficient to turn the short duration of a nerve action potential into a long lasting cardiac action potential similar to the Noble 1962 cardiac Purkinje fibre action potential.

The basic premise of this example is based on the first biophysical cardiac cellular electrophysiology model, published by Denis Noble in 1962. The Hodgkin-Huxley 1952 squid axon model is modified, such that it becomes capable of reproducing the action and pacemaker potentials observed in Purkinje fibres. In this tutorial we start with a CellML description of the Hodgkin-Huxley model and modify it to reproduce some of the key features of the Noble 1962 model.

4.1 Hodgkin-Huxley Model Structure

The structure of the NUS implementation of the Hodgkin-Huxley model is illustrated in the figure below:



This diagram emphasizes the modular definition of the model using CellML 1.1. The mathematical model is separated from the instantiation into a specific simulation experiment. Reuse is illustrated by the use of a single definition of the standard Hodgkin-Huxley gating kinetics being reused for each of the m, h, and n gating variables.

4.2 Hodgkin-Huxley Model Validation

The Hodgkin-Huxley model used here has been previously implemented and is available online from the Computational Bioengineering Laboratory at the National University of Singapore. The original source of these models is http://www.bioeng.nus.edu.sg/compbiolab/p3/. All the graphs illustrated here are simulated directly from the data described on the NUS server. The following simulations serve to validate this implementation by showing simulation observations which correspond directly to those published by Hodgkin & Huxley (1952).

All references to files loaded in OpenCell are relative to the IUPS-2009-CellML-tutorial-models folder extracted from the model archive (ROOT).

We describe below the steps for recreating two of the original validation studies provided under http://www.bioeng.nus.edu.sg/compbiolab/p3.

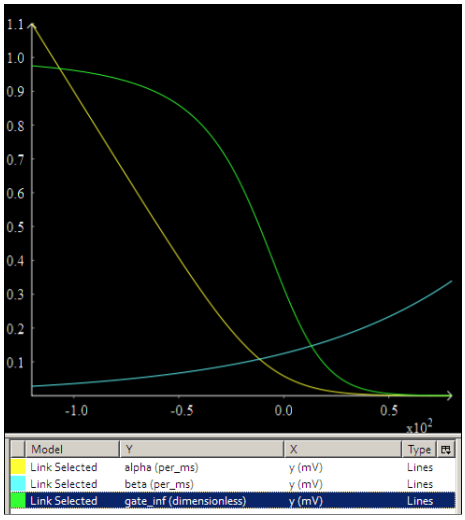
4.2.1 Potassium Channel Kinetics Validation

We will perform a validation study, investigating the kinetics of the potassium channel of the base implementation of the Hodgkin-Huxley model. Start OpenCell and load the model:

```
ROOT/NUS-HodgkinHuxley-Implementation/experiments/IK-gating-kinetics.xml.
```

The simulation metadata gets parsed (this sets up the settings in the *integration settings* panel) so you can leave all of the integration settings as they are.

