Real-time eXperiment Interface (RTXI)
User Guide

RTXI v2.0

help@rtxi.org

2014-02-31

This file documents the Real-time eXperiment Interface (RTXI), a program for hard real-time data acquisition and control applications in biological research.

This edition of the RTXI documentation is consistent with RTXI v2.0.

Contents

1 Introduction

1.1 About RTXI

The Real-Time eXperiment Interface (RTXI) is a collaborative open-source software development project aimed at producing a real-time Linux based software system for hard real-time data acquisition and control applications in biological research. RTXI merges three previous systems for closed-loop biological experiments: RTLab [7, 8], Real-time Linux Dynamic Clamp (RTLDC) [11], and Model Reference Current Injection (MRCI) [4, 23]. RTLDC and MRCI focus on implementing dynamic clamp, an experimental technique in cardiac and neural electrophysiology that is used to simulate ionic membrane currents. RTXI combines the features of all three predecessor platforms into a more general platform for real-time closed-loop experimental protocols. Using real-time control, scientists can quantify biological function via perturbations that change according to closed-loop analysis of measured system variables, rather than being restricted to measuring responses to pre-determined stimuli. Real-time control applications are abundant throughout biological research, including, for example, dynamic probing of ion-channel function, control of cardiac arrhythmia dynamics, and control of deep-brain stimulation patterns. There is a wide range of biological research endeavors for which real-time control can offer insight that cannot be obtained with traditional methods.

RTXI is based on Linux, which is extended with Xenomai [**?**] to provide a hard real-time platform with the comprehensive Linux desktop environment [1]. Data acquisition and analog/digital interfaces to other hardware are implemented in real-time using the Analogy real-time driver interface, which provides support to a variety of commercial multifunction data acquisition cards. Experimental protocols and other real-time algorithms are implemented within a modular framework that allows users to easily reuse existing code and construct complex protocols. Users can also take advantage of previously written code or other C++ libraries to add functionality to their modules. As such, RTXI is a generic real-time platform with potential applications beyond dynamic clamp.

RTXI is released under a combination of the GPL and LGPL licenses. The core RTXI code is covered by a GPL license but user modules distributed as binary libraries are covered under LPGL and their source code may be available at the discretion of the original authors. All documentation is released under the GNU Free Documentation License.

! → If your use of RTXI leads to scientific publication, we request that you cite RTXI in your paper with text such as: "Experiments were performed using the Real-Time eXperiment Interface (RTXI; www.rtxi.org)."

---

[1] Legacy support is provided for the Real-time Applications Interface (RTAI) [19] and Linux Control and Measurement Device Interface (COMEDI) [6]

1.2 Overview of Features

RTXI contains many features that enable users to quickly implement complex interactive experimental protocols:

1. Modular signal-and-slots architecture that allows multiple instantiations of user modules, makes it easy to reuse code such as event detection and online analysis algorithms, and allows branching logic so that signals (such as acquired data) can be routed through multiple algorithms in parallel
2. Data acquisition system that can stream multiple channels of acquired or computed data along with experimental metadata and user comments with timestamps
3. Ability to interface with a variety of multifunction DAQ cards and external hardware through analog or digital channels, e.g. via TTL pulses
4. Ability to change experimental parameters on-the-fly without recompiling or stopping real-time execution
5. Ability to save and reload your entire working environment with custom parameter settings
6. Virtually no limit to algorithms that can be implemented since user modules are written in C++
7. Real-time digital oscilloscope that can plot any acquired or computed signal
8. Base class for constructing user modules with a customized graphical user interface to experimental protocols
9. Ability to "play back" previously acquired data or surrogate data as if it were being acquired in real-time for debugging or simulation purposes
10. A complete simulation platform that can be used to solve systems of differential equations in real-time and integrate biological signals acquired in real-time with model systems

In addition, RTXI is available on a Live CD, which provides a complete real-time Linux operating system with RTXI without installing anything on your computer. This live environment allows you to mount your existing hard drive and you can conduct experiments and collect data. Note that the real-time performance will be slower compared with an actually installed system. Running the live environment from a USB flash drive is faster than from an actual CD. A DAQ card does not need to be installed for RTXI to run and RTXI can be successfully run on many laptops, including Intel MacBooks and MacBook Pros.

1.3 Introduction to Dynamic Clamp

Traditionally, the properties of electrically excitable cells are assessed using current clamp and voltage clamp electrophysiology protocols. In *current clamp*, an electrical current waveform is specified and applied to the cell

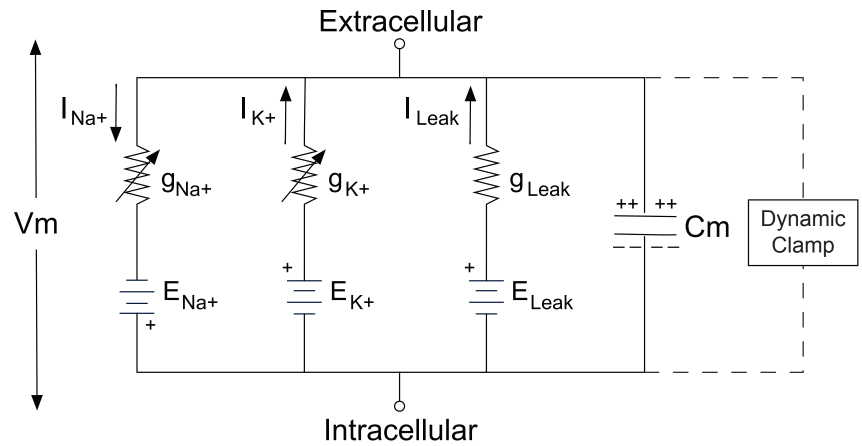

Figure 1.1: Equivalent circuit model of an excitable cell where V_m represents the transmembrane potential, C_m represents the membrane capacitance, and each conductance is defined by a reversal potential E_i and a conductance g_i. In this model, the voltage-dependent sodium and potassium ion channels are depicted as variable resistors where $g_i = g_i(V_m)$. The reversal potential (or equilibrium potential) of an ion is the value of transmembrane voltage at which diffusive and electrical forces counterbalance, so that there is no net ion flow across the membrane. The reversal potential is represented as a battery since the potential difference $V\mathrm{m} - E_i$ gives the driving force across the membrane for that ionic current.

through a microelectrode while the transmembrane potential is recorded. In *voltage clamp*, a desired voltage waveform is specified and analog circuitry is used to determine and inject a current that is necessary to maintain, or clamp, the membrane potential at the specified values. The *dynamic clamp* allows the insertion of artificial membrane conductances, such as ion channels, by injecting current that is a function of the cell's membrane potential. The injected current is computed by computer software or analog circuitry based on the equivalent circuit model of an excitable cell (Fig. 1.1). The artificial conductance is effectively in parallel with other membrane processes, each of which contributes to the total transmembrane current.

The total transmembrane current is related to changes in the membrane potential, V_m, through the following equation:

$$C_m \frac{dV_m}{dt} = -\Sigma I_i \tag{1.1}$$

where C_m is the membrane capacitance. Any conductance that can be described mathematically can be applied to a neuron using the dynamic clamp. The current passing through an ion channel, for example, is often described by Ohm's law using the following conductance-based equation:

$$I_i = g_i(V_m)(V_m - E_i) \tag{1.2}$$
$$g_i(V_m) = \bar{g}_i m^{\mathrm{p}} h^{\mathrm{q}} \tag{1.3}$$

where $\bar{g}$ is the maximal conductance, and m and h are voltage-dependent activation and inactivation gating variables (p and q are integers) that describe the kinetic activation of the channel. Gating variables have values between 0 and 1 to scale the channel conductance and are typically described by a first order differential equation:

$$\frac{dx}{dt} = \frac{x_\infty(V_m) - x}{\tau(V_m)} \tag{1.4}$$

The membrane potential is re-sampled and the equations are re-evaluated on every computational cycle of the dynamic clamp system. As such, the dynamic clamp is also sometimes termed *conductance clamp* or *conductance injection*.

While the dynamic clamp was first demonstrated in cardiac electrophysiology to electrically couple embryonic chick myocytes [32], the technique was independently introduced [24, 29] and is now more prevalent in neural electrophysiology [12, 13, 22]. Applications of this technique include the insertion of non-native ion channels (a virtual "knock-in"), subtraction of native ion channels (a virtual "knock-out"), and simulation of synapses and electrical gap junctions to create small networks of biological and/or simulated neurons. By varying the parameter values of a model channel or synapse, experiments can be conducted to determine how these properties shape membrane dynamics and neuron activity. These approaches have made the dynamic clamp a valuable tool for studying the intrinsic properties of single neurons and the behavior of small neural networks.

Dynamic clamp studies have also made important contributions to our understanding of neuronal dynamics under in vivo-like conditions in which neurons receive a constant barrage of synaptic inputs which can easily reach thousands of events per second. Artificial synaptic input can be constructed from pre-recorded activity of presynaptic neurons but is more commonly based on statistical descriptions of noisy conductance waveforms. This high conductance state has been shown to enhance the cell's responsiveness to small inputs, also known as its gain [5, 10, 28], and can change the signal integration of synaptic input, creating distinct modes of firing patterns [9,26,30,33]. The statistics of current-based versus conductance-based input, such as correlations and relative balance between excitation and inhibition, are translated differently into output statistics such as the membrane potential distribution, the distribution of interspike intervals (ISIs), the coefficient of variation (CV), and the mean and variance of the neuron's output firing rate [17, 25, 27, 31]. Together, these factors result in a dynamical behavior of the neuron that is usually quite different from the intrinsic dynamics of the voltage-gated currents.

Results from dynamic clamp experiments must be carefully interpreted due to several experimental limitations. Space-clamp problems arise in that the injected current is limited to a space around the recording electrode. In some experimental studies, an artificial dendrite is modeled as well to simulate the cable effects of synaptic inputs propagating to the action potential initiation zone [15]. Since current can usually only be injected at the soma, the dynamic clamp may be a poor approximation of dendritic input in some cell types. In most cells, the dynamic clamp is operated in discontinuous current clamp (DCC) mode in which a single electrode switches between recording and current injection states. In this configuration, it is not possible to inject large conductances that approach the magnitude of the cell's intrinsic resting conductance while still accurately recording the membrane potential. The injected current induces a voltage drop through the electrode and causes measurement accuracies that are propagated through the closed feedback loop in the dynamic clamp system and may cause ringing artifacts in the recording [2, 16, 20, 21]. In larger cells, two electrodes may be used, one to record and one to inject current. Researchers also typically use the same ion channel or synapse model parameters for all cells used in an experiment, assuming that neurons of the same cell type have identical intrinsic properties, both within an animal and between animals [14]. There usually is not time during an experiment to manually adjust the model to optimal parameters for each cell.

Compared to other real-time closed-loop experimental protocols, the dynamic clamp has perhaps the most stringent performance requirements. These limitations involve numerical, algorithmic, and hardware platform-specific issues. Dynamic clamp performance depends on how accurately the model is solved, measurement error in sampling the voltage, and the sampling rate of the system. The sampling rate determines how much time is in a given computational cycle for various operations to be performed, and the duration of the cycle restricts the types of numerical methods that may be used to integrate the gating variables. Thus, the computational performance of dynamic clamp suffers from a trade-off between the speed of computation and numerical accuracy. Dynamic clamp sampling rates are currently chosen based on the limits of the hardware platform being used and the temporal dynamics being simulated. While it is possible to compute the time step necessary for the Euler and exponential Euler methods to achieve a desired one-step integration accuracy for a known voltage measurement error, few studies employ this technique [3]. In simulations of dynamic clamp, Euler integration was insufficient to model fast sodium Na_v channels at sampling rates under 30 kHz and nearly identical integration results for three different deterministic integration methods was only achieved at rates $\geq$50 kHz [18]. Standard performance benchmarks are needed for dynamic clamp to justify the sampling rates that are used.

Other hardware that are typically required for dynamic clamp are an electrophysiology amplifier for measuring membrane potential and injecting current and a multifunctional data acquisition system (DAQ) for performing analog-to-digital (ADC) and digital-to-analog (DAC) conversion. The technical specifications of each of these hardware components can affect the performance of the overall system by introducing additional jitter, latency, and quantization error that can affect system timing and the numerical computation [1, 4]. Recent results show that faster systems would result in a greater range of conductances that could be utilized, improved stability, and more accurate real-time model simulations [20, 21]. Faster dynamic clamp systems have been developed, largely due to the increasing power of personal computers, but also due to the development of systems based on the GNU/Linux operating system and embedded real-time processors.

2 RTXI Features

Hard real-time platform RTXI provides a platform for data acquisition and custom control paradigms involving a variety of hardware and software algorithms. It runs on a hard real-time Linux operating system (OS), which guarantees reliable timing for periodic tasks such as sampling from experimental equipment, performing computations, and generating external signals. This differs from platforms based on general purpose operating systems that assign different scheduling priorities to tasks based on optimizing the user experience in a multitasking environment. For closed-loop applications such as dynamic clamp that run at very high sampling rates, it is important that data is sampled at a precise time and that all computations are completed in time for the next scheduled feedback input. Operating systems that can absolutely guarantee a maximum time for these operations are referred to as hard real-time, while operating systems that can only guarantee a maximum most of the time are referred to as soft real-time. A soft real-time system can handle such lateness, usually by pausing processes with a lower priority. For dynamic clamp, a soft real-time system may occasionally wait so long to compute the injected current, that the actual value of the membrane potential has changed in the meantime. In that case, the simulated ion channel (or other membrane conductance) is based on incorrect assumptions about the state of the system. Real-time Linux also maximizes the sampling rate that can be used by minimizing system latencies related to the hardware and analog-to-digital conversion (and vice versa). For some experimental designs with closed-loop feedback, a higher sampling rate improves the stability of the protocol.

Modular architecture Users can quickly implement complex experimental protocols in RTXI, including both open and closed-loop control modes as well as many data acquisition modes. This is accomplished by RTXI's unique architecture in which system features and custom user code are implemented as modules. Modules contain function-specific code that can be used in combination to create larger workflows. They communicate with each other within the RTXI workspace by a system of *signals and slots* and event handling. All data acquired through a DAQ card are preserved as signal streams that can be passed to other modules that implement real-time analyses such as event detection, digital filters, etc. Similarly, all user modules can accept input signals and generate output signals that can be connected to other modules or to a DAQ card to produce external analog or digital signals. In the following example of a RTXI workflow for a dynamic clamp protocol, the recorded membrane potential from a cell is connected to a module that contains a model of an ion channel. The output of the module is the computed current, which is based on the value of Vm. The output signal from the ion channel module is connected to the DAQ card so that it can be injected back into the cell. The Oscillocope can plot any signal in the workspace, including actual outputs of a model as well as internally computed state variables.

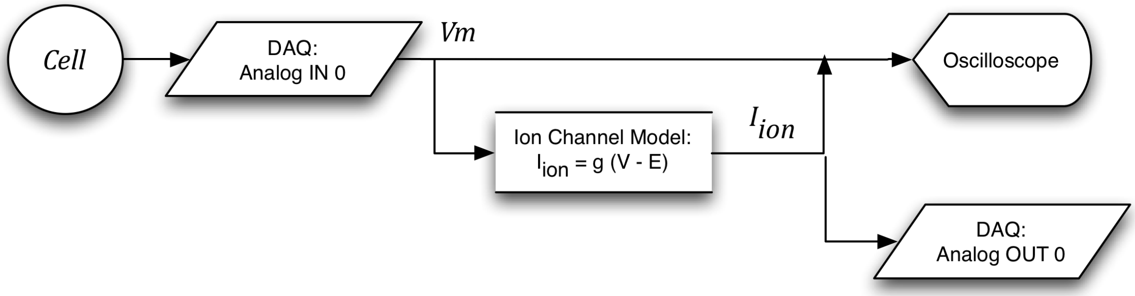

Figure 2.1: A user module computes a current based on a model of an ion channel and the acquired value of the cell's membrane potential.

This modular signal-and-slots architecture allows multiple instantiations of user modules and all modules accept one-to-many and many-to-one connections. For example, a single module output signal can be routed to multiple targets. If multiple signals are connected to a single target, or slot, the signals are summed before being used in any further calculations within the module. This framework makes it easy to reuse code and implement branching logic. It is similar to graphical programming frameworks used in MATLAB™ Simulink® and LabVIEW™ . In the following example, a biological cell is reciprocally coupled through artificial synapses to a model neuron simulated within RTXI. The synapse is triggered by a presynaptic action potential and the synaptic current is modeled using a conductance-based equation that also depends on the measured membrane potential of the presynaptic neuron. In Figure 2.2, the membrane potential is split into two streams. One is sent to a Spike Detector module, which generates a trigger signal for the Synapse Model module. The other branch is sent directly to the Synapse Model module to be used in the calculation of the synaptic current. Each of these modules is instantiated twice since the two neurons are reciprocally coupled. Furthermore, each instantiation operates completely independently and can have different parameter values. This modular approach makes it very easy to experiment with asymmetric coupling, where one synapse is stronger than the other, by using different values for the maximal conductance or the reversal potential of the synapse.

This modular architecture also allows RTXI to be used solely as a experimental control system or as both a control system and data acquisition system. RTXI can be installed on most personal desktop and laptop computers and it is possible to run RTXI without a data acquisition card. This allows users to develop modules and online algorithms on one computer and copy their module to the actual computer used for experiments.

Changing parameters on-the-fly

If you choose to change modules during an experiment, e.g. to change spike detection algorithms or use a different synaptic model, it is as simple as loading the new module and adding the connections. Other popular platforms for real-time closed-loop data acquisition may require you to recompile your program or re-download it onto a dedicated real-time processor. Similarly,

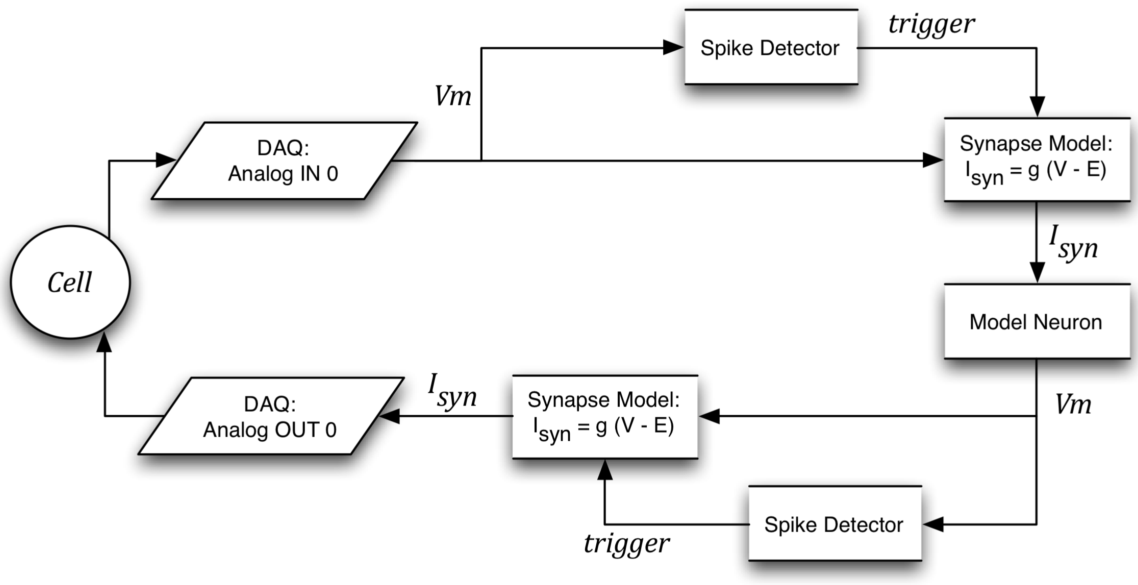

Figure 2.2: Two user modules, a Spike Detector and a Synapse Model, are used to reciprocally couple a biological neuron with a model neuron. Each module is instantiated twice and can have different parameter values.

an important feature of RTXI is the ability to change experimental parameters on-the-fly without recompiling or stopping real-time execution. This is accomplished through an intuitive GUI interface.

Custom algorithms

→ Chapter 4
Custom User Modules

RTXI and all core system features are written in C++ and are based on the Qt GUI framework. Users implement custom experimental protocols by writing their own modules, in either C++ or the DYNAMO scripting language. User-contributed modules are available on the website (http://www.rtxi.org) and examples are included in the RTXI source code. There is virtually no limit to what can be implemented in RTXI. Users can link to C++ libraries of their own or that others have created to leverage complex computations or algorithms. Custom protocols can include not only unique online or real-time analysis but also unique experimental *control* paradigms. Examples include the automation of protocols by automatically changing parameter values or using event detection to trigger a new sequence of experimental perturbations.

The flexibility of RTXI through custom C++ modules gives it features that are not common to other data acquisition systems, especially those for electrophysiology experiments. For example, RTXI is a complete simulation platform that can be used to solve systems of differential equations in real-time and integrate biological signals acquired in real-time with model systems. This was demonstrated in Figure **??**. Modules can also be written to load data from external files. RTXI has the ability to “play back” this previously acquired data or surrogate data as if it were being acquired in real-time. This is accomplished by using a module that generates an output signal based on this surrogate data. Thus, online algorithms in development can be thoroughly tested and debugged using actual real-time execution without using the resources and time required for setting up an actual experiment. There are several advantages of this approach over simulating real-time computation offline using other platforms. First, it automatically takes into account the computing overhead associated with the actual RTXI system and gives a more accurate picture of your real-time performance. It also eliminates any redundant programming tasks involved with migrating code between platforms.

Custom data acquisition

By default, RTXI runs *continuous* protocols in that both the Data Recorder and digital Oscilloscopes modules act primarily as chart recorders. In addition, modules continuously execute their real-time code until they are paused. More complex sequences of operations can be accomplished by programming different operating modes within a single module. It is also possible to write a module that programmatically starts and stops other user modules. These "parent" or "controller" modules can be used to construct complex sequences or hierarchies of experimental stimuli. *Sweep-based* or *trial-based* recordings are implemented by a module that instructs the Data Recorder to increment trials. In addition, a module that generates a timed trigger signal can be used along with the trigger feature of the Oscilloscope to align recorded data in time. Examples of all these methods of implementing complex control paradigms are available.

Metadata capture

RTXI can be used in parallel with your choice of data acquisition software. However, RTXI's ability to capture important metadata about an experiment is only present if it is also used for data acquisition. This is accomplished through the Data Recorder system module which streams the acquired data along with any computed signal to a Hierarchical Data Format (HDF5) file. HDF5 is an open data model that is increasingly popular for representing complex data, data relationships, and their associated metadata. In RTXI, any module that generates data that is being saved to HDF5 also has its parameters and parameter values stored in the same file. This includes not only acquired data, but also any computed intermediate signal, which is valuable for debugging algorithms offline. When a parameter value is modified on-the-fly during data acquisition, the new value and a timestamp for the modification is also saved. Other system parameters are also automatically saved to the HDF5 file so that important metadata is always adjacent to the actual experimental data. RTXI modules also exist for explicitly saving comments or creating custom experimental logs to capture any additional information. Such user modules can be used to standardize experimental logs by providing templates for information that the user is expected to supply or a finite set of options the user can choose from.

Portability

Once a protocol has been created by making connections between modules and setting parameter values, the entire workspace can be captured to a settings file. Reloading this file will restore all system and user module settings, as well as the layout of the windows on the screen. This reduces the chances of errors when setting up a complicated experiment. These settings files can be transferred between different computers as long as the target computer contains the modules that were used. This is independent of the actual experimental equipment that you use for data acquisition and/or cur-

→ Chapter **??**
COMEDI support

rent injection. RTXI uses the open source COMEDI drivers, which provides a generic interface to your choice of hardware. Users should check that they are using the correct hardware driver and configure their channel gains such that RTXI modules are receiving values in the correct units. The settings file uses a simple XML-based syntax and can be opened in any generic text editor to manually edit the default parameter values.

→ Chapter 3.8
HDF5

HDF5 files are also compatible with many commercial and free software for a variety of platforms. There is no required proprietary software for viewing or analyzing data stored in RTXI-generated HDF5 files. Much of the available software also support editing data in place within the HDF5 file or appending new data to an existing file. This allows users to add associated data such as images, post-processed data, or additional notes.

2.1 RTXI Menus

Although RTXI is dependent on Linux, it is a complete desktop application and configuration of system settings and interaction with most features are available through a graphical user interface.

The **File Menu** is used to save and load settings files that capture the entire working environment. This includes settings configured in the System Control Panel, such as channel gains, parameters set within modules, and connections between modules. Reloading a settings files will also restore the window sizes and positions at the time the file was created. Recently used settings files are also available from this menu.

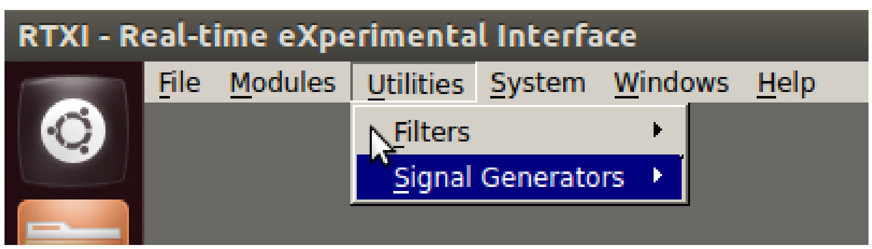

Figure 2.3: The File Menu is used to save and load settings files that help you capture and recreate your working environment.

The **Modules Menu** is used to load user modules or compile DYNAMO model descriptions. User modules are shared libraries that are installed to `/usr/local/lib/rtxi` during the compilation process. Choosing "Load User Module" opens a file dialog box at this location, from where users may select modules based on the *.so filename. Choosing "Load DYNAMO Model" will open a file dialog box that can be used to select a *.dynamo file containing a DYNAMO model description. This file will be parsed to generate a C++ header and implementation file that will then be compiled to produce an RTXI module based on the `DefaultGUIModel` class. After the initial parsing and compilation, this module may then be loaded using the first menu item as with other user modules.

The **System Menu** is used load core RTXI system features and modules. The Control Panel is used to configure channels and set the nominal system period (or sampling rate). The Oscilloscope is the digital oscilloscope

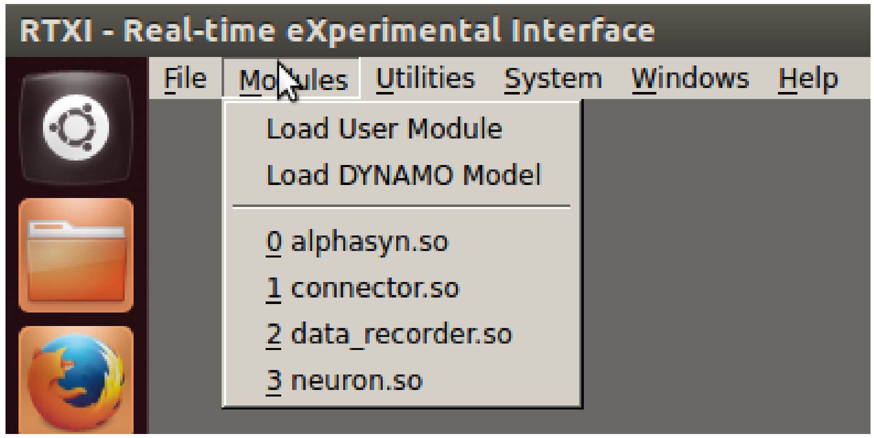

Figure 2.4: The Modules Menu is used to load user modules and compile DYNAMO models into RTXI objects.

that can be used to plot any signal in real-time. The HDF Data Recorder allows users to select signals to stream to a HDF5 file in real-time. The Connector is used to specify connections between modules and the DAQ card. The Performance Measurement module displays running statistics on the computational load currently used by RTXI and how it compares to the nominal system period, or the amount of time available for performing the computations. There is also a Preferences panel for specifying commonly used directories, etc.

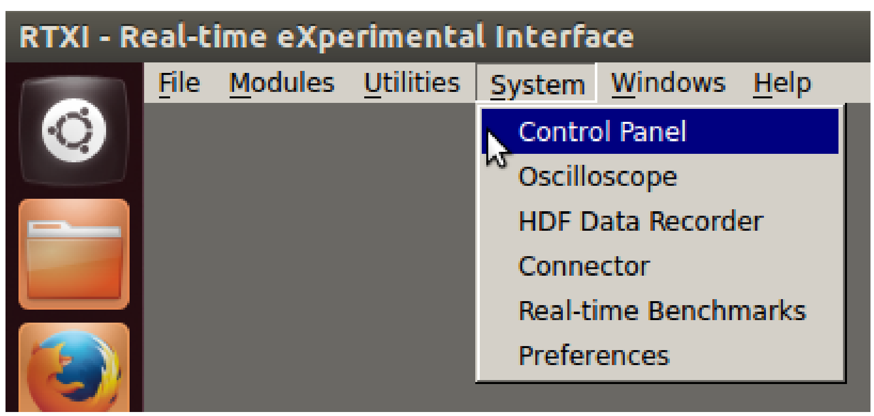

Figure 2.5: The System Menu is used to access system settings and other core RTXI tools.

The **Help Menu** contains important information about your system and RTXI. Choosing the "What's This" menu item turns the mouse cursor from a pointer into a question mark. In this mode, clicking on any module will display a window with contextual help, if available, about that module.

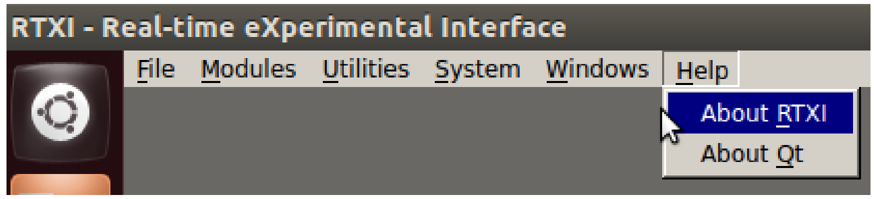

Figure 2.6: The Help Menu gives users access to contextual help for other modules and information about RTXI, COMEDI, and Qt.

2.2 Core System Modules

The following modules are included in RTXI by default and are available through the **System** menu. No additional modules are necessary for acquiring data through a DAQ card and saving data to an HDF5 file.

2.2.1 System Control Panel

The System Control Panel allows you to set important parameters on all the input and output channels of your DAQ card and set the nominal real-time period of your system. RTXI automatically detects the manufacturer and board names of available DAQ cards and the number and type of input and output channels. The first DAQ card installed in your system is assigned the Linux device name: `/dev/comedi0`. Additional DAQ cards are assigned device names `/dev/comedi1` and so on.

→ Chapter 3.3
Configure RTXI for more cards

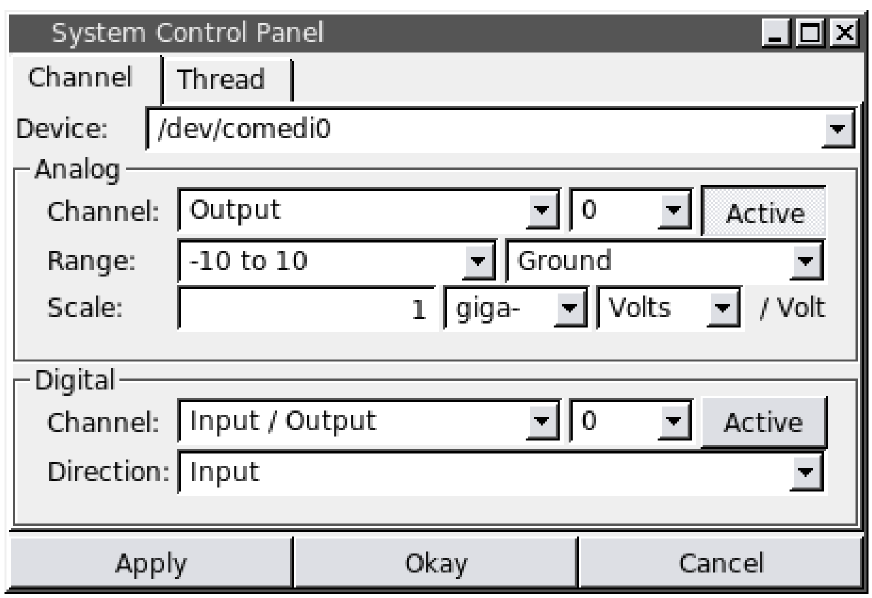

Figure 2.7: The Channel Tab allows users to activate and set channel properties on the DAQ card.

Channel Tab

By default, RTXI assumes that all analog DAQ input and output channels have a range from -10 V to +10 V, unity gain (or a scaling of 1 V/V), and a ground reference. You must set the correct options for each channel you are using to acquire and output the correct signal values. In the screenshot below, the DAQ card is being used as a signal generator on the first analog output channel (Channel 0). This channel is connected to an external amplifier that specifies a gain of 1 nA/V. This gain is inverted here to 1 gigaVolt/Volt to output the correct values and can be verified on an external oscilloscope. For your own reference, you could set the dropdown boxes to read 1 gigaAmp/Volt to indicate that you are ultimately generating a current signal, but this setting does not affect the computation. **!** → You must click the **"Active"** toggle button to actually activate the channel. You must click **"Apply"** to commit changes made to any other channel properties such as the scale. When you have set the correct parameters for all your input and output channels, you can save your settings by choosing **File→Save Workspace** from the RTXI menu bar. This will create a basic *.set file that you can load in the future to recreate this environment or use as a foundation for additional *.set files.

Thread Tab

→ Chapter 4.1.6
`DefaultGUIModel` PAUSE flag

In the Thread tab, you can change the real-time period of the entire system. You must click "Apply" for this change to take effect because it triggers a real-time event that is propagated to other modules. Custom user modules can be programmed to execute specific code when the system's real-time period is changed.

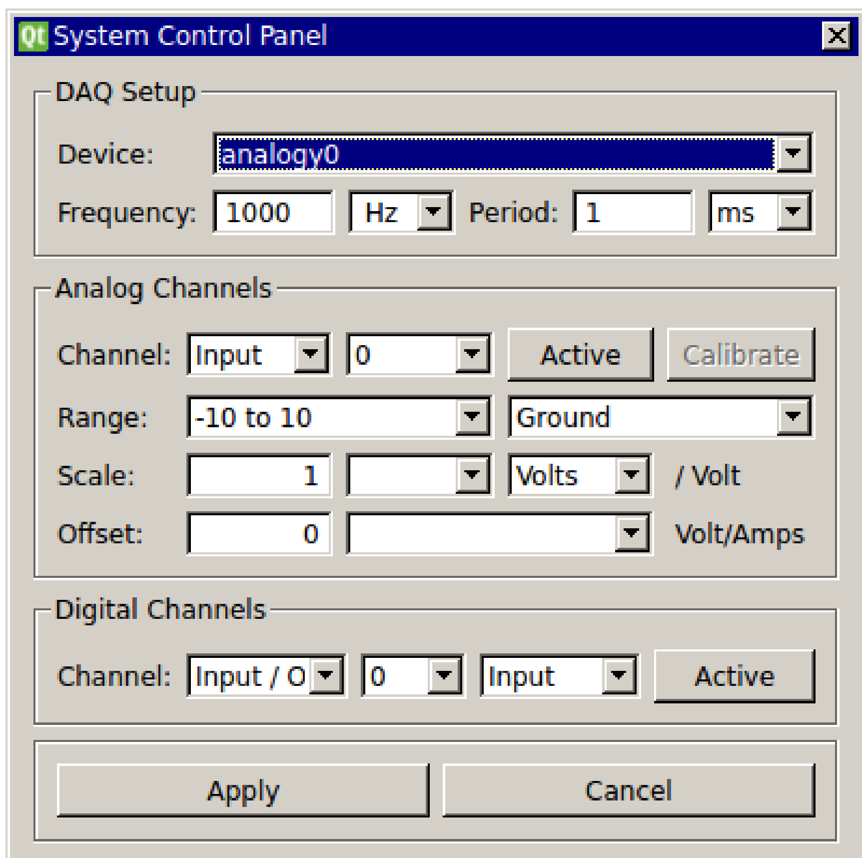

Figure 2.8: The Thread Tab allows users to change the period for real-time execution.

2.2.2 Oscilloscope

The Oscilloscope allows you to plot any system signal in real-time, including signals from/to the DAQ card and signals from user modules. To plot multiple signals, you can instantiate multiple oscilloscopes or superimpose multiple signals on a single oscilloscope. Each signal may have a different vertical scale and line style and a legend is automatically generated in the Oscilloscope window. Below is a screenshot of the Oscilloscope acquiring data using the CLAMP-1U model cell (by Molecular Devices) with an applied square pulse current. It is plotting two input channels from the DAQ card as well as the command current (Iout1) generated by a module. The lower ! → right-hand corner displays the time scale for each grid division. The Oscilloscope uses a right-click context menu that allows you to pause/unpause real-time plotting or access the "Properties" panel for choosing signals and setting the axes properties.

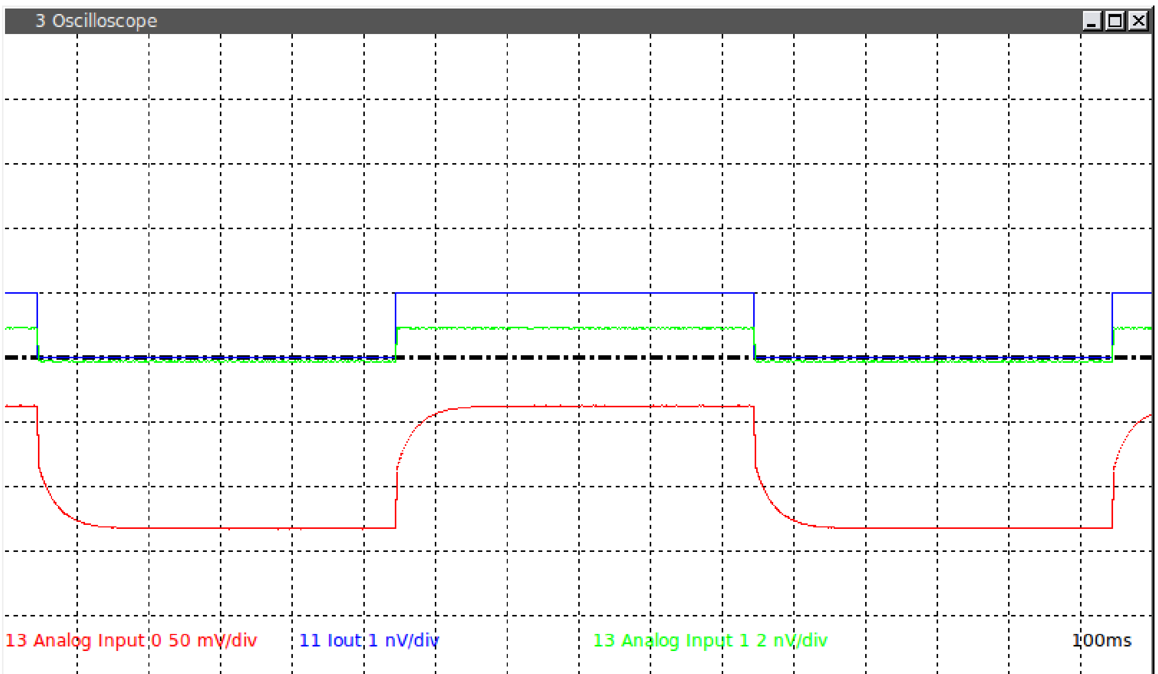

Figure 2.9: The Oscilloscope can plot multiple signals in real-time.