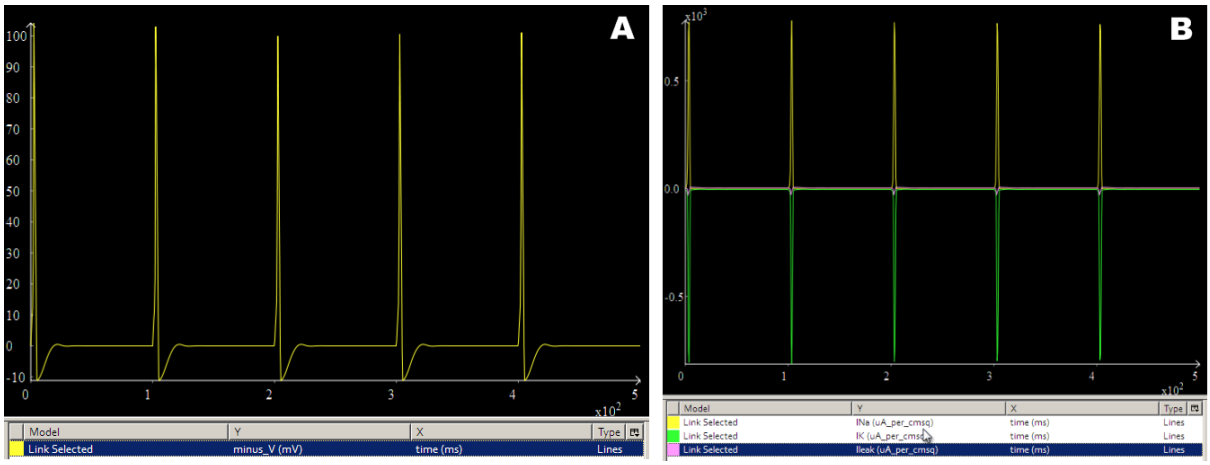
1. Click the Integrate button, and wait until the Integration Complete message appears.
2. Plot *alpha_n* and *beta_n* against *membrane potential*. To do this, right click in the area below the graph panel, and select *New Trace* from the context menu. Click where it says *click to select*, and select *alpha_n/alpha* for the *Y* axis, and *ramp/y* for the *X* axis (membrane potential is described in the model mathematically as a linear ramp with a generic name, although imported as V).This reproduces figure 4 from the Hodgkin & Huxley (1952) article, shown as the yellow and green traces in the graph below.
3. Plot *n_infinity* against *membrane potential*. [*membrane potential* as above; *n_infinity* = *hh_gating_variable_steady_state/gate_inf* (generic gating kinetics imported for each gate)]. This reproduces Figure 5 from the HH (1952) article, shown as the cyan trace below.
4. Click the large *Integrate* button near the bottom centre of the OpenCell window in the integration pane. The graph traces will appear as shown in the graph below, although the colours will be different as they are randomly selected. You can choose the colour of a trace by clicking on the small coloured box at the left of the trace settings, and selecting a new colour from the pop-up palette.



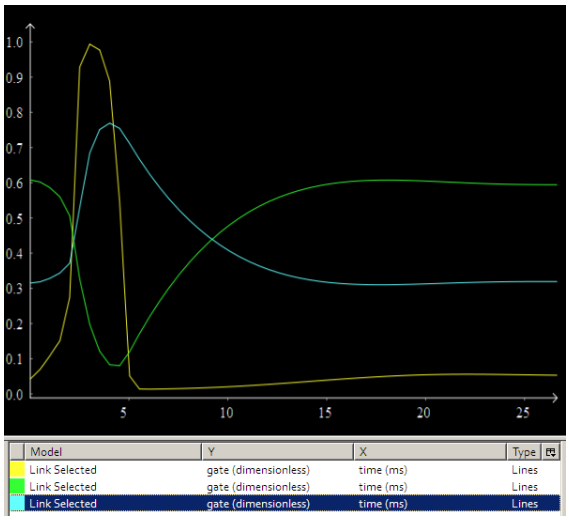
4.2.2 Action Potential Validation

We will perform a validation study investigating the dynamic response of the base implementation of the Hodgkin & Huxley model to an applied electrical stimulus current.

5. Load the following model into OpenCell:
   `ROOT/NUS-HodgkinHuxley-Implementation/experiments/periodic-stimulus.xml`
   Once again, the simulation metadata gets parsed and you can just click Integrate.
6. Plot action potentials [*action_potential/minus_V* vs *time/time*], as shown on graph A below.
7. Plot currents [*INa/INa* vs *time*; *IK/IK* vs *time*; *Ileak/Ileak* vs *time*], as shown on graph B below. The first variable can be plotted simply by changing the *Y* variable to *Ina/Ina* for that trace.