Channel Tab

On the Channel tab, you can select signals to plot. Signals from the DAQ card will appear in the "Channel" dropdown box as a COMEDI source, eg. `/dev/comedi0`. To get correct values for your signal, you may have to set additional settings in the System Control panel if you use any other instrumentation that applies a gain to your signal. Input to the DAQ card from external instrumentation, such as an external amplifier, is designated as output from the DAQ card within RTXI. RTXI will automatically detect how many input and output channels your card has. Signals from other modules will be identified by the module name and you can choose from

→ Chapter 2.2.1
System Control Panel

any inputs, outputs, parameters, or states that are defined in those modules. ! → To actually plot the signal, you must depress the **"Active"** toggle button and hit **"Apply."** Any modifications you make to the scaling, offset, or line style of the signal are not active until you hit "Apply" again. Note that plotting too many signals in real-time may affect your system performance and cause your GUI to freeze during program execution.

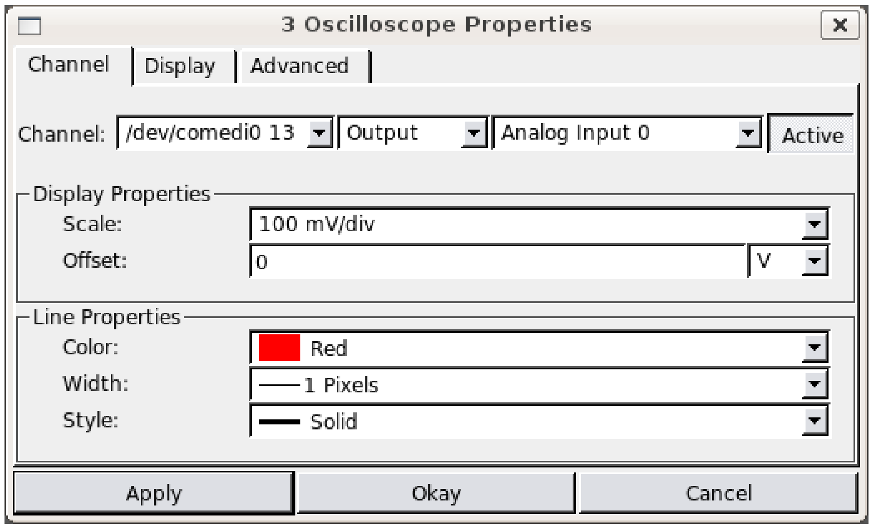

Figure 2.10: The Channel Tab allows you to select the signals to plot and set line styles.

Display Tab

On the Display tab, you can set the time scale of the oscilloscope and the screens refresh rate. You can also set up a trigger to freeze the oscilloscope when certain criteria are met by the triggered channel. Set the trigger to operate on a “rising edge” or “falling edge” by clicking the radio buttons marked “+” or “-”, respectively. The “Trigger Channel” can be set to any signal that is currently plotted on the oscilloscope. “Trigger holding” allows you to set the amount of time that lapses before the trigger is reset again. The trigger threshold is indicated on the oscilloscope by a horizontal yellow dashed line.

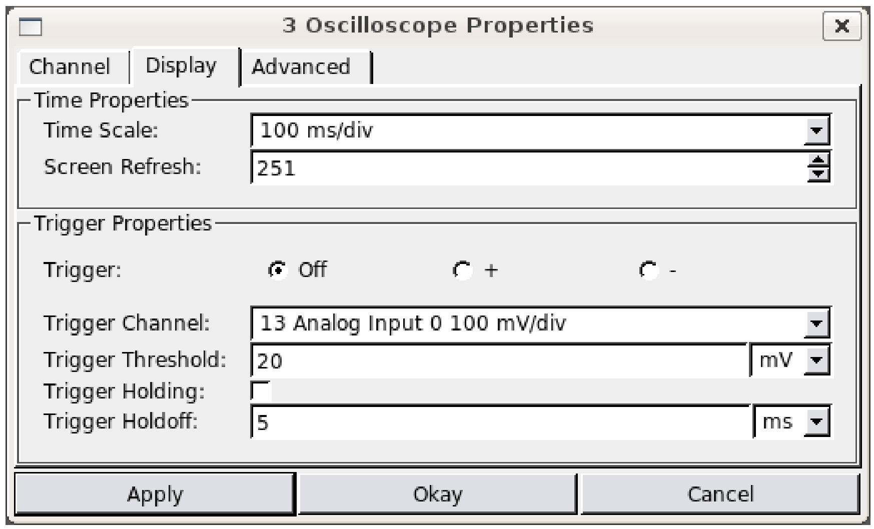

Figure 2.11: The Display Tab allows you to set the time scale for horizontal divisions and set a trigger on any plotted signal.

Advanced Tab On the Advanced tab, you can choose to downsample the data that is plotted on the oscilloscope or change the number of grid divisions used for scaling.

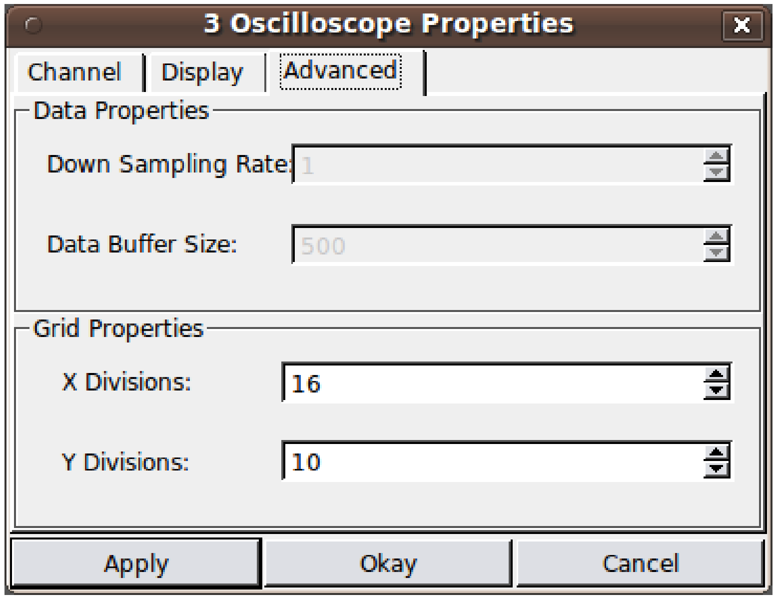

Figure 2.12: The Advanced Tab allows users to set downsampling rates on the plotted data, change the size of the data buffer, and change the number of divisions in the oscilloscope.

2.2.3 Data Recorder

The Data Recorder module allows users to stream synchronous data to a HDF5 file. To record data, select **System→Data Recorder** from the RTXI menu bar. The **“Block”** menu is a list of your DAQ card(s) and any loaded user modules. Selecting a block device then populates the **“Type”** and **“Channel”** menus. Select the Analog Input 0 channel from your DAQ device and click the “>” button. To remove a channel from the list, highlight it in the listbox and click the “<” button. Before you can start recording, you must select a file by clicking the **“Choose File”** button. Click **“Start Recording”** to begin recording and **“Stop Recording”** to stop recording. For each module connected to the Data Recorder, it also grabs all the parameters values and saves them as metadata. In addition, it logs when any of these parameter values change so that you have a complete record of your experiment. If you have a configuration such that Module A: Output 0 –> Module B: Input 0, saving both of these respective signals in the Data Recorder will give you exactly the same data. Similarly, saving `/dev/comedi0`: Analog Output 0 will save the signal that has been assigned to that channel (perhaps generated by a user module), not the actual signal the DAQ card is outputting. To check the actual output of the DAQ card, you will need to make a connection from that output to another input channel.

For real-time streaming of multiple signals, an HDF5 data type is used in RTXI that does not map efficiently onto MATLAB native data types. While

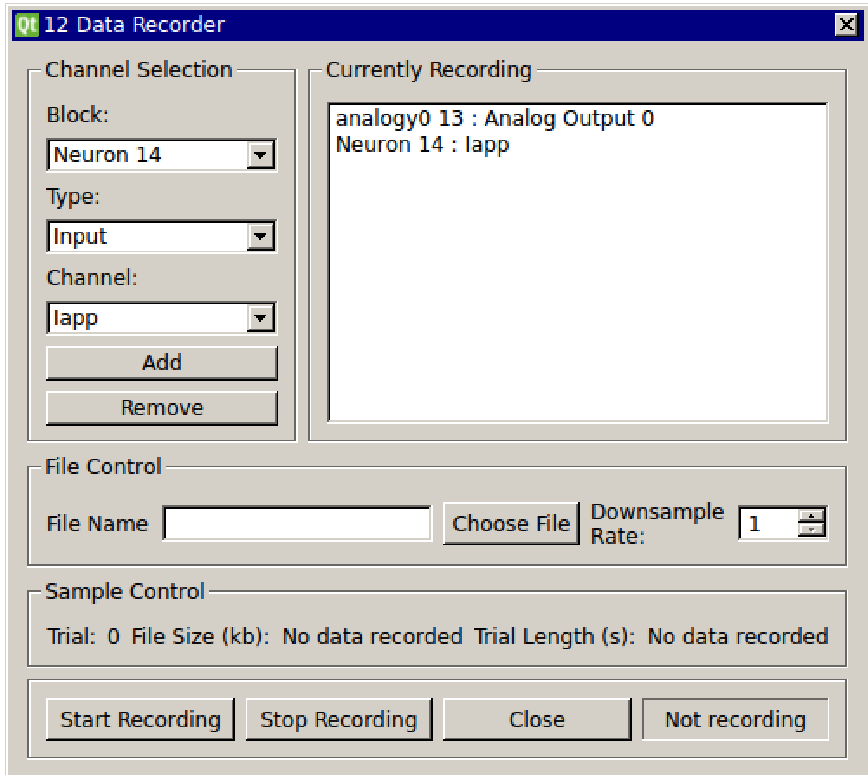

Figure 2.13: The Data Recorder allows you to save any signal in your workspace synchronously to an HDF5 file. Use the ">" and "<" buttons to add and remove signals from the list.

MATLAB can read this data using its low-level functions, this process can be very slow. To load RTXI HDF5 files quickly into MATLAB, you will first need to run a small utility function on your HDF5 file to convert the *Synchronous Data* dataset to a different data type. This function is compiled when RTXI is compiled and is located in `/rtxi/hdf`. To convert your HDF5 file:

```
$ rtxi_hdf_matlablize YOUR_FILE.h5
```

To make this utility accessible from any directory on your system, make a symbolic link in `/usr/bin` to the location of this function in your RTXI source directory. If you installed RTXI from the Live CD, the source directory is `/home/rtxi`:

```
$ sudo ln -s RTXI_SRC_DIR/hdf/rtxi_hdf_matlabize
      /usr/bin/rtxi_hdf_matlabize
```

RTXI also includes a simple MATLAB GUI for quickly viewing the data within a single trial. The MATLAB code is available in `/rtxi/hdf/RTXIh5`. A sample m-file is provided with examples of how to extract data to the MATLAB workspace, how to use the GUI browser, and how to add new datasets to your file. It is also possible to embed binary formats, such as images, within a trial.

2.2.4 Connector

The Connector module allows you to create connections between modules or between modules and the DAQ card. Incoming signals to RTXI through the DAQ card appear in the **"Output Block"** and outgoing signals through the DAQ appear in the **"Input Block."** RTXI automatically detects how many inputs and outputs are available for your installed DAQ card. Similarly, any signal that is defined as an output of a module appears in the "Output Block" and any input slot of a module appears in the "Input Block." After you have made your selection, click the central toggle button to activate the connection. Your active connections are listed in the "Connections" box. To quickly turn off an existing connection, double-click on its entry in the table and click the toggle button. Below is a screenshot of how to connect the command current from the Istep module to the first analog output channel of the DAQ card.

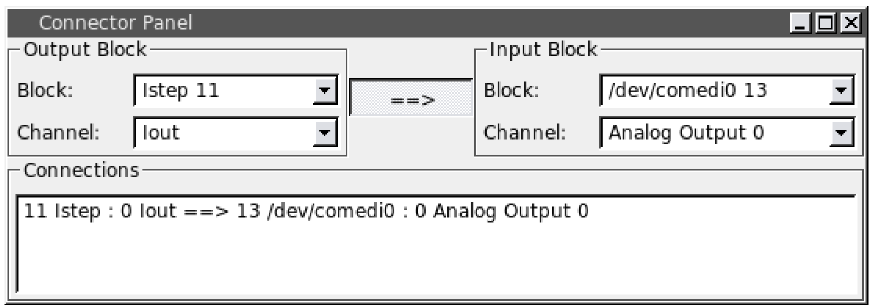

Figure 2.14: The Connector allows you to connect any signal from the left "Output Block" and connect it to a slot in the right "Input Block". RTXI automatically detects the available signals and slots from the DAQ card and any loaded user modules.

Here is a screenshot containing connections between modules only. The membrane potential of a model neuron is fed into a spike detector that in turn, informs a dynamic clamp module when spikes have occurred. The model cells voltage is also fed into the dynamic clamp module for computing

! → the current that is connected back to the model cell. The signals-and-slots architecture of RTXI allows any signal to be connected to any slot. In this example, the spike detector could accept input from any model neuron simulated within RTXI or input from an actual recorded cell. RTXI also allows one-to-many and many-to-one connections. If multiple signals are connected to one slot, the signals are first summed before any additional operations are performed. Thus, multiple signals could be connected to be a single output channel of your DAQ card, allowing you to "stack" stimuli being generated from multiple user modules.

2.2.5 Performance Benchmark

The Performance Benchmark module gives you timing statistics for RTXI. For hard real-time performance, it is important that all operations, computations executed by user modules and tasks related to data acquisition, etc., complete within the nominal system period. This module continuously keeps track of the time needed to complete these tasks, updated once every

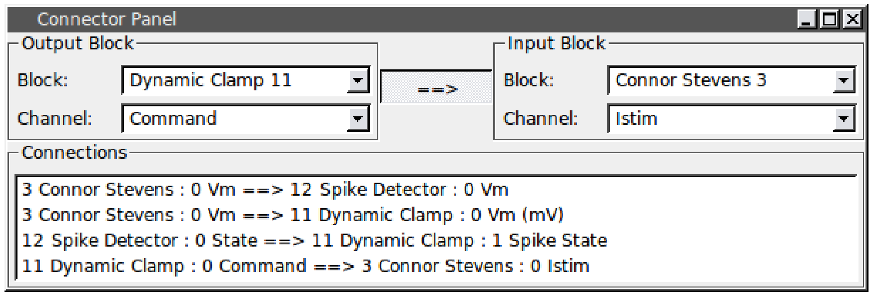

Figure 2.15: The signals-and-slots architecture of RTXI allows any signal to be connected to any slot and allows one-to-many and many-to-one connections.

second in the GUI, as well as the actual real-time period. In addition, the module reports the worst case total computation time and the worst case time step since the statistics were last reset.

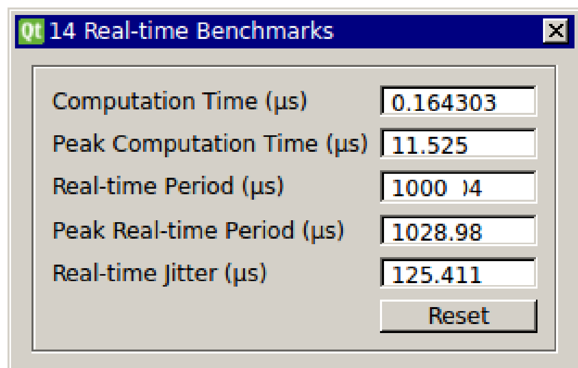

Figure 2.16: The Performance Benchmark module gives you timing statistics related to computational tasks and the actual real-time period (system sampling rate).

3 Getting Started

3.1 RTXI Software Installation

There are two methods for installing RTXI on your computer:

1. **Easy:** Booting off an RTXI Live USB.
2. **Hard:** Compiling a real-time Linux operating system and then installing RTXI.

We suggest that users new to Linux use the Live USB. As of v2.0, we support installations built on Ubuntu 12.04. Ubuntu is the most popular desktop-oriented Linux distribution and has an extensive online support community. If you choose to compile the operating system yourself, you can choose any Linux distribution, but your should be prepared to address issues during installation. All Linux distributions are based on the same Linux "kernel," but they differ in their default utilities, filesystem hierarchies, and libraries. In general, these issues can be address across distributions.

Note that manual compilation can result in better real-time performance, particularly when compiling for a specific processor family rather than a generic type.

! → The current Live USBs available on the RTXI website (http://www.rtxi.org) are configured to handle processors with either one or two cores. You may experience issues if your system has more cores. If this causes RTXI to not start off the Live USB, check your kernel log by by running:

```
$ dmesg
.
.
[ 390.069252] RTAI [hal]:  RTAI CONFIGURED WITH LESS THAN
   NUM ONLINE CPUS
```

If the error matches the one here, you will need to start Ubuntu with fewer cores enabled. Press "E" when you see the GRUB bootloader menu to edit the boot command, ant at the end of the line beginning with "`boot`", add the flag `maxcpus="X"` where X is the number cores you want to use. The, to boot, use the keyboard shortcut CTRL-X. Note that this modification is not permanent and you will need to do this step every time you restart the computer. It is possible, though, to permanently edit your GRUB menu with this flag when you are booted into your system.

3.2 Hardware Requirements

RTXI is designed to run on a standard personal desktop computer. Uniprocessor, multi-core processors, and multiple CPUs with and without hyper-threading are supported by Linux and RTXI. Xenomai must be correctly configured for multiple cores to be used. More stable systems are typically realized with Intel processors rather than AMD processors. In rare cases, a particular CPU and motherboard combination is not supported. Certain advanced motherboards may contain features that are not compatible with Xenomai. For example, some that use integrated graphics chips use hardware techniques to speed up computation that are also not compatible. An external video card is recommended and NVIDIA cards tend to have better Linux support than ATI/Radeon cards. This can be important since the greatest overhead in RTXI is related to data visualization in the oscilloscope. For newer graphics cards, you may need to manually install the Linux drivers, which are usually available on the manufacturer's website. Some systems may also include BIOS level or hardware interrupts that are not captured by Xenomai or advanced power management features (sometimes these can be turned off by the user in the BIOS).

The real-time Linux kernel has extremely low latencies and little software overhead. RTXI is also designed so that the minimum number of dynamic modules can be loaded at any time. RTXI has been successfully tested on computers with Pentium III processors up to 6-core Intel Xeon processors. While the processor speed allows RTXI to complete more computations within a single real-time cycle, the amount of RAM and the amount of video memory have a significant impact on the stability and speed of the system. Users should also consider high speed hard drives, large cache sizes, and high speed bus interfaces. If you are purchasing an off-the-shelf desktop computer system and plan to add a DAQ card, be sure that your power supply is powerful enough to handle the extra load. At least a 450W power supply is recommended.

3.3 Data Acquisition Cards

For closed loop experiments using RTXI, your computer must be equipped with an analog-to-digital converter (ADC) to acquire data and a digital-to-analog converter (DAC) to generate signals. Of course, external hardware such as an oscilloscope or function (waveform) generator can be used in conjunction with RTXI. A popular solution is to purchase a commercial multifunction data acquisition card that provides analog input and output, digital input and output, and counter/timer circuitry. DAQ cards using **!** → the USB interface are *not compatible* with RTXI since the USB drivers in Linux are not capable of hard real-time operation. Furthermore, the USB interface can only achieve a maximum sampling rate of approximately 1 kHz, which may be sufficient for some closed-loop real-time applications but not for dynamic clamp. Many DAQ cards using the PCI, PCI express, or PXI interface are available from a variety of manufacturers. Your choice of DAQ card should depend on the number of analog and/or digital channels that you need, the amount of data resolution (eg. 12, 16-bit), the amount of sampling resolution (determined by the speed of the card), and whether you need simultaneous sampling (rather than sequential sampling) of multiple input channels.

! → Most RTXI users use products developed by National Instruments. A complete list of COMEDI supported DAQ cards is available at http://www.comedi.org. COMEDI also provides low-level drivers for cards using a 8255 chip, which provides three channels of 8 bit digital input or output, and for standard PC parallel ports. A list of currently supported NI cards and the corresponding COMEDI driver name is given in Table 3.2. A list of other COMEDI supported DAQ manufacturers is given in Table 3.1.

Table 3.1: DAQ Manufacturers with COMEDI supported Hardware

ADLINK	http://www. adlinktech.com
Advantech	http://www.advantech.com
Amplicon	http://www.amplicon.com
Data Translation	http://www.datatranslation.com
Fastwel	http://www.fastwel.com
General Standards Corporation	http://www.generalstandards.com
ICP	http://www.icpdas-usa.com
Intelligent Instrumentation	http://www.instrument.com
Keithley Instruments	http://www.keithley.com
Measurement Computing	http://www.mccdaq.com
National Instruments	http://www.ni.com/dataacquisition

Table 3.2: COMEDI supported National Instruments DAQ cards

Device	**Driver**	**Device**	**Driver**
AT-MIO-16E-1	ni_atmio	PCI-MIO-16XE-50	ni_pcimio
AT-MIO-16E-2	ni_atmio	PCI-MIO-16XE-10	ni_pcimio
AT-MIO-16E-10	ni_atmio	PCI-MIO-16E-1	ni_pcimio
AT-MIO-16DE-10	ni_atmio	PCI-MIO-16E-4	ni_pcimio
AT-MIO-64E-3	ni_atmio	PCI-6014	ni_pcimio
AT-MIO-16XE-50	ni_atmio	PCI-6030E	ni_pcimio
AT-MIO-16XE-10	ni_atmio	PCI-6040E	ni_pcimio
AT-AI-16XE-10	ni_atmio	PCI-6031E	ni_pcimio
		PCI-6033E	ni_pcimio
PCIe-6251	ni_pcimio	PCI-6071E	ni_pcimio
PCIe-6259	ni_pcimio	PCI-6023E	ni_pcimio
		PCI-6024E	ni_pcimio
PXI-6030E	ni_pcimio	PCI-6025E	ni_pcimio
PXI-6040E	ni_pcimio	PCI-6034E	ni_pcimio
PXI-6025E	ni_pcimio	PCI-6035E	ni_pcimio
PXI-6281	ni_pcimio	PCI-6036E	ni_pcimio
PXI-6711	ni_pcimio	PCI-6052E	ni_pcimio
PXI-6713	ni_pcimio	PCI-6070E	ni_pcimio
PXI-6071E	ni_pcimio	PCI-6110	ni_pcimio
PXI-6070E	ni_pcimio	PCI-6111	ni_pcimio
PXI-6052E	ni_pcimio	PCI-6143	ni_pcimio
PXI-6733	ni_pcimio	PCI-6220	ni_pcimio
PXI-6143	ni_pcimio	PCI-6221	ni_pcimio
		PCI-6224	ni_pcimio
		PCI-6225	ni_pcimio
		PCI-6229	ni_pcimio
		PCI-6250	ni_pcimio
		PCI-6251	ni_pcimio
		PCI-6254	ni_pcimio
		PCI-6259	ni_pcimio
		PCI-6280	ni_pcimio
		PCI-6281	ni_pcimio
		PCI-6284	ni_pcimio
		PCI-6289	ni_pcimio
		PCI-6711	ni_pcimio
		PCI-6713	ni_pcimio
		PCI-6731	ni_pcimio
		PCI-6733	ni_pcimio

Table 3.3: Analogy supported National Instruments DAQ cards

Device	**Driver**	**Device**	**Driver**
		PCI-6224	ni_pcimio
		PCI-6225	ni_pcimio
		PCI-6229	ni_pcimio
		PCI-6250	ni_pcimio
		PCI-6251	ni_pcimio
		PCI-6254	ni_pcimio
		PCI-6259	ni_pcimio
		PCI-6280	ni_pcimio
		PCI-6281	ni_pcimio
		PCI-6284	ni_pcimio
		PCI-6289	ni_pcimio
		PCI-6711	ni_pcimio
		PCI-6713	ni_pcimio
		PCI-6731	ni_pcimio
		PCI-6733	ni_pcimio

! → RTXI has no built-in software limitations on the number of DAQ cards but is configured for only one card by default. If you want to use additional cards, you will need to edit the configuration file. Here is the relevant excerpt of `/etc/rtxi.conf`:

```
<OBJECT component="plugin" library="comedi_driver.so"
   id="2">
<PARAM name="0">/dev/comedi0</PARAM>
<PARAM name="Num Devices">1</PARAM>
<OBJECT id="13" name="0" />
</OBJECT>
```

Edit the lines to add another COMEDI device and change the number of devices:

```
<PARAM name="0">/dev/comedi0</PARAM>
<PARAM name="1">/dev/comedi1</PARAM>
<PARAM name="Num Devices">2</PARAM>
```

You will need to exit and restart RTXI for the new configuration to take effect. Settings files that you have already created should still work when you change `rtxi.conf` but you may not have access to both DAQ cards in the System Control Panel, the Oscilloscope, and the Connector. You will have to rebuild those settings files or edit them as above using your choice of text editor.

RTXI automatically detects the manufacturer and board names of available DAQ cards and the number and type of input and output channels. The first DAQ card installed in your system is assigned the Linux device name: `/dev/comedi0`. Additional DAQ cards are assigned device names `/dev/comedi1` and so on. You can check that your DAQ card has been correctly detected and see the corresponding device name by clicking **Help→About COMEDI** from the RTXI menu bar.

To calibrate your DAQ card, use the comedi_calibrate utility as follows for each COMEDI device:

```
$ sudo comedi_calibrate --reset --dump --calibrate --results
   --verbose /dev/comedi0
```

If you are using a National Instruments M-Series card, you will need to use the `comedi_soft_calibrate` utility instead.

3.4 Installation

The following sets of instructions are provided for installation of RTXI on Ubuntu 12.04 and Scientific Linux 6.5. Other Linux distributions are compatible with RTXI and require similar steps for compiling a real-time kernel, installing Xenomai, and installing RTXI and its dependencies. Differences include the commands for compiling a new kernel, filesystem hierarchies, and package names within each distribution's repository. To date, RTXI has been successfully installed on Ubuntu, Scientific Linux, SUSE, and Fedora.

RTXI requires a real-time kernel, which is created by patching Xenomai on to an existing mainline kernel. Xenomai versions are compatible with specific kernel versions. Users can choose any supported combination of Linux and Xenomai, but our installation scripts and media push:

Distribution	**Linux kernel**	**Xenomai**	**Live CD Available?**
Ubuntu 12.04	3.8.13	2.6.3	Yes
Scientific Linux 6.5	3.8.13	2.6.3	No

Table 3.4: Recommended platforms for RTXI installation. The first column gives the distribution version installed via an official Live CD. The second and third respectively specify the vanilla Linux kernel and Xenomai version used to build a real-time kernel.

3.4.1 The Easy Method

The Live USB provides a complete real-time Linux operating system with RTXI without installing anything on your computer. Within the live environment, you can mount your hard drive and technically run experiments, though real-time performance will be slower compared to that of an actually installed system. For actual experiments, we strongly recommend installing RTXI and the entire operating system environment on your hard drive. This can be easily done from within the Live USB.

Note: You do not need a DAQ card installed to test the Live CD or for RTXI to run.

The Live USB is distributed as an ISO image on our website (www.rtxi.org) for both 32 and 64-bit systems. To decrease the size of the Live USB, we removed many common desktop applications, such as LibreOffice, that can easily be reinstalled later. Note that while the image can be burned to a DVD and then run, there will be significant speed limitations.

To install the live image to a USB, follow these steps:

1. **Download the image.** Pick the one that corresponds to your processor's architecture. As a general rule, systems built in the past few years support 64-bit, but if your machine is old, try the 32-bit one.
2. **Download Unetbootin.** (http://unetbootin.sourceforge.net/) Unetbootin is a cross-platform (Windows, Linux, and OS-X) application that can install a bootable .iso image to a USB. When it starts, you will see a window similar to this:
3. **Format a flash drive.** Format your USB drive to FAT32. Note that this process will wipe all the data already on the drive. If you do not know how to do this, Unetbootin should be able to do it for you.
4. **Burn the ISO.** Select the second option to pick the ISO file you want to burn. In the dialog window that opens, select the image you downloaded in step 1. Then, select the USB where you want

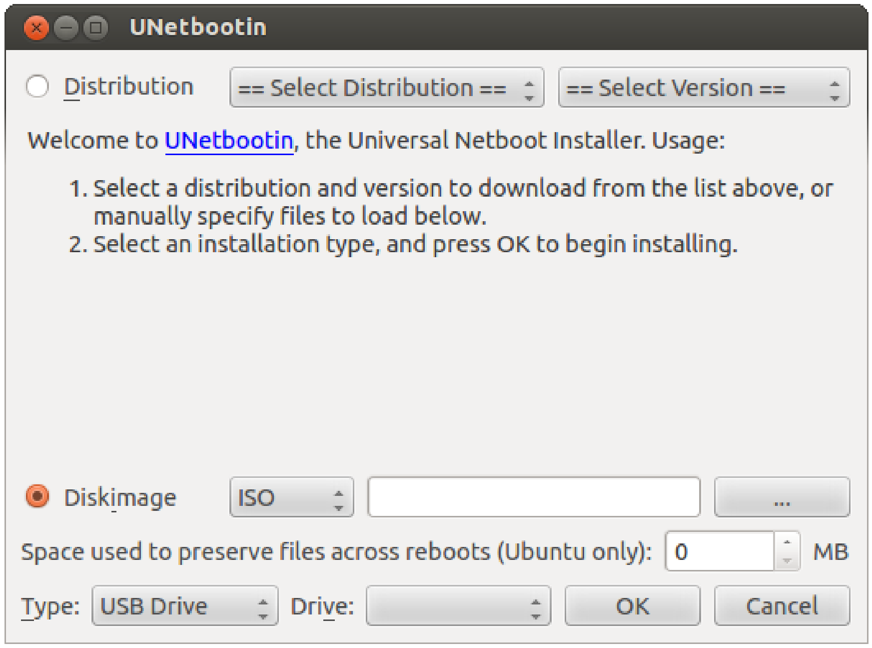

Figure 3.1: Unetbootin is a cross-platform program that burns ISO images to bootable media.

to install RTXI. If you are using Ubuntu, you may want to use the persistence feature available as "Space used to preserve...". This allows you to add/modify files and have the changes saved from one session to another. Without it, all changes to the live environment are lost when the system shuts down.

Note of caution: Be sure you know which drive is which. If you accidentally write to the wrong drive, you will lose any data already stored on it.

Once everything is set, click "OK" to start the installation process.

5. **Boot using the Live USB.** Once the Unetbootin is finished, shut off your system. By default, your computer may not be configured to boot off a USB drive. When you turn on your computer, you will have to press "F2", "F4", or another F key depending on your computer to access your BIOS menu. The following instructions use the BIOS of our system, but your will likely be similar.

   You can navigate though the menu using the arrow keys of your keyboard. Press "Enter" to select a highlighted option and "Esc" to move from a sub-menu to its parent menu. Navigate to "Boot" or some similar window and change the boot order.

   Move to EXIT, save the changes, and exit BIOS.

   Your system will restart and should now boot into your Live USB. In some cases, Ubuntu will correctly load but will drop you to a shell prompt rather than the GUI. This is usually due to video drivers that were not bundled directly into the kernel or were not able to be loaded.

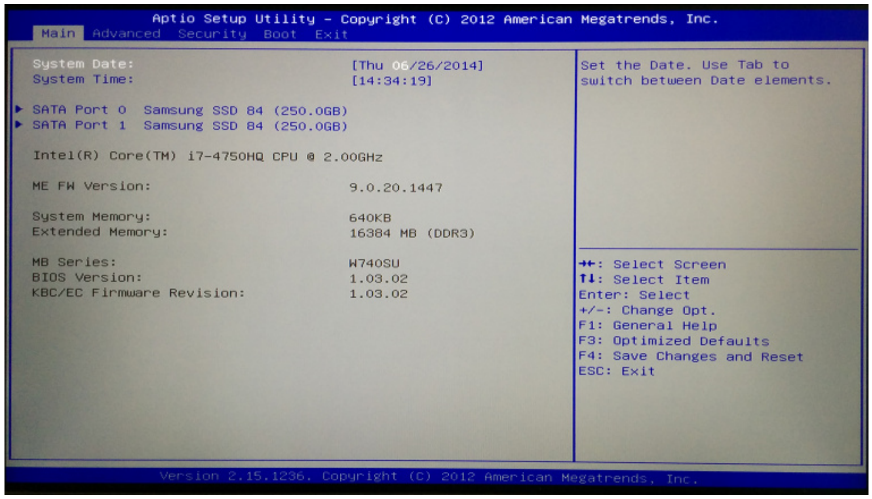

Figure 3.2: The BIOS provides low-level control over system functions, such as the media used to boot an operating system.

If you see a shell prompt, try starting the GUI by typing:

```
# startx
```

To install RTXI from the live environment to your hard drive, follow these steps.

1. **Start the installation program.** To start, double-click the “Install” icon on the desktop.
2. **Set configuration options.** Answer all the setup questions. If you are in the U.S., the default choices will likely suffice.
3. **Choose a partitioning scheme.** The installer provides several options for installing Linux on a hard drive. You can:
   (a) **Erase the hard drive and install the operating system.** To do this simply choose “Erase hard drive and install”. Remember, once the hard drive is erased, all previously stored information will be lost, so make sure to select the correct drive and check that no needed files have been left on it.
   (b) **Install the operating system alongside an existing one(s).** This option is called dual-booting, where two or more operating systems are present. Users are prompted at boot to select which one to use.

   For users who want to dual-boot Ubuntu with another operating system, below is an example of a computer with two hard drives, each of which has a single partition defined.

   SATA hard drives are listed as sdX where X is either a letter or a number. In our case, `sda` refers to the first hard drive connected to the motherboard (in this case the SATA0 slot) and `sdb` is the second hard drive. If you have IDE rather than SATA hard drives, you should see `hda` drive designations.

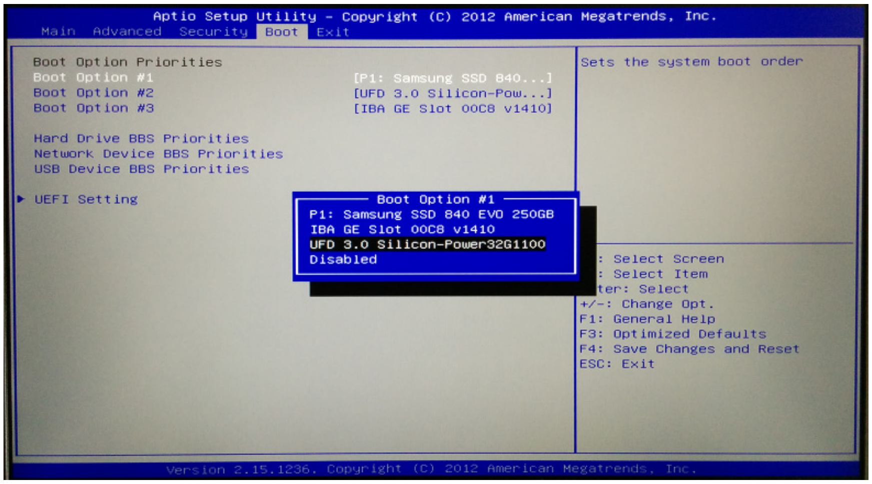

Figure 3.3: The "Boot" tab provides options to change the boot priority of different media.

`sda1` is the first and only partition on the first hard drive. The second hard drive has about 35GB of free space in which to create new partitions for Linux. If you don't have any free space for Linux, you will need to resize your partition. Do this by clicking on it (eg. `/dev/sdb1`) and then on "Edit Partition." You will get a pop-up window with a colored bar representing the partition. Use your mouse to drag the right edge of the colored bar so that the partition is smaller, leaving you with unallocated free space at the end. Hit "OK" and you should see that on `/dev/sdb` you have a `/dev/sdab` partition and more free space.

4. **Set up a SWAP partition.** Linux uses a special partition called SWAP to augment RAM and increase the total amount of virtual memory available to running applications. This allows Linux to move unused files from RAM to SWAP, freeing up fast-operating RAM for new processes, and it avoids problems arising from running out of RAM, which causes the Linux scheduler to random kill processes to free memory. Note that SWAP exists on the hard drive and doesn't not perform nearly as quickly as RAM. Systems that have to access SWAP frequently will have significantly degraded performance.

   For RTXI, SWAP is needed mainly for avoiding process-killing behavior. If you have 3 or more GB of RAM, 1-2GB of SWAP will suffice. If you only have 1-2GB of RAM, you should add 2GB of swap space. You should also consider adding more RAM. Using SWAP will degrade real-time performance. To add SWAP, from within the installer, click on the free space in your partition table and click "New Partition." The default size of a new partition is the rest of the free space on the hard drive so decrease it to the desired size of your swap partition and select the type as "swap."