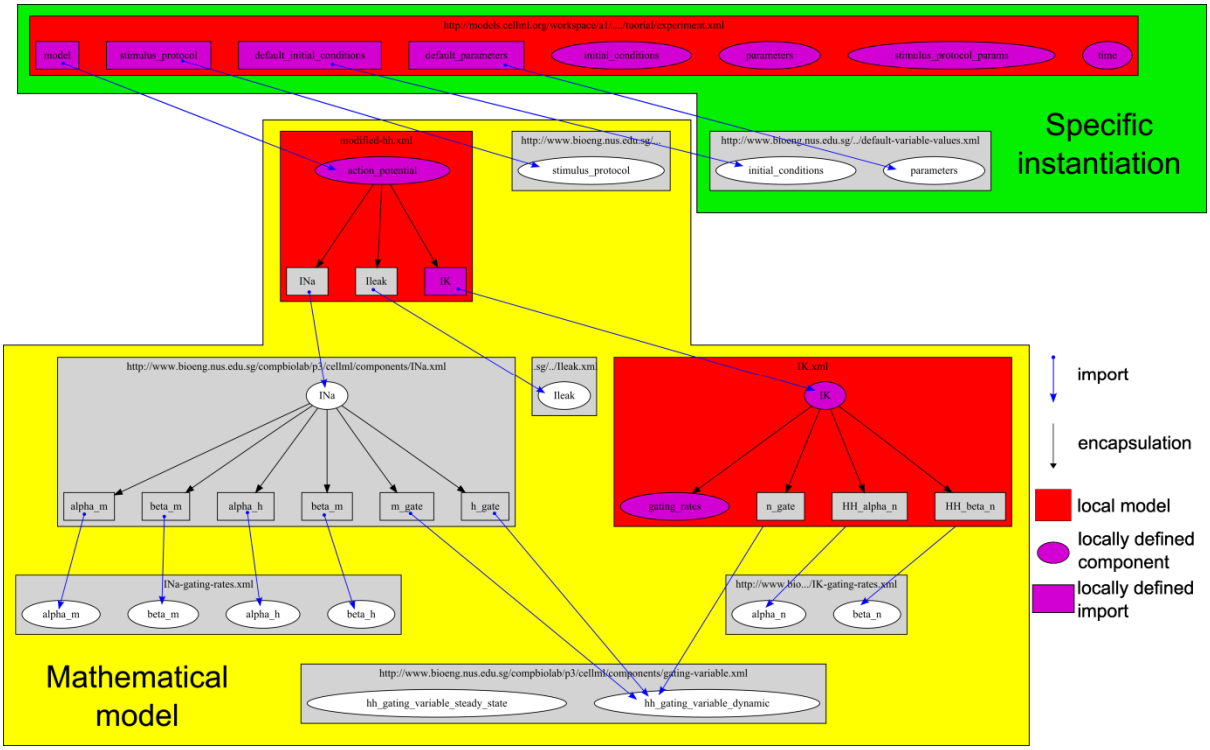


8. Plot gating variables [all three of *hh_gating_variable_dynamic/gate* vs *time*]. The graph below shows the gating variables *n* (cyan), *m* (yellow), and *h* (green). You can do this either by changing the *Y* variable on all 3 traces, or deleting the traces and creating them from scratch. Use the right mouse button to zoom in on the graph in order to view a single action potential as shown below.



4.3 Modification toward Noble 1962 Model

This task defines the modifications made to the base HH implementation in order to approximate the action potential shape of the Noble 1962 model. The figure below illustrates the model structure that we want to develop:



In this figure, all the red models are defined “locally” and the rest come from the base implementation from NUS, as described above. Within the local models, some of the components are imported from the base NUS model and some are newly defined. All units are imported from the base NUS model implementation. The major change to the mathematical model is to replace the Hodgkin & Huxley potassium channel with one which is based on the potassium channel described by the Noble 1962 model. To achieve this, the Noble potassium channel description has to be altered to reflect the modelling conventions used by Hodgkin & Huxley.

4.3.1 Examining the Mathematical Model

Load the following model into OpenCell:

```
ROOT/models.cellml.org/workspace/a1/tutorial/modified-hh.xml
```

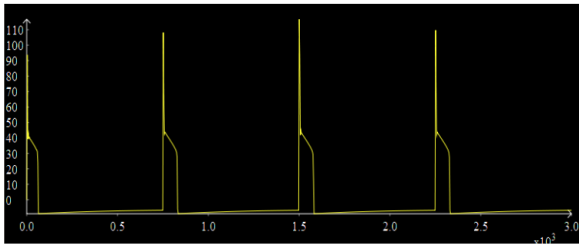
Ignore the message that says the model is unable to be integrated, and select the "show complete model structure" tree view by clicking on the second icon on the tree view toolbar. You can now see the structure of the actual mathematical model. Take a look at the imports elements, showing that the sodium and leakage current are being imported from “NUS” and the potassium current is being imported locally.