5. **Set up the Linux partition.** For the actual Linux OS, make another new partition in the remaining free space. This one should be a "Pri-

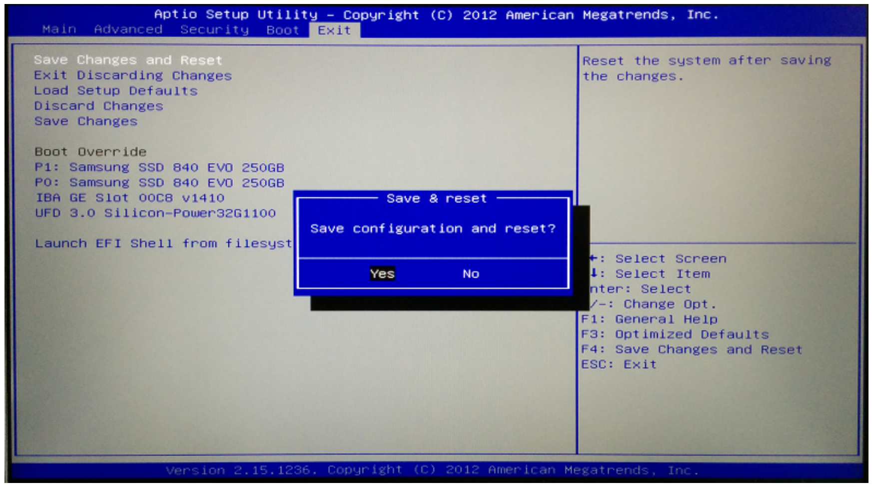

Figure 3.4: To implement changes, save them and then restart the system. At this point, any changes to boot media priority will be applied.

mary" partition and set the mount point to be a single forward-slash "/". Set filesystem format as ext4.

You may also want to leave some extra room for a data partition that is read/write-able in both Windows and Linux. This partition should be formatted as NTFS. Note that FAT32 can only handle file sizes up to 4GB. You can have up to 4 primary partitions. If you need more, you need to create an extended partition under which you can create as many "logical" partitions as you like.

6. **Linux installation.** Your new Linux partition should have a flag indicating that the hard drive space will be formatted. If you are dual-booting, make sure your Windows (NTFS) partition is NOT set to be formatted if you want to keep your data. When you're all done setting up your partitions, click "Forward." You can always restart the hard drive partitioner in Linux to resize your partitions. If you decide you need more room in Linux, you can cancel your changes and repeat the previous two steps, this time shrinking down your existing OS even more.

   When satisfied with your settings, start the installation.

7. **Boot into the GRUB menu.**In addition to the OS, the installer will set up a program called GRUB. When you reboot your computer, GRUB will set up a menu that where you choose which OS to boot. Note that if you ever re-install Windows, GRUB will get overwritten, and the OS-selection menu will no longer appear. Linux will still exist on your hard drive. You will just need to reinstall GRUB using a Live CD.

8. **Start RTXI.** Now, with our Linux OS installed, you can start RTXI. From the terminal (CTRL+ALT+T), enter:

```
$ sudo rtxi
```

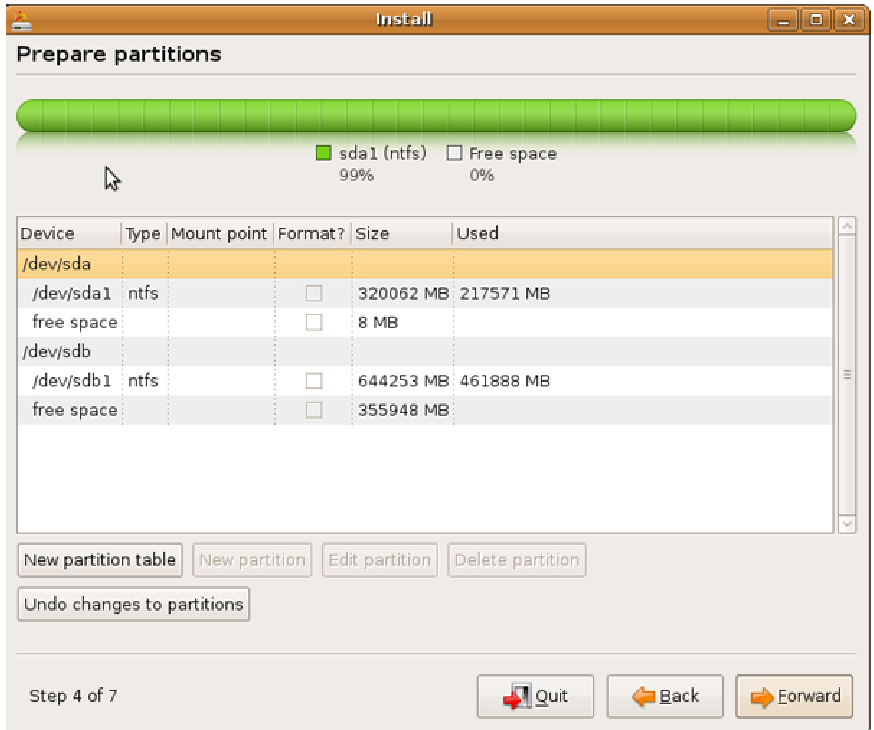

Figure 3.5: The Ubuntu installer allows you to reconfigure your partitions. This configuration shows a single Windows partition (`/dev/sdb1`) in NTFS file format that uses the entire second hard drive (`/dev/sdb`).

You will be prompted for the password you set up when you installed Ubuntu and logged in to your system. Enter this, and RTXI will start.

3.4.2 The Hard Method

The following instructions are provided in the event that you can not use the Live USBs or prefer to customize your system yourself. The scripts we provide are designed for use in Ubuntu 12.04 and Scientific Linux 6.5. The specific configuration is displayed in Table 3.4.

! → All lines beginning with "$" are commands to execute in the terminal. Also, for convenience, use the TAB key to allow the shell to autocomplete directory and filenames for you.

1. textbfDo a clean install. Install a clean version of Ubuntu or Scientific Linux using the official Live CD (links below). Be sure to choose the appropriate 32 or 64-bit version.

   Scientific Linux 6.5 (64-bit):
   `http://ftp1.scientificlinux.org/linux/scientific/livecd/65/x86_64/SL-65-x86_64-2014-02-06-LiveCD.iso`

   Scientific Linux 6.5 (32-bit):
   `http://ftp1.scientificlinux.org/linux/scientific/livecd/65/i386/SL-65-i386-2014-02-06-LiveCD.iso`

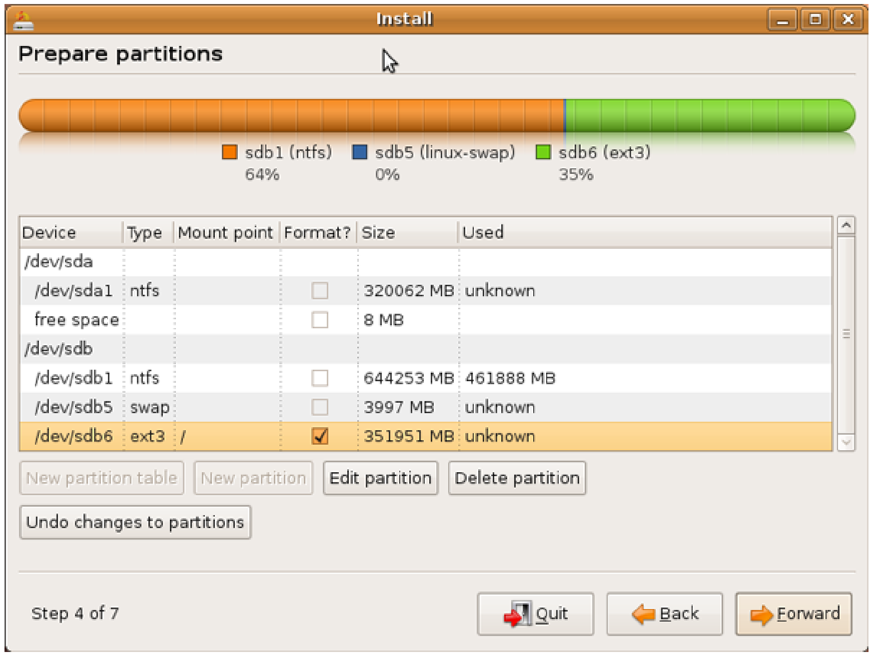

Figure 3.6: This configuration shows a hard drive (`/dev/sdb`) partitioned to dual boot Windows and Linux. The Windows NTFS formatted partition remains and was simply resized. There are additional Linux swap and Linux ext4 formatted partitions as well. The ext4 partition is set to the root `/` mount point and has been selected to be formatted. The NTFS partition is NOT going to be formatted so no data will be lost.

Ubuntu 12.04 (64-bit):
`http://releases.ubuntu.com/precise/ubuntu-12.04.4-desktop-amd64.iso`

Ubuntu 12.04 (32-bit):
`http://releases.ubuntu.com/precise/ubuntu-12.04.4-desktop-i386.iso`

2. **Clone our GitHub repository.** Log into the freshly installed distribution and clone the RTXI repository. If git is not installed on your system by default, run `sudo apt-get install git` in Ubuntu or `sudo yum install git` in Scientific Linux.

   To download our source code and installation scripts from GitHub, where they are stored, run:

   ```
   $ git clone https://github.com/RTXI/rtxi.git
   ```

3.] **Go to the scripts directory.** Change into the new directory and then into the scripts directory. Scripts are files ending with the extension ".sh". Run:

   ```
   $ cd rtxi
   $ cd scripts
   ```

4. **Install dependencies for RTXI.**

   ```
   $ bash ./install_dependencies.sh
   ```

5. **Download, compile, and install the real-time Xenomai-enabled kernel.** This is done via one bash script. Be patient as it may take up to an hour to complete. By default, this script is setup to utilize the maximum number of cpus.

   ```
   $ sudo bash ./install_rt_kernel.sh‘
   ```

   Note: You need to reboot your system into the newly installed real-time kernel once the script has run.

6. **Compile and install RTXI.** The installation process will ask you to select your real-time kernel configuration and the drivers you wish to use. By default, you should select option 1 to install RTXI for Xenomai and Analogy, the respective kernel and drivers installed by the `install_rt_kernel.sh` script.

   ```
   $ bash ./install_rtxi.sh
   ```

7. (OPTIONAL) If you need DYNAMO, you will need to compile the DYNAMO translation utility.

   ```
   $ sudo apt-get install mlton
   $ cd /rtxi/dynamo
   $ mllex dl.lex
   $ mlyacc dl.grm
   $ mlton dynamo.mlb
   $ sudo cp dynamo /usr/bin
   ```

8. **Run RTXI.** You may be asked for your password.

   ```
   $ sudo rtxi
   ```

3.5 RTXI Configuration Options

RTXI can be manually configured with other options. For example, you may want to run RTXI using the RTAI real-time interface rather than Xenomi or in non-real-time mode using the POSIX interface for debugging purposes. You may also direct RTXI to libraries/packages in non-standard locations. The full configuration options are below:

```
Usage:  ./configure [OPTION]...  [VAR=VALUE]...To assign environment variables
   (e.g., CC, CFLAGS...), specify them as VAR=VALUE. See below for descriptions
   of some of the useful variables.  Defaults for the options are specified in
   brackets.

Configuration:

 -h, --help              display this help and exit
 --help=short            display options specific to this package
 --help=recursive        display the short help of all the included packages
 -V, --version           display version information and exit
 -q, --quiet, --silent   do not print `checking...'  messages
 --cache-file=FILE       cache test results in FILE [disabled]
 -C, --config-cache      alias for `--cache-file=config.cache'
 -n, --no-create         do not create output files
 --srcdir=DIR            find the sources in DIR [configure dir or `..']

Installation directories:

 --prefix=PREFIX         install architecture-independent files in PREFIX
                         [/usr/local]
 --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX [PREFIX]

By default, `make install' will install all the files in `/usr/local/bin',
   `/usr/local/lib' etc.  You can specify an installation prefix other than
   `/usr/local' using `--prefix', for instance `--prefix=$HOME'.

For better control, use the options below.

Fine tuning of the installation directories:

 --bindir=DIR           user executables [EPREFIX/bin]
 --sbindir=DIR          system admin executables [EPREFIX/sbin]
 --libexecdir=DIR       program executables [EPREFIX/libexec]
 --sysconfdir=DIR       read-only single-machine data [PREFIX/etc]
 --sharedstatedir=DIR   modifiable architecture-independent data [PREFIX/com]
 --localstatedir=DIR    modifiable single-machine data [PREFIX/var]
 --libdir=DIR           object code libraries [EPREFIX/lib]
 --includedir=DIR       C header files [PREFIX/include]
 --oldincludedir=DIR    C header files for non-gcc [/usr/include]
 --datarootdir=DIR      read-only arch.-independent data root [PREFIX/share]
 --datadir=DIR          read-only architecture-independent data [DATAROOTDIR]
```

```
 --infodir=DIR    info documentation [DATAROOTDIR/info]
 --localedir=DIR  locale-dependent data [DATAROOTDIR/locale]
 --mandir=DIR     man documentation [DATAROOTDIR/man]
 --docdir=DIR     documentation root [DATAROOTDIR/doc/rtxi]
 --htmldir=DIR    html documentation [DOCDIR]
 --dvidir=DIR     dvi documentation [DOCDIR]
 --pdfdir=DIR     pdf documentation [DOCDIR]
 --psdir=DIR      ps documentation [DOCDIR]

Program names:

 --program-prefix=PREFIX           prepend PREFIX to installed program names
 --program-suffix=SUFFIX           append SUFFIX to installed program names
 --program-transform-name=PROGRAM  run sed PROGRAM on installed program names

X features:

 --x-includes=DIR   X include files are in DIR
 --x-libraries=DIR  X library files are in DIR

System types:

 --build=BUILD  configure for building on BUILD [guessed]
 --host=HOST    cross-compile to build programs to run on HOST [BUILD]

Optional Features:

 --disable-option-checking      ignore unrecognized --enable/--with options
 --disable-FEATURE              do not include FEATURE (same as --enable-FEATURE=no)
 --enable-FEATURE[=ARG]         include FEATURE [ARG=yes]
 --enable-shared[=PKGS]         build shared libraries [default=yes]
 --enable-static[=PKGS]         build static libraries [default=yes]
 --enable-fast-install[=PKGS]   optimize for fast installation [default=yes]
 --disable-dependency-tracking  speeds up one-time build
 --enable-dependency-tracking   do not reject slow dependency extractors
 --disable-libtool-lock         avoid locking (might break parallel builds)
 --enable-rtai                  build the Xenomai interface
 --enable-posix                 build the POSIX non-RT interface
 --enable-debug                 turn on debugging
 --enable-comedi                build the comedi driver
 --enable-ni                    build the ni driver

Optional Packages:

 --with-PACKAGE[=ARG]            use PACKAGE [ARG=yes]
 --without-PACKAGE               do not use PACKAGE (same as --with-PACKAGE=no)
 --with-cppunit-prefix=PFX       Prefix where CppUnit is installed (optional)
 --with-cppunit-exec-prefix=PFX  Exec prefix where CppUnit is installed (optional)
 --with-pic                      try to use only PIC/non-PIC objects [default=use both]
 --with-gnu-ld                   assume the C compiler uses GNU ld [default=no]
 --with-x                        use the X Window System
```

```
  --with-Qt-dir=DIR          DIR is equal to $QTDIR if you have followed the
                             installation instructions of Trolltech.  Header files
                             are in DIR/include, binary utilities are in DIR/bin.
                             The library is in DIR/lib, unless --with-Qt-lib-dir
                             is also set.
  --with-Qt-include-dir=DIR  Qt header files are in DIR
  --with-Qt-bin-dir=DIR      Qt utilities such as moc and uic are in DIR
  --with-Qt-lib-dir=DIR      The Qt library is in DIR
  --with-Qt-lib=LIB          Use -lLIB to link with the Qt library
  --with-rtai-config=FILE    location of the rtai-config program

Some influential environment variables:
  CC          C compiler command
  CFLAGS      C compiler flags
  LDFLAGS     linker flags, e.g.  -L<lib dir> if you have libraries in a nonstandard
              directory <lib dir>
  LIBS        libraries to pass to the linker, e.g.  -l<library>
  CPPFLAGS    C/C++/Objective C preprocessor flags, e.g.  -I<include dir> if you have
              headers in a nonstandard directory <include dir>
  CPP         C preprocessor
  CXX         C++ compiler command
  CXXFLAGS    C++ compiler flags
  CXXCPP      C++ preprocessor
  XMKMF       Path to xmkmf, Makefile generator for X Window System

Use these variables to override the choices made by ‘configure’ or to help it
   to find libraries and programs with nonstandard names/locations.
```

3.6 Installing User Modules

RTXI comes with a limited set of core system modules and sample user modules. All modules are compiled as Linux shared object libraries that are linked into the core system. This allows RTXI to have minimal overhead and user modules are loaded only as needed. This architecture also allows multiple instantiations of user modules so that elements such as filters and event detectors can be reused on a variety of signals.

User modules are available on the RTXI website (http://www.rtxi.org/modules) as zipped files or tarballs and also on our GitHub repository (https://github.com/RTXI). Each module consists of a single directory, typically containing a single class header file (*.h), class implementation file (*.cpp), and a Makefile that informs the GCC compiler. For modules that implement more complex protocols, we also have a directory called `rtxi_includes` that contains additional needed sources and headers. For simplicity, we recommend that user modules be stored together in a single directory (such as `$HOME/modules`). To extract a module compressed as a tarball, in this case the plugin template, directory and compile the module:

```
$ tar xvf plugin_template.tar.gz
$ cd plugin-template
```

```
$ sudo make install
```

If you have Git installed, you can also download and install modules by running:

```
$ git clone https://github.com/rtxi/plugin-template.git
$ cd plugin-template
$ sudo make install
```

This process will create an RTXI object library (*.so extension), which will then be copied to `/usr/local/lib/rtxi`, where RTXI will initially look for them. User modules must be recompiled if any changes are made to their sources after installation. Note that when reinstalling modules, the corresponding *.so files are overwritten, so always make sure that different modules have unique names. Users should carefully name their custom modules since the compiled binaries are automatically overwritten. Instructions for writing custom user modules are given in Chapter 4.

3.7 Acquiring Data (Model Cell Tutorial)

This section presents a tutorial similar to those described by Molecular Devices for their suite of electrophysiology products. This exercise uses a CLAMP-1U model cell with an Axon™ Axoclamp™ 2B amplifier operating in bridge mode. Connect the HS-2A-x0.1LU headstage to the ME1 PROBE connector and the HS-2A-x1LU head stage to the ME2 probe connector on the back panel of the amplifier.

Connect equipment to DAQ card

Make the following connections between the amplifier and the DAQ card.

1. Axoclamp 10Vm Output → DAQ Analog Input 0
2. Axoclamp Im Output → DAQ Analog Input 1
3. DAQ Analog Output 0 → Axoclamp EXT. ME1 COMMAND

Start RTXI

If you have not already calibrated your DAQ card, use `comedi_calibrate` or `comedi_soft_calibrate` (if you have an NI M-series card):

```
$ sudo comedi_calibrate --reset --dump --calibrate --results
   --verbose /dev/comedi0
```

If you installed RTXI using the Live CD, you may start RTXI from the Applications menu. However, it is a good idea to start RTXI from the terminal since some modules output error messages or warnings to the terminal:

```
$ rtxi
```

Configure DAQ Channels

While RTXI automatically detects the available channels on your DAQ card, they need to be configured inside RTXI. From the **System** menu on the RTXI menu bar, choose the **Control Panel**. The default device should be your DAQ card listed as `/dev/comedi0`. The analog channels on most multifunction DAQ cards have a range of -10 V to +10 V based on a ground reference and this is the default in RTXI. You should check the specifications for your DAQ card and choose the corresponding settings in RTXI. For Analog Input 0, the amplifier specifies that the signal has a gain of 10 applied (10Vm). Invert this and enter a scale of 0.1 V/V for this channel. Click the **"Active"** toggle button and click the **"Apply"** button. You will not acquire any actual data on a channel until it has been set to "Active."

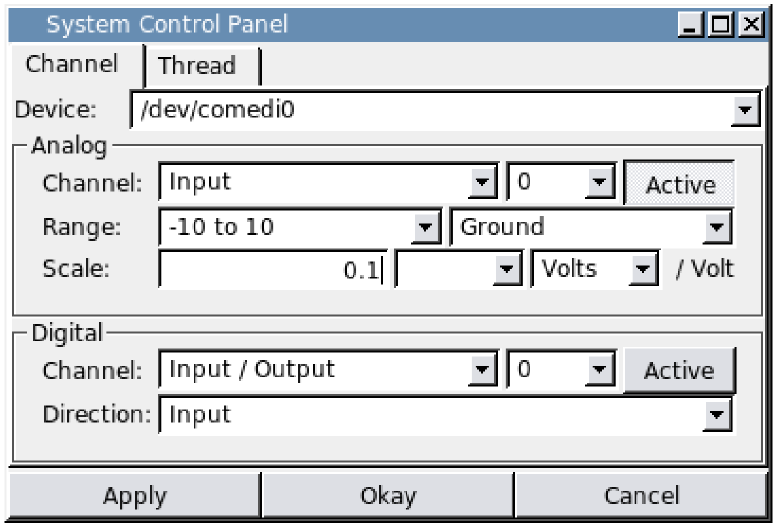

Figure 3.7: Configuring Axoclamp 10Vm output on Analog Input 0

For the membrane current assigned to Analog Input 1, the Axoclamp amplifier specifies that there is a gain of 10 ÷ H mV/nA. Since the headstage gain on ME1 is H=0.1, the conversion is 100 mV/nA or 0.1 V/nA. Invert this to get a scale of 10 nA/V. You may set the "Scale" dropdown box to either units of volts or amperes but this does not affect the computation. **!** → Note that you must compute the total gain applied to a channel by any combination of hardware and software along the path of the signal. For example, if you are using the Axon^{TM} Multiclamp^{TM} Microelectrode Amplifier by Molecular Devices, you should take into account Multiclamp Commander software gain.

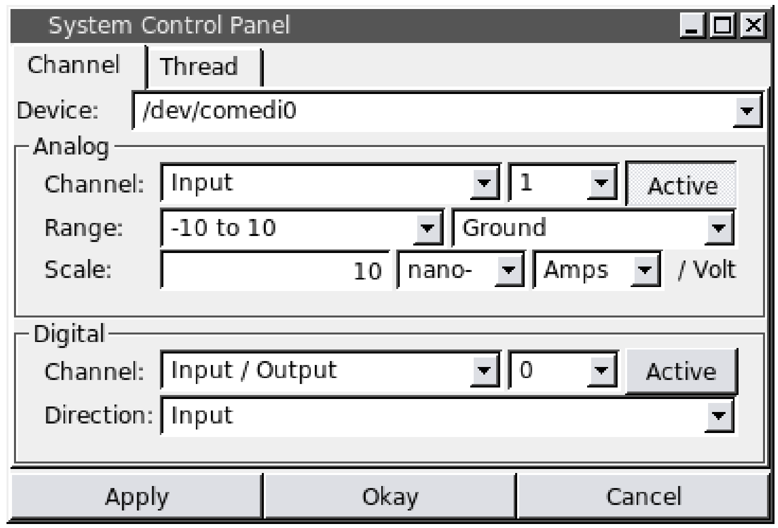

Figure 3.8: Configuring Axoclamp Im output on Analog Input 1

For the EXT. ME1 COMMAND assigned to Analog Output 0, the Axoclamp specified a gain of 10 x H nA/V, which comes to 1 nA/V. Invert this to get a scale of 1 gigaV/A.

! → The Thread Tab of the System Control Panel allows you to set the real-time period or sampling rate of the system. The default sampling rate is 1 kHz, which is sufficient for this exercise.

Configure stimulus module

The Istep module generates current step stimuli (square wave pulses). Install the Istep module according to the directions in Chapter 3.6 and load it by selecting **Modules→Load Modules** from the RTXI menu bar. Set the amplitude of the current pulse to 5 nA and set the width of the pulse to 40 ms using the Period and Duty Cycle options. The number of pulses is set using the Cycles option.

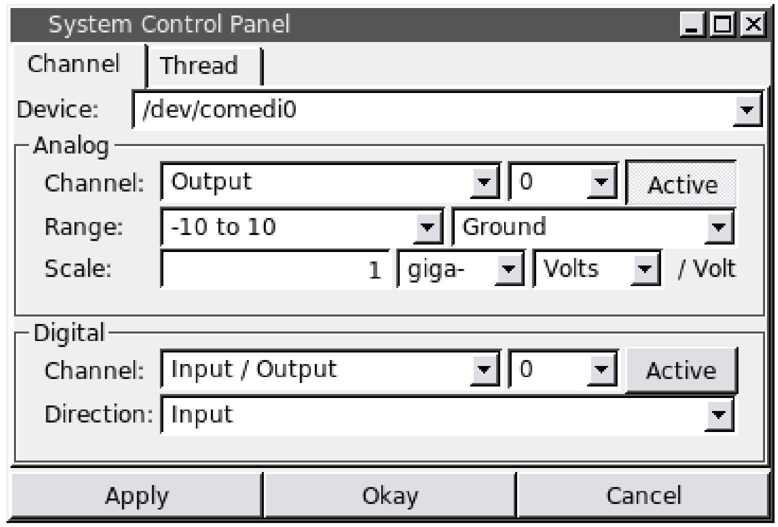

Figure 3.9: Configuring EXT. ME1 COMMAND on Analog Output 0

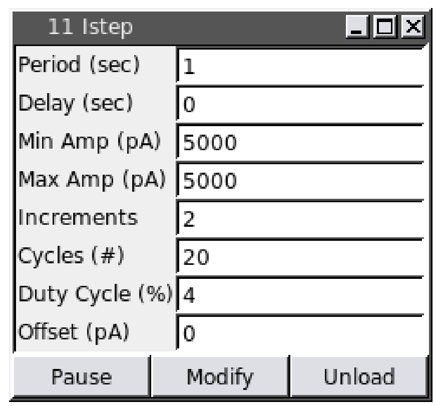

Figure 3.10: The Istep module delivers current step pulses.

Configure Oscilloscope

→ Chapter 2.2.2 Oscilloscope

! →

Start the Oscilloscope by selecting **System→Oscilloscope** from the RTXI menu bar. Right click anywhere in the Oscilloscope window to bring up the context menu and select the **Properties** menu. The **Channel Tab** is used to select signals to plot and set an appropriate scale and line style. The architecture of RTXI is based on modular components that have input and output signals. The DAQ card is abstracted as a DAQ device block such that a signal acquired on an input channel of the DAQ *card* becomes an output signal of the DAQ *device* within RTXI. To plot the voltage acquired on Analog Input 0, use the dropdown box to select **"Output"**. The right-most dropdown box will automatically be populated with the analog input channels of the DAQ card. Click the **"Active"** toggle button to plot the signal.

When a signal is plotted, the Oscilloscope will generate a legend in the lower part of the window. In the **Display Properties** section, choose a scale and if needed, an offset. You may also choose a linestyle in the **Line**

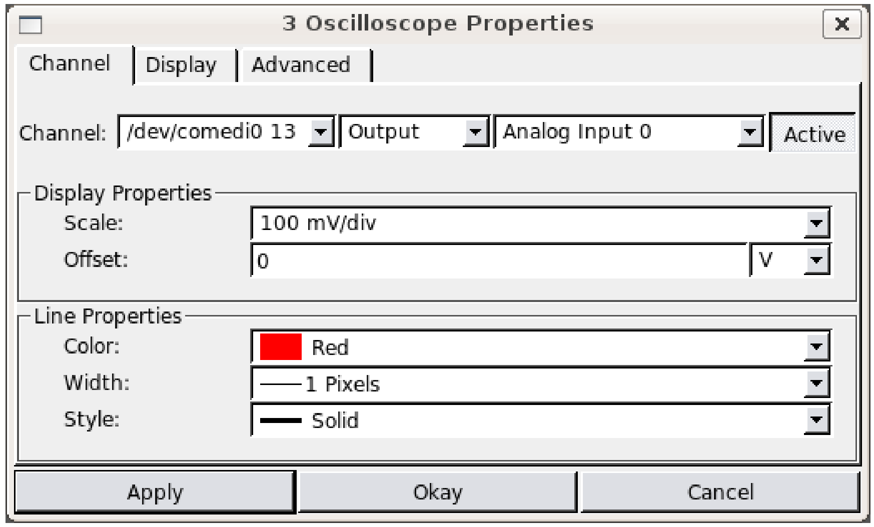

Figure 3.11: The Oscilloscope Channel Tab allows you to select signals to plot and choose different scales and line styles.

Properties section. You must click the **Apply** button for these changes to take effect. Each signal may have a different scale and different line style. Use the System Control Panel to plot Analog Input 0 and Analog Input 1 to monitor the voltage and the current applied to the model cell. You should also plot the "Iout" output signal of the Istep module.

Connect Signals Within RTXI

To generate signals from the Istep module, untoggle the **"Pause"** button. RTXI will begin executing the real-time code specified in this module. You should see pulses in the Istep signal in the Oscilloscope window. You can use the textboxes in the GUI to change the parameter values. The text will turn red but there will be no change in the module's output signal until you click the **"Modify"** button. Clicking this button initiates an event that will update the parameter in real-time and you should see the corresponding change immediately in the Oscilloscope. At this point, you should not see any pulses in Analog Input 1 from your DAQ card, which is the current actually delivered by the amplifier. To apply this stimulus to the model cell, you need to make a connection between the Istep module and the DAQ card. From the RTXI menu bar, select **System→Connector**.

The Connector module populates the **"Output Block"** with the available signals in your workspace and the **"Input Block"** with the available slots, or destinations. Select the Istep module in the "Output Block" and the "Iout" signal. Choose your DAQ card (`/dev/comedi0`) in the "Input Block" and the Analog Output 0 channel. To make the connection, click the central "==>" toggle button. Now, the Oscilloscope should show matching data for both Istep: Iout and Analog Input 1.

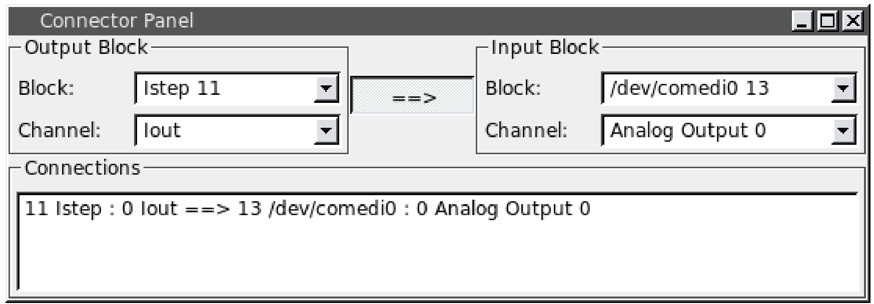

Figure 3.12: The Connector allows you to create connections between user modules or between modules and the DAQ card.

Balance the Bridge in the Bath

Switch the CLAMP-1U model cell to the BATH position. Use the amplifier INPUT OFFSET knob to zero out the voltage based on the readings in the Oscilloscope. The Oscilloscope shows the actual sample values (with the channel gain applied) that will later be saved using the Data Recorder module. If your amplifier or other control software indicates a nonzero voltage when RTXI reports a zero voltage, calibrating your DAQ card may eliminate this offset. Unpause the Istep module to begin delivering current pulses to the model cell. Adjust the BRIDGE knob until the voltage deflection is eliminated and then adjust the CAPACITANCE NEUTRALIZATION knob until the residual transients are minimized. Now switch the model cell to the CELL position. If you have correctly tuned these settings, you should see a response to each current pulse as in Figure ??.

→ Chapter 3.3 COMEDI calibration

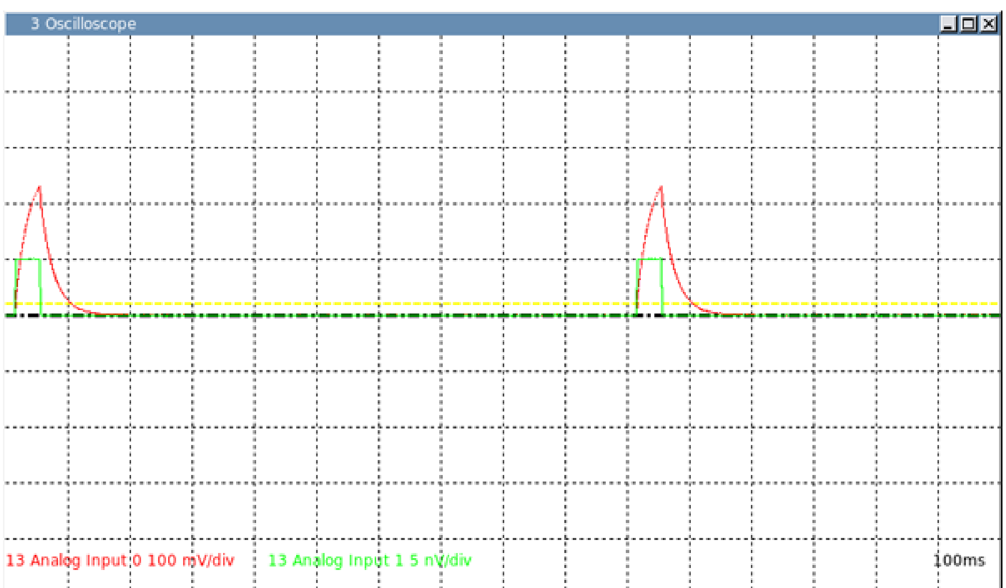

Figure 3.13: Model cell response to current injection pulses with correctly balanced bridge and capacitance neutralization

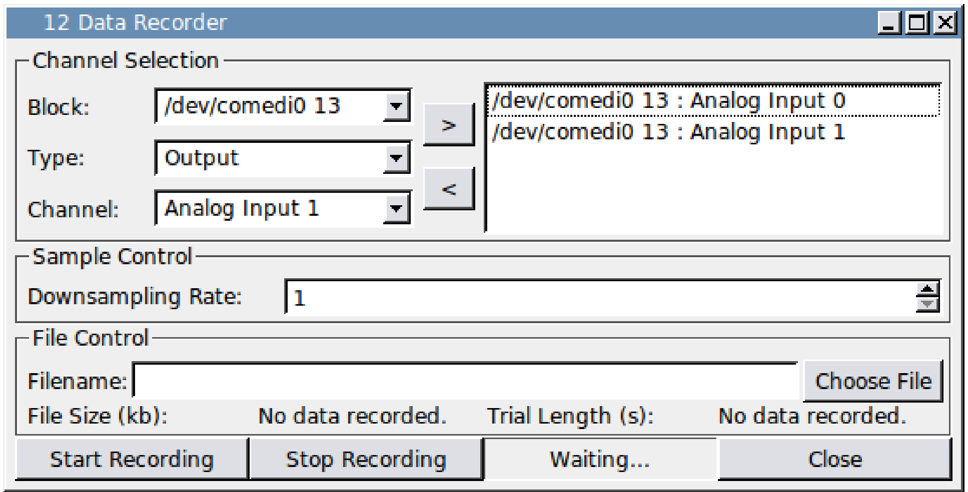

Figure 3.14: The Data Recorder saves any synchronous signal in your workspace to an HDF5 file.

Saving Data

To record data, select **System→Data Recorder** from the RTXI menu bar. The **"Block"** menu is a list of your DAQ card(s) and any loaded user modules. Selecting a block device then populates the **"Type"** and **"Channel"** menus. Select the Analog Input 0 channel from your DAQ device and click the ">" button. To remove a channel from the list, highlight it in the listbox and click the "<" button. Before you can start recording, you must select a file by clicking the **"Choose File"** button. Click **"Start Recording"** to begin recording and **"Stop Recording"** to stop recording.

Saving Your Workspace

At this point, you have configured several channels on the DAQ card and the Oscilloscope, set custom parameters for a user module, and connected the module to the DAQ card to generate an external signal. RTXI allows you to save all these settings to a file by selecting **File→Save Workspace** from the RTXI menu bar. To reload the file and reconstruct your entire working environment, select **File→Load Workspace**.

3.8 HDF5 Data Files

The HDF5 file format is a portable and extensible binary data format designed for complex data. It features support for an unlimited variety of datatypes, and has flexible and efficient data retrieval and storage methods. HDF5 features a hierarchical structure that allows you to access chunks of data without loading the entire file into memory. An HDF5 file produced by RTXI's Data Recorder is organized as shown in Figure **??**.

At the topmost level, an RTXI HDF5 file is divided into separate *Trial* groups, each of which contains the system settings and module parameter values that existed at the time that data was recorded. The Data Recorder

! → only saves parameters values for modules from which it is recording a signal. A new *Trial* is created whenever the Data Recorder is used to start recording. For example, if you stop and start recording multiple times in a single session, RTXI automatically increments the trial number each time. If you choose to save data to a file that already exists, RTXI will prompt you with a choice to overwrite the file or append new data to the file. Appending data to a file also creates a new *Trial*. Thus, it is possible to have trials within the same file that contain different parameter settings and even data downsampling rates.

Parameter values from user modules are saved in the *Parameters* group within each *Trial*. The name of each parameter includes the module instance ID number within RTXI, the name of the module, and the name of the parameter itself. If the value of the parameter changes during recording, all the values are saved with a corresponding index value that is the timestamp in nanoseconds from the start of the recording. This feature is only available

→ Chapter 4.1 `DefaultGUIModel`

! → for user modules that are based on the `DefaultGUIModel` class. Note that certain naming conventions for parameters also apply in order for them to be captured to HDF5.

Real-time signals in RTXI are streamed to the *Synchronous Data* structure within each *Trial*. This group contains separate fields with the name of each synchronous channel and a single dataset that contains all the synchronous data. The order of the channel names corresponds with the columns in the dataset. In Figure **??**, "/Synchronous Data/Channel 1 Name" refers to the data stored in column 0.