4.3.2 Modifying the Parameters

Close and re-start OpenCell. Now load the following model into OpenCell:

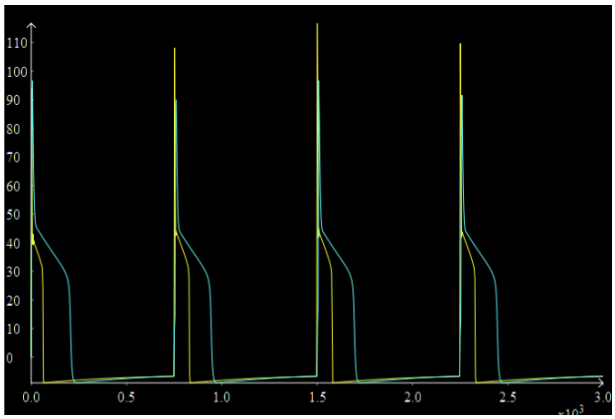
```
ROOT/models.cellml.org/workspace/a1/tutorial/experiment-start.xml
```

In this example, the modified mathematical model is imported into a specific experiment model and all the default parameter values and initial conditions are imported from the NUS model implementation. The stimulus protocol parameters are defined locally to define a suitable stimulus protocol. Integrating this model and plotting [*action_potential/minus_V* vs *time*] shows the altered action potentials which result from the changes made to the potassium current formulation. At this point, click on the trace settings where it says *Link selected*, and change this to *Copy selected model*. This will make this trace keep the original model data once we create a clone.



We will now make the changes required to create the plateau-like phase in the action potential.

9. Clone the model by using the context menu on the model in the listing at the far left, and selecting Clone. Select the new cloned model, which will be named with a timestamp.
10. Create a new component in the tree view (bring up the context menu on the Components element icon and select New). Name this component `parameters`.
11. Create a new variable in the parameters component, called Cm. Give it a value of `12`, and units of *uF_per_cmsq* (predefined in the drop-down menu). Change the public interface to *out*.
12. Create another new variable in the parameters component, called gK_max. Give it a value of `12`, and units of *mS_per_cmsq*. Change the public interface to *out*.
13. Expand the *model, default_parameters* connection, and delete the variable mappings for *Cm* and *gK_max* - these will be the first two, but you must expand them to see what the variable mapping actually is.
14. Create a new connection (use the context menu on the *Connections* element) and connect the components model and parameters using the drop-down selection boxes.
15. Create two new variable mappings within this new connection using the context menu on the model, parameters element. Expand the two variable mappings. Select *Cm* in both *variable 1* and *variable 2* for the first mapping, and select *gK_max* for both in the second mapping.
16. Integrate your edited model, and graph the same variables [*action_potential/minus_V* vs *time*] as for the original model. You should notice that the action potentials have the plateau-like phase, as shown below on the cyan trace:



5 Ten Tusscher, Noble, Panfilov 2004 (TNNP) stimuli

The following part applies the same techniques as demonstrated in part 4 to a more realistic scenario, namely that of modifying the stimulus protocol being applied to a human cardiac myocyte model.

In this section of the tutorial, we make use of an existing encoding of the ten Tusscher et al (2004) human ventricular myocyte model which has a periodic stimulus current applied. We replace this stimulus protocol with one suited for restitution analysis.

Nickerson & Buist (2008) (PBMB, Practical application of CellML 1.1...) published a study investigating the practical application of CellML 1.1 based on the ten Tusscher et al (2004) human ventricular myocyte electrophysiology model. The online supplement for this study is available at:

http://www.bioeng.nus.edu.sg/compbiolab/papers/nickerson-PBMB-2008/

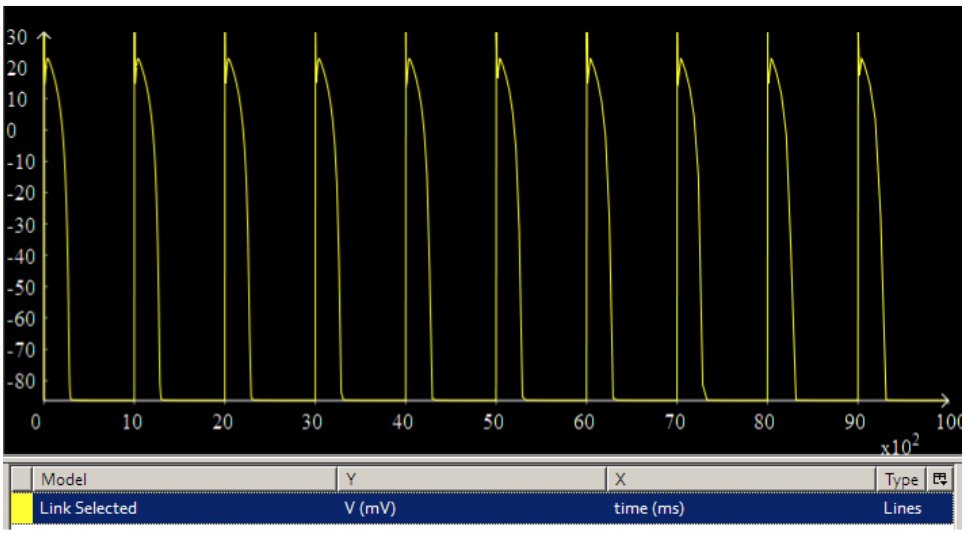
This includes all the simulation experiment descriptions presented in that study. As for the Hodgkin-Huxley example above, we first check that the base implementation available from the NUS server is correct and then modify it with a new stimulus protocol.

5.1 Periodic

Close and re-start OpenCell. Now load the following model into OpenCell:

```
ROOT/NUS-TNNP-Implementation/papers/nickerson-PBMB-2008/experiments/TNNP-periodic.xml
```

1. Change the end time point to `10000 ms` in the integration pane. Turn on tabulation control with a tabulation interval of `1ms` - the tabulation settings are in the *Advanced* settings tab of the integration pane.
2. Create a trace for the action potentials; *membrane_potential/V* vs *time/time*. The list of components and their variables in this model is very long, and you will have to scroll the list in order to find the ones you need to graph.
3. Click the *Integrate* button.