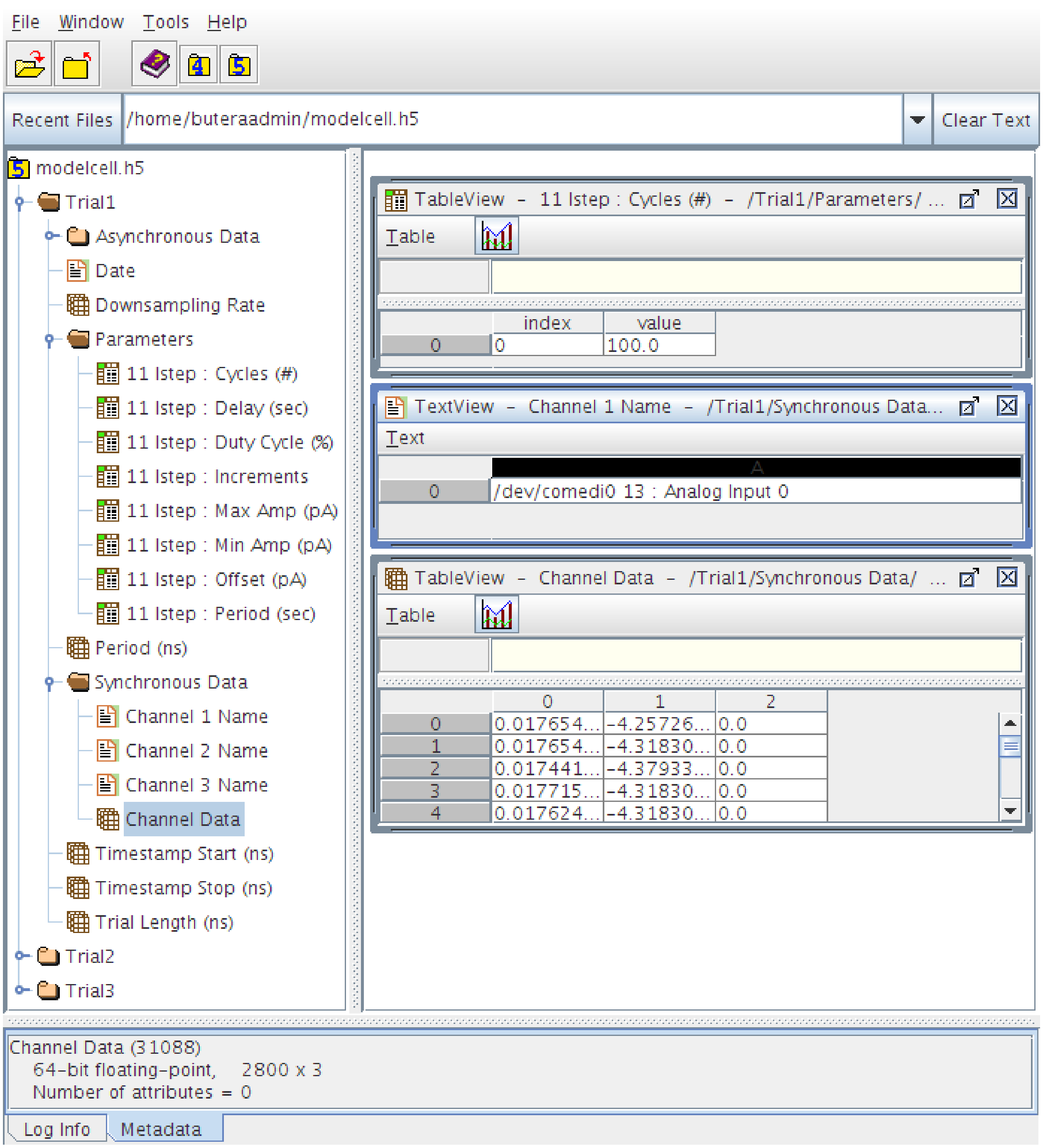

Figure 3.15: RTXI uses a hierarchical HDF5 data structure organized into *Trials*. Each *Trial* contains the system settings and parameter values for that trial. This screenshot is taken using HDFView, a free software for browsing HDF5 files.

MATLAB & RTXI HDF5

There are various software available for working with HDF5 files. To simply browse the file structure, you can use the free HDFView application. HDFView provides some limited editing capabilities. For trials where only a single channel is saved, you can also preview a plot of the data. To extract the data for analysis and for complete editing capabilities, APIs are available for MATLAB, GNU Octave, Igor Pro, Mathematica, Python, Scilab, and other software. For real-time streaming of multiple signals, an HDF5 data type is used in RTXI that does not map efficiently onto MATLAB native data types. While MATLAB can read this data using its low-level functions, this process can be very slow. To load RTXI HDF5 files quickly into MATLAB, you will first need to run a small utility function on your HDF5 file to convert the *Synchronous Data* dataset to a different data type. This function is compiled when RTXI is compiled and is located in `/rtxi/hdf`. To convert your HDF5 file:

```
$ rtxi_hdf_matlablize YOUR_FILE.h5
```

To make this utility accessible from any directory on your system, make a symbolic link in `/usr/bin` to the location of this function in your RTXI source directory. If you installed RTXI from the Live CD, the source directory is `/home/rtxi`:

```
$ sudo ln -s RTXI_SRC_DIR/hdf/rtxi_hdf_matlabize
      /usr/bin/rtxi_hdf_matlabize
```

RTXI also includes a simple MATLAB GUI for quickly viewing the data within a single trial. The MATLAB code is available in `/rtxi/hdf/RTXIh5`. A sample m-file is provided with examples of how to extract data to the MATLAB workspace, how to use the GUI browser, and how to add new datasets to your file. It is also possible to embed binary formats, such as images, within a trial.

The GUI browser allows you to view the parameters, channels, and plots of the data within a single trial with the `rtxibrowse()` function. This generates a MATLAB figure window with the filename and trial number in the menu bar. To browse trials within the same HDF5 file, use the buttons in the lower left corner. The left panel lists the initial values for all the module parameters. If a parameter value has changed during the recording, this is denoted with an asterisk. The new values and their timestamps can be viewed by using the `getTrial()` function, which returns a MATLAB structure containing all the information within a trial. The GUI plots two channels from the same trial. Use the middle upper and and lower panels to select the data that is plotted in the right upper and lower panels. Double-clicking on a channel name in the middle panel will create a new figure window with that data plotted. This allows you to continue browsing through other trials in the main GUI window while keeping this additional plot available.

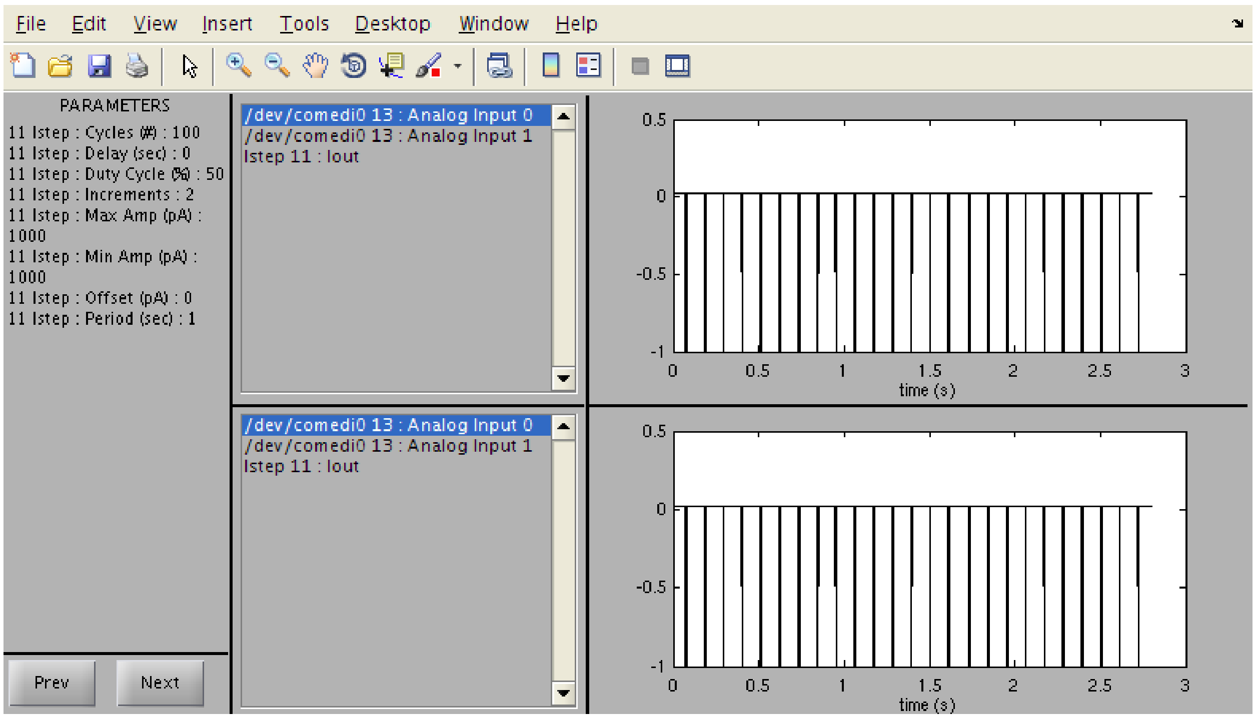

Figure 3.16: The MATLAB GUI browser allows you to view the parameters, channels, and plots of the data within a single trial.

4 Writing Custom User Modules

4.1 Using the `DefaultGUIModel` Class

User modules are implemented within RTXI as custom C++ classes. The recommended way to create a module is to abstract from our provided base class named `DefaultGUIModel`. `DefaultGUIModel` constructs a simple graphical user interface (GUI) that allows users to interact with parameters and activate real-time code. Modules abstracted from it also inherit its methods for hard real-time execution and event handling, generating and accept signals, and capturing metadata automatically by the Data Recorder in HDF5 format.

The following sections describe the `Neuron` module, a Hodgkin-Huxley model neuron class abstracted from DefaultGUIModel that generates a membrane voltage signal and accepts an optional external current input. The GUI consists of a column of textboxes and associated labels that display the module's parameters and internal state variables. Parameters are user-editable variables displayed in black, and internal state variables are intermediate computed values that cannot be edited manually by the user. States are shown in gray. Also, at the top left corner of the window is a unique instance ID that is given to each instantiated user module. This ID is important when connecting input and output from one module to another.

4.1.1 Creating your own module class

The quickest way to create a new user module is to duplicate an existing module directory and rename the files and the class. This involves renaming the class header (*.h) file, the class implementation (*.cpp) file, editing the Makefile and editing any instances of the old class name within each of these files. The latter include the class name, scope names, the constructor, and the deconstructor. A template user module is available online at `https://github.com/RTXI/plugin-template.git`.

Alternatively, you can browse through our module repository on GitHub (`https://github.com/RTXI/`) to find modules that perform functions similar to those desired. All our code is open-source, so you are free to fork our existing code and reconfigure it to meet your needs.

4.1.2 Edit the Makefile

The Makefile instructs the compiler how to build your module and link it to RTXI. In RTXI v1.2 and later, the Makefile allows modules to be compiled outside the core RTXI source tree. The following sample Makefile installs a plugin called `my_plugin` with dependencies `included_class.h` and its source `included_class.cpp`.

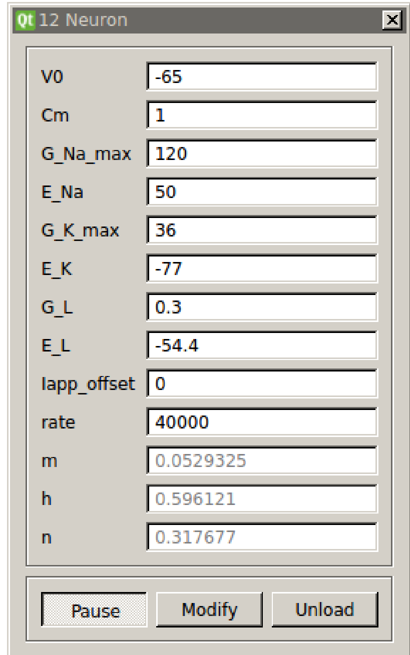

Figure 4.1: The `Neuron` module is a Hodgkin-Huxley model neuron described by conductance-based differential equations. This GUI provides an interface by which a user can modify parameters, such as the conductance of the ion channels, on-the-fly and start and stop real-time execution of the module

```
PLUGIN_NAME = my_plugin

HEADERS = my_plugin.h

SOURCES = my_plugin.cpp \
          included_class.h \
          included_class.cpp

LIBS = -lgsl ### Do not edit below this line ###

include $(shell rtxi_plugin_config --pkgdata-dir)/Makefile.plugin_compile
```

The `PLUGIN_NAME` is the name of the shared object library (*.so) file when it is compiled. All modules should be given unique names because the compilation process will automatically overwrite identically-named modules. The `HEADERS` and `SOURCES` should also be edited to reflect the new source file names. For simple modules based on a single class, a single header and source file is all that is needed. You may base your module on additional custom classes whose sources must then be included here as well. The `LIBS` flag is used for any additional library flags. Here, `-lgsl` links this module against the GNU Scientific Library.

4.1.3 Define model parameters, inputs, and outputs

`DefaultGUIModel` uses a special workspace variable `vars[]` to define quantities in the module. The declaration of these types follows a simple syntax. ! → Every `DefaultGUIModel` module must have a workspace variable of type `variable_t` as shown in the `vars[]` of the Neuron module.

```
static DefaultGUIModel::variable_t vars[] =
{
   {"Iapp", "A", DefaultGUIModel::INPUT,},
   {"Vm", "V", DefaultGUIModel::OUTPUT,},
   {"V0", "mV", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"Cm", "uF/cm^2", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"G_Na_max", "mS/cm^2", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"E_Na", "mV", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"G_K_max", "mS/cm^2", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"E_K", "mV", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"G_L", "mS/cm^2", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"E_L", "mV", DefaultGUIModel::PARAMETER|DefaultGUIModel::DOUBLE,},
   {"rate", "Hz", DefaultGUIModel::PARAMETER|DefaultGUIModel::UINTEGER,},
   {"m" "Sodium Activation", DefaultGUIModel::STATE,},
   {"h", "Sodium Inactivation", DefaultGUIModel::STATE,},
   {"n", "Potassium Activation", DefaultGUIModel::STATE,},
};
```

Each element in `vars[]` defines an `INPUT`, `OUTPUT`, `PARAMETER`, `STATE` variable, or `COMMENT` for the module. The first argument for each element is the label for the textbox in the GUI. This does not have to be the same as the variable name you use in the code to actually store the parameter value. The second argument is displayed as a Tooltip when you use your mouse to hover the cursor over that entry in the GUI. Enter any descriptive information here about the variable, such as an expanded form of your text label or the correct units of measurement. The third argument defines the variable as an input, output, etc. Notice that for parameters, you can also specify whether it is a double or integer numeric type.

Declaring an `INPUT` creates a slot for your module to acquire data from the DAQ card or from another module. An `OUTPUT` creates a signal that is emitted from your module that can be sent to your DAQ card or any other module. These inputs and outputs can be directed from the Connector or the Data Recorder modules. In the Neuron module below, there is only one input, Iapp, and its value is accessed as the variable `input(0)`. If additional inputs had been declared, they would be accessed as `input(1)`, `input(2)`, and so on. The same rules apply for outputs.

`STATE` variables and `PARAMETER`s are numeric datatypes. State variables are internal model variables that cannot be modified by the user through the GUI. Their values may be constant or they may change over time. Use a `STATE` to track the values of intermediate or computed quantities that can then be saved via the Data Recorder. A `PARAMETER` will accept user input through the GUI and can be modified on-the-fly during real-time execution. State variables and parameters appear in the GUI in the order that they are declared. In the example code, this mechanism is used to monitor the ion channel's activation variables, which are dependent on membrane voltage and integrated in real-time. A `COMMENT` is similar to a `PARAMETER`, but is used to store text strings such as information about the experiment that you would like to log. These are saved to the Data Recorder just like parameters, but should not be modified in real-time during model execution.

! → If your parameter name contains a forward slash "/", its values will not be automatically saved by the Data Recorder. This is a limitation of the HDF5 file format, which uses a directory-like syntax for specifying the data structure.

4.1.4 Initialize the model

The next section is the model constructor. If you changed the class name, this would read "`YOURMODEL::YOURMODEL`". You can set the text that will appear in the title bar of your module window using the first argument of the constructor method. ! → The next required line is a call to the `createGUI()` function which generates the GUI shown in Figure 4.1. In this section, you should initialize all the variables and parameter values and make sure that the GUI reflects the actual values that are being used. In this example, much of this code is performed by the `update()` function under the `INIT` flag. In other modules downloadable from our website, you will find a separate `initParameters()` function that handles all variable initializations.

→ Chapter 4.1.6
`update()`

It is convenient to perform unit conversions when calling these functions so that the GUI accepts input in more user-friendly units. Finally, you should call `refresh()` to update the GUI to reflect your changes. The GUI textboxes will be initialized to the current values of the variables and `STATE` variables will be updated periodically during model execution.

```
Neuron::Neuron(void) :  DefaultGUIModel("Neuron", ::vars, ::num_vars) {
   createGUI(vars, num_vars); // creates the GUI
   V = V0;
   m = m_inf(V0);
   h = h_inf(V0);
   n = n_inf(V0);
   period = RT::System::getInstance()->getPeriod() * 1e-6; // convert ns to ms
   update(INIT); // calls the update() function with the INIT flag
   refresh(); // refreshes the GUI to reflect parameter values stored in
   variables
}
```

Notice the method for retrieving the real-time period (sampling rate) of the system:

```
RT::System::getInstance()->getPeriod();
```

This returns the period in nanoseconds.

4.1.5 The execute() loop

The `execute()` function will run to completion on every time step. The computations performed here must complete within the real-time period that you have set in the System Control panel to maintain system stability. The efficiency of your code here will affect the performance of your system. You should use private variables defined in the class header rather than creating variables inside the function on every time step. If you absolutely must create a variable inside `execute()`, use a static call so that the same memory block is used each time. You should be wary of using `do-while` and `for` structures if you are uncertain how long these loops will take to complete. Within the execute function, you must also be careful to bound the output signal and perform your own error checking to maintain the stability of the closed-loop. Notice that at the end, we have set `output(0)` to update the membrane voltage signal emitted by this module. RTXI's signals-and-slots architecture allows you to connect any signal to any slot. There is no error checking that the connection is valid, eg. that quantities with matching units of measurement are connected.

→ Chapter 2.2.1
System Control Panel

! →

! →

```
void Neuron::execute(void) {
  for (int i = 0; i < steps; ++i)
    solve(period / steps, y); // integrate equations
  output(0) = V * 1e-3; // convert to mV
}
```

4.1.6 The update() function

The `update` function implemented in `DefaultGUIModel` that is designed to handle function calls depending on the state of the GUI. It provides several flags to help organize code and handle events in modules.

- `INIT`: non-event related but useful for placing code to initialize the model
- `MODIFY`: called when the "Modify" button is pressed in the GUI
- `PAUSE`: called when the model is paused
- `UNPAUSE`: called when the model is unpaused
- `PERIOD`: called when the real-time period of the system is changed

Under the `INIT` flag, you should initialize any variables or GUI settings that were not already addressed in the constructor. To assign a variable as a `STATE` variable in the GUI, use:

```
setState("YOUR_GUI_LABEL", YOUR_VARIABLE);
```

! → `YOUR_GUI_LABEL` must exactly match the label that you set in `variable_t vars`[] above.