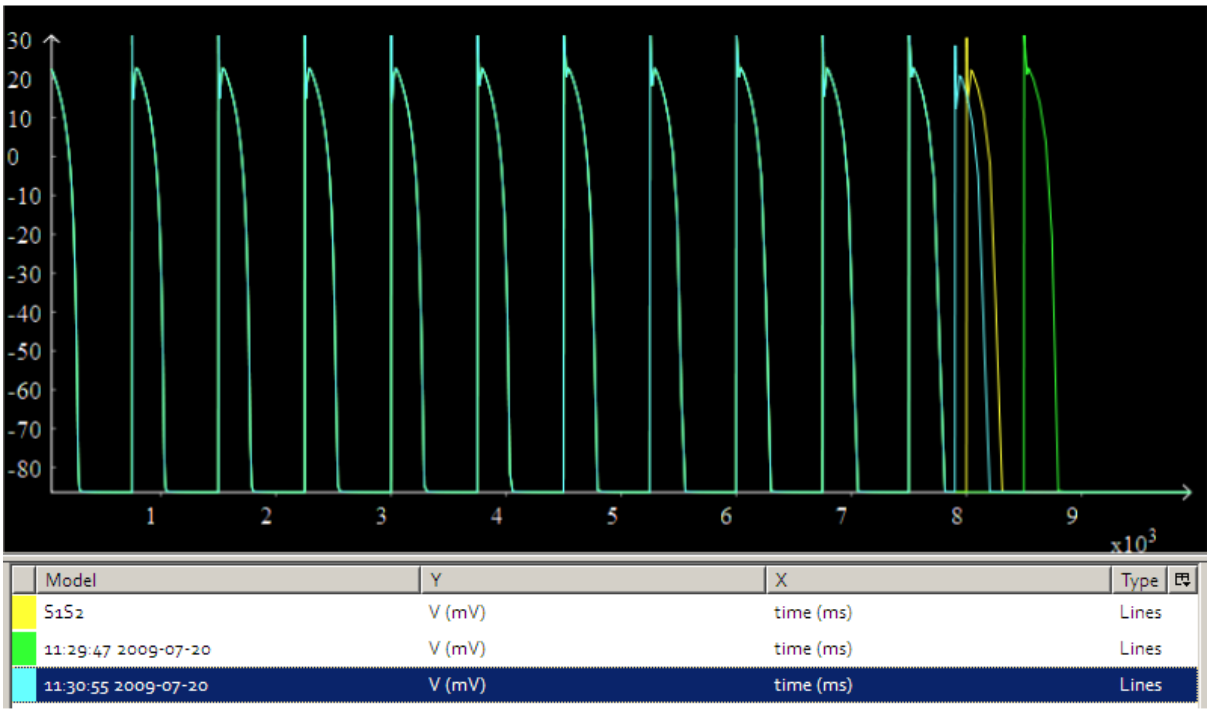


5.2 S1-S2 Restitution

The Nickerson & Buist (2008) implementation of the ten Tusscher et al (2004) model includes the description of the model with an applied S1-S2 stimulus protocol (10 stimuli with a fixed period followed by a single stimulus at some interval after the 10th periodic stimulus. This is a published and "fixed" set of CellML encoded models which we are not able to change directly, but we would really like to use this model with our own stimulus protocol description. Due to the modular nature of the model implementation available from NUS this is relatively straightforward to achieve, by following these steps:

1. Create a new model in OpenCell by selecting *New* from the *File* menu, or by clicking on the *New model...* button on the toolbar. Select CellML 1.1 as the version, and give the model the name `S1S2`.
2. Save the model to `ROOT/my-S1S2.xml`.
3. Create a new import, and enter the following in the text field:
   `NUS-TNNP-Implementation/papers/nickerson-PBMB-2008/experiments/TNNP-S1S2.xml`
4. Create a new component in this import; enter *model* and *interface* into the two text fields.
5. Create another new import, and enter the following in the text field:
   `NUS-TNNP-Implementation/cellml/models/common/units.xml`
6. Create a new *units* within this import; enter *ms* and *ms* into the two text fields.
7. Save your model again.
8. Create a new component in the top level model, with the name `parameters`.
9. Create a new variable within this new component, using the following settings:
   name: `basicCycleLength`
   value: `750`
   units: *ms*
   public_interface: *out*
10. Create another new variable within the parameters component, using the following settings:
    name: `S2Interval`
    value: `500`
    units: *ms*
    public_interface: *out*
11. Save your model again.
12. Create a new connection, expand it, and select parameters and model for the two components to connect.
13. Create new variable mapping within this new connection, and select *basicCycleLength* for *variable 1* and *variable 2*.
14. Create another new variable mapping, and select *S2Interval* for *variable 1* and *variable 2*.
15. **Save your model again!**
16. Right click on your model in the model listing at the far left and select *Reload*. This should now enable the integration of the model.
17. Set the tabulation control to on using the check box (in the *Advanced settings* tab), and set a tabulation interval of `1ms`.
18. Set the integration end time to `10000ms`, and the maximum step size to `0.1ms`.
19. Integrate the model, and plot the action potentials by creating a trace for *membrane_potential/V* vs *time/time*.
20. Change the trace setting from *Link selected* to *Copy selected model*.
21. Right click on the model in the model listing and select *Clone*. Select this cloned model.
22. Edit the *S2Interval* variable value of this cloned model; change it to `1000` for example - now integrate the cloned model, and create a trace of *membrane_potential/V* vs *time/time*.
23. Repeat steps 20, 21, and 22, but this time set the *S2Interval* to `400`.

You should now have a graph that shows three action potential traces - the first 11 peaks should overlap, and the last should be different between the three traces. Below is a graph showing these three traces, where *S2Interval* has been set to 400ms (cyan), 750ms (yellow), and 1000ms (green).



6 Conclusion

In this tutorial we have introduced you to the basic concepts of CellML 1.1, in particular the modularity and re-use features of the language. For further tutorials and information, please visit the *getting started* section at http://www.cellml.org/getting-started/.

We encourage you to get involved with the CellML project via the mailing lists and the tracker. You will find links to the mailing lists on the *getting involved* section of the website. You can discuss features of and issues with the software on the CellML sections of the Physiome Tracker, at https://tracker.physiomeproject.org/ .