Similarly, you initialize the GUI for a `PARAMETER` with:

```
setParameter("YOUR_GUI_LABEL", YOUR_VARIABLE);
```

It is often the case that you may want to display units in the GUI with more convenient physiological units of measurement, eg. mV instead of V. In that case, you can call the function as follows:

```
setParameter("E_Na", E_Na*1000); // convert to mV
```

! → Always comment your units. Otherwise, your code will not be readily readable by other or even yourself later on,

Under the `MODIFY` flag, you should grab all the values in the GUI textboxes and update the values of the parameters as follows:

```
YOUR_VARIABLE = getParameter("YOUR_GUI_LABEL").toDouble();
```

! → If you do any unit conversions with `setParameter()`, make sure you do the inverse with `getParameter()`. You may also want to add code to the `PAUSE` flag to set the output of your module to zero, e.g. the amplitude of an injected current. In some cases, you will want to reset certain internal variables when you stop or start the model eg. a counter that keeps track of your model execution time. Under the `PERIOD` flag, you will always want to update your model with the new real-time period.

```
void Neuron::update(DefaultGUIModel::update_flags_t flag) {
  switch (flag) {
    case INIT:
      setState("m", m);
      setState("h", h);
      setState("n", n);
      setParameter("V0", V0);
      setParameter("Cm", Cm);
      setParameter("G_Na_max", G_Na_max);
      setParameter("E_Na", E_Na);
      setParameter("G_K_max", G_K_max);
      setParameter("E_K", E_K);
      setParameter("G_L", G_L);
      setParameter("E_L", E_L);
      setParameter("Iapp_offset", Iapp_offset);
      setParameter("rate", rate);
      break;
    case MODIFY:
      V0 = getParameter("V0").toDouble();
      Cm = getParameter("Cm").toDouble();
      G_Na_max = getParameter("G_Na_max").toDouble();
      E_Na = getParameter("E_Na").toDouble();
      G_K_max = getParameter("G_K_max").toDouble();
      E_K = getParameter("E_K").toDouble();
      G_L = getParameter("G_L").toDouble();
      E_L = getParameter("E_L").toDouble();
      Iapp_offset = getParameter("Iapp_offset").toDouble();
      rate = getParameter("rate").toDouble();
      steps = static_cast (ceil(period * rate / 1000.0));
      V = V0;
      m = m_inf(V0);
      h = h_inf(V0);
      n = n_inf(V0);
      break;
    case PAUSE:
      break;
    case UNPAUSE:
      break;
    case PERIOD:
      period = RT::System::getInstance()->getPeriod() * 1e-6; // ms
      steps = static_cast (ceil(period * rate / 1000.0));
      break;
    default:
      break;
  }
}
```

4.2 DYNAMO Modules

! → A complete manual for the DYNAMO class is given in Appendix .2. In order to compile DYNAMO models from within RTXI, you will need to start RTXI with `sudo` privileges. DYNAMO is already installed on the Live CD but users installing RTXI manually should follow the instructions for enabling DYNAMO on their system.

→ Chapter 7
Installing DYNAMO

DYNAMO is a scripting language that allows you to create a RTXI module based on a dynamical system model described by ordinary differential equations. It uses a simple syntax for declaring the system states, parameters, state functions, and differential equations. DYNAMO models can be written using any plain text editor and are loaded into RTXI using the menu item **Modules→Load DYNAMO Module**. This calls the DYNAMO translator, which generates a C++ header and implementation file and compiles an RTXI module based on the `DefaultGUIModel` class. The generated header and implementation file are not readable since the computations are parsed into single multiply and add arithmetic operations such that intermediate values are given arbitrary variable names. After the translation step, the module is accessible through the regular **Load User Module** menu item. Unless the DYNAMO model file has been edited, it will not be re-translated and re-compiled.

A DYNAMO model file consists of a declaration section followed by a time block. The declaration section specifies the names and initial values of all quantities in the dynamical system. Every declaration is ended by a semicolon. The first declaration has to be a declaration of the system, which simply states the name of the model for informative purposes:

```
MODEL system_name
```

where *system_name* follows the rules for an identifier name.

After the system declaration, there follow a number of declarations of states, parameters, and functions. *Parameters* are constants during integration. The syntax for declaring a parameter is

```
PARAMETER name = default_value ''description''
```

where *description* is optional. The description string is there for convenience and is not read by any program. It is always optional, so it can be omitted. *States* are the components of the dynamical system whose values change over time and are computed by a difference or differential equation. There are several different kinds of states. *Scalar states* can only contain a scalar value and are declared with the keyword `STATE`:

```
STATE name = initial condition ''description'';
```

where *name* is the name of the state as the user sees it and *initial condition* is the default initial condition, a real constant. For example, in the following declaration,

```
STATE x = 0.1 "gating variable for inward conductance";
```

`x` is the name of the state, and 0.1 is the default initial condition. The above declaration will create a state variable which is integrated using an equation in the time block, described later in this section. The default method for integration is Euler's method. DYNAMO also supports a method we call *multiply-add-update*, in which the state variable being integrated is multiplied and added with the values returned by two functions dependent on `dt`. The method of integration can be specified with the `METHOD` attribute of the state definition, as follows:

```
STATE name = initial condition METHOD method_name;
```

where `method_name` can be either `euler` or `mau`, indicating Euler or multiply-add-update, respectively. Thus, our example can be changed to:

```
STATE x = 0.1 METHOD "mau" "gating variable for inward
   conductance";
```

External states are states whose value is either obtained through the data acquisition board (*external input*), or whose value is being output to the data acquisition board (*external output*). They are declared as:

```
EXTERNAL INPUT Vin1, Vin2; EXTERNAL OUTPUT Vout;
```

The input state can then be used in equations and expressions; the output state may not be used in expressions, and it must be assigned a value. The values of these external state variables are in terms of the units provided by the data acquisition board, usually volts. The order in which external input and output states are declared determines their assignment to physical channels of the data acquisition board. For example, in the above declaration, state *Vin1* will be assigned to input channel 0, and state *Vin2* will be assigned to input channel 1. Had they been declared in reverse order, then state *Vin1* would have been assigned to input channel 1, and state *Vin2* would have been assigned to input channel 0.

Functions are quantities that are statically dependent on other quantities in the system—unlike state equations, their equations are not permitted to use the previously computed value of the quantity. There are *scalar functions* which return a scalar value:

```
STATE FUNCTION name "description";
```

For all systems, there is only one "time", to be declared with the declaration `TIME`. The syntax is

```
TIME name;
```

The time variable that was declared with the above statement can be used anywhere in the model equations. Its value is a real number that represents the number of milliseconds elapsed since the beginning of the simulation. At each computational step it is incremented by `dt`.

A time block describes the equations which are in effect during the named time. Dynamic equations are all in the `AT TIME t`-block (assuming that the system time is called `t`). The equations in a time block are, if possible, sorted in order so they can be sequentially executed. If the equations contain a circular dependence, the sorting will fail. The DYNAMO translator can not solve algebraic loops (It can be claimed that in this case, the user has not written complete and consistent equations for the system). Other sanity tests (like that every derivative is assigned exactly once) are also performed.

Function expressions are specified in the following manner:

```
f1 = sin((1 + a) / 5)
```

The above statement will specify that at each iteration, the quantity *f1* will have the value computed with the given expression. *f1* then can be used in the expressions of other functions, differential equations, etc:

```
f2 = sin(f1 * 12)
```

On the right hand side, almost any scalar C expression is allowed: The exponential operator, denoted either ** or ^, has been added to the C syntax. The equation, which may run over several lines, is terminated with a semicolon. Further, the sequencing operator (the comma) is not allowed, since an expression sequence can hardly be an "equation". See Appendix .2.3, for a complete description of the possible arithmetic expressions. Standard mathematical functions are available with their usual names (log, cos, atan, etc@dots). See Appendix .2.3, for a list of all DYNAMO-supported mathematical functions.

Differential equations are specified in the following form:

```
d(state) = right-hand side;
```

Here `d( )` denotes the differential operator, and `d(x)` should here be interpreted as dx/dt. On the right-hand side, the same rules apply as for function expressions. In cases where the desired method of integration requires more than one equation (such as the multiply-add-update method), the equations are written as a comma-separated list enclosed in brackets. Thus:

```
d(state) = [ exp1, exp2 ];
```

A complete sample DYNAMO script is available in Appendix .3.

4.3 Developing Modules with Custom GUIs

The code that creates the GUI for a `DefaultGUIModel`-derived user module is located within the `createGUI()` member function. This function can be overloaded by a derived class to generate a custom GUI. RTXI uses the Qt platform, which is open-source and has a very well-documented API for creating GUI controls). Qt also uses a signal-and-slots architecture in which interactions with these GUI elements, such as pushing a button, emits a signal that can then be connected to a function. Whenever an event occurs, the slot function is executed. This architecture is made possible by a C++ preprocessor called MOC that generates additional *.cpp implementation and *.h header files for each class that has these features. These additional source files must also be listed in the Makefile.

There are several options available to users seeking to build their own GUIs. The simplest by far is to leave everything to `DefaultGUI`'s `createGUI` function. Customization options are also available through the `customizeGUI` function and overriding the `createGUI` function altogether. The following describes GUI creation in the context of the Neuron module.

! → The latter option is generally not recommended. If possible, rely on `customizeGUI`.

4.3.1 Leave everything to `createGUI`

The Neuron module's GUI is automatically generated by DefaultGUIModels's `createGUI` function. It operates by taking each `PARAMETER`, `STATE`, `EVENT`, and `COMMENT` variable in `vars[]` and creating a row in the GUI containing the variable name and stored value. After that, the utility buttons ("Pause", "Modify", and "Unload") are added to the bottom. `createGUI` should be adequate for simple modules, such as Neuron. In other words, no GUI configuration is necessary. For more complex modules, use `customizeGUI`.

4.3.2 Use `customizeGUI`

To understand how to use `customizeGUI`, it is important to understand the format of the GUI. For all of its windows, RTXI uses Qt. When RTXI is opened, one large window is displayed, and every module is opened as a subwindow within it. Each subwindow has its own formatting styles, set in source code. `DefaultGUIModel` uses `QGridLayout` to define elements within its subwindow. `QGridLayout` splits windows into cells indexed by row and column. Users can then place items in any cell or combination of contiguous cells. The `createGUI` function generates the column of variables with values and then the utility buttons and places them respectively in cells (1,0) and (2,0). The `customizeGUI` function takes the window created by `customizeGUI` and allows users to place additional elements in any positions that are not (1,0) and (2,0). Below is one example of a possible grid configuration.

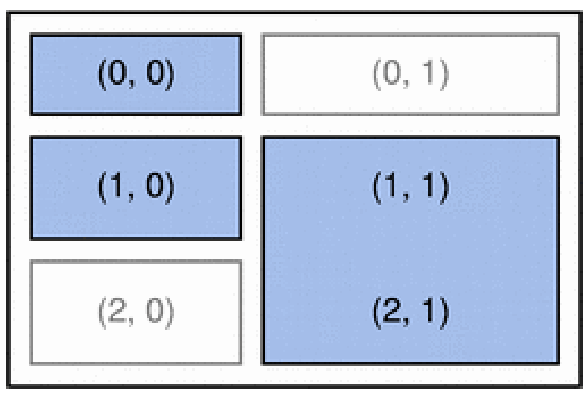

Figure 4.2: `QGridLayout` allows users to add elements to a window in a grid structure. The top-left corner is (0,0), and users can insert widgets spanning however many rows and columns in however many rows and columns.

As is, Neuron does not use `customizeGUI`. Suppose that we wanted to add two buttons in a row above the parameters. The syntax would be as follows:

```
void Neuron::customizeLayout(void) {
   QGridLayout *customlayout = DefaultGUIModel::getLayout();
```

```
QGroupBox *buttongroup = new QGroupBox;
QHBoxLayout *buttonlayout = new QButtonLayout;
QPushButton *aBttn = new QPushButton("Button A",
buttonlayout);
QPushButton *bBttn = new QPushButton("Button B",
buttonlayout);
buttongroup->setLayout(buttonlayout);

customlayout->addWidget(buttongroup, 0, 0)
setLayout(customlayout); }
```

The first and last two commands are necessary for `customizeLayout` to function. The first one grabs the window object created by `DefaultGUIModel::createGUI` and assigns it to `customlayout`. The middle section defines the elements to be added to the GUI. These are all Qt-defined classes for which documentation is freely available online (http://qt-project.org/doc/qt-4.8/). The second-to-last line assigns the GUI elements to the window. It specifies position (0,0), so the buttons will be displayed in the top left corner. The last line sets the GUI to the newly modified one.

5 Real-time Performance

RTAI provides several command line utilities for testing your real-time performance. These are available in your RTAI installation directory (usually `/usr/realtime`) in the `/testsuite` directory. There are both user and kernel space versions of the tests. If you already have RTAI kernel modules loaded, which is the case if you installed from the Live CD, you will only be able to run the userspace tests. These tests will be more accurate and you can see how the performance changes if you add additional processing load on your system as the test runs. The most important test is the latency test since this will verify that you are actually running a hard real-time system. Sample output for each of these tests is provided below. To run each test, within the appropriate directory, execute: `$ sudo ./run` If you installed RTXI from the Live CD, these tests are available from the RTXI folder in the Ubuntu Applications menu. You will need to type the root password. To stop execution of the test, use the keyboard shortcut `CTRL-C`.

There are many factors that affect your real-time performance. The maximum computational rate at which you can integrate differential equations for dynamic clamp is actually not dependent on absolute processor speed. For simple applications such as a single ion channel, similar results can be obtained on 200 MHz or 2 GHz processors. The limiting factors actually involve the design of the motherboard and chipset, the cost of reading and writing to a DAQ card, and the cost of accessing the hard disk when streaming data. Multi-processor systems or multicore processors also operate differently than single processors. RTXI allows the system to distribute processes among individual cores and does not assign any operations to particular cores. **!** → The user can use the `isolcpus` boot option to limit real-time operation to a single core and let all other operations be distributed among other available cores. It is also recommended that the computer be disconnected from a network, especially if is a wireless network, and to plot only the minimum number of signals in the Oscilloscope module as possible.

5.1 Latency Test

This test will verify the overall performance of your system. In oneshot mode, it measures the difference in time between the expected switch time and the time when a task is actually called by the scheduler. This test prints one line every second and gives you the minimum, average, and maximum latencies for that period as well the minimum and maximum overall latencies that occurred over the entire test. Open up some other programs, copy some files from one location to another, and load the network connection to see how it affects the latency. You should find slightly higher latencies with the user space test than the kernel space test. Your real-time performance is limited by the maximum latency (lat max) you can achieve and you generally dont want to be doing other tasks. You also should not see any overruns, which occurs when the latency completely exceeds your nominal period.

Negative time in the latency test is due to the fact that RTAI performs a calibration at startup that tries to minimize the jitter in the real-time task and anticipates the call.

```
## RTAI latency calibration tool ##
# period = 100000 (ns)
# avrgtime = 1 (s)
# do not use the FPU
# start the timer
# timer_mode is oneshot
RTAI Testsuite - KERNEL latency (all data in nanoseconds)
RTH| lat min| ovl min| lat avg| lat max| ovl max| overruns
RTD| -1524| -1524| -1442| -83| -83| 0
RTD| -1491| -1524| -1440| 3395| 3395| 0
RTD| -1489| -1524| -1441| 3381| 3395| 0
RTD| -1491| -1524| -1440| 3349| 3395| 0
```

If you periodically see an overrun (perhaps every 64 seconds) that results in a maximum latency of several hundred microseconds, you may have an SMI (System Maintenance Interrupt) issue. This feature can be found on certain chipsets e.g. Intel 82845 845. Disabling SMI can cause some computers to overheat and may damage those computers. Other latency killers are: heavy DMA activities (using the hard disk), using an accelerated Xserver, USB legacy support, power management (APM and ACPI), and CPU frequency scaling. If you have disabled all of these in the kernel already, check your BIOS and see if you can disable them there.

5.2 Preempt Test

This test is a stress utility that verifies the real-time schedulers under heavy processing load. This software combines the latency calibration task with a fast and slow task to have two levels of preemption.

```
RTAI Testsuite - UP preempt (all data in nanoseconds)
RTH| lat min| lat avg| lat max| jit fast| jit slow
RTD| -1781| -1267| 1930| 3228| 2724
RTD| -1782| -1143| 1930| 3228| 2724
RTD| -1782| -1135| 1930| 3228| 2724
RTD| -1782| -1166| 1930| 3228| 2724
```

5.3 Switches Test

This test provides information about the maximum amount of time RTAI needs to disable interrupts. The test uses a repeated sequence of suspend/resume and semaphore signal/wait calls under a heavy processing load. The switching time should be less than the maximum latency time. The real latency limitation is seldom due to RTAI but an intrinsic drawback of using a general purpose CPU for real-time applications.

```
Nov 11 20:49:02 dynamic kernel:  [ 9006.244009]
Nov 11 20:49:02 dynamic kernel:  [ 9006.244009] Wait for it .  .  .
Nov 11 20:49:02 dynamic kernel:  [ 9006.244009]
Nov 11 20:49:02 dynamic kernel:  [ 9006.244009]
Nov 11 20:49:02 dynamic kernel:  [ 9006.244009] FOR 10 TASKS: TIME 14 (MS),
   SUSP/RES SWITCHES 40000, SWITCH TIME (INCLUDING FULL FP SUPPORT) 339 (ns)
Nov 11 20:49:02 dynamic kernel:  [ 9006.244009] FOR 10 TASKS: TIME 14 (MS), SEM
   SIG/WAIT SWITCHES 40000, SWITCH TIME (INCLUDING FULL FP SUPPORT) 347 (ns)
Nov 11 20:49:02 dynamic kernel:  [ 9006.244009] FOR 10 TASKS: TIME 14 (MS),
   RPC/RCV-RET SWITCHES 40000, SWITCH TIME (INCLUDING FULL FP SUPPORT) 385 (ns)
```

6 Utilities and Experimental Suites

Outside the core modules, such as the data recorder, RTXI provides a number of useful utilities and experimental suites. While most utilities are designed as an optional aid, RTXI has a growing library of experimental suites intended to perform complete biological experiments. The experimental suites can be augmented with user-written custom modules to create complex experimental systems outside the intended use of these suites. For RTXI version 1.4, the Patch Clamp Experimental Suite is the only suite currently available. However, a number of other suites, such as a digital camera and imaging suite, are in construction.

6.1 Signal Utilities

The signal utilities are modules that can output several types of standard signals. These modules are useful for diagnostic purposes, such as testing a data acquisition board, or can be used as part of an experimental protocol, such as white noise injection into a neuronal network.

6.1.1 Signal Generator

Overview

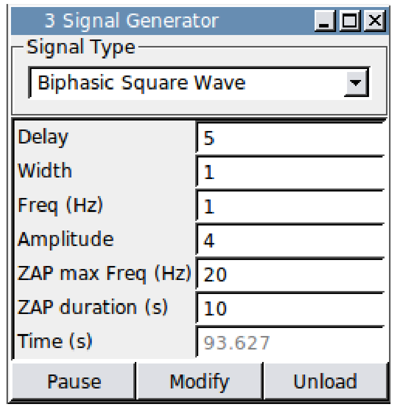

Figure 6.1: The signal generator module is set to output a 1s long biphasic square wave every 5s, with an amplitude of 4. A tutorial is provided to replicate this figure.

→ Chapter 6.1.1
Signal Generator Tutorial

The signal generator module can generate a number of different signals, and each signal is modified through several parameters. The available signals and their corresponding parameters are described below.

- Sine Wave: requires frequency and amplitude
- Monophasic Square Wave: requires delay, pulse width, and pulse amplitude
- Biphasic Square Wave: requires delay, pulse width, and pulse amplitude
- Sawtooth Wave: requires delay, pulse width, and maximum amplitude
- ZAP Stimulus: needs starting and ending frequencies, amplitude, and duration of ZAP

All signals are continuous except for the ZAP stimulus, which has a specified duration.

Output Channels

Signal Waveform : Signal output

Parameters

Delay (s) :	Time before each square wave or sawtooth wave starts
Width (s) :	Width of each square wave and sawtooth signal
Freq (Hz) :	frequency of sine wave and starting frequency for ZAP stimulus
Amplitude :	amplitude for all signals
ZAP max Freq (Hz) :	maximum frequency for ZAP stimulus
ZAP duration (s) :	duration of ZAP stimulus

Tutorial

In this tutorial, the Signal Generator will be used to output a biphasic square wave every 5s. Each phase will be 1s long, with an amplitude of 4. The oscilloscope will be used to visualize the signal, with the option of outputting the signal through a data acquisition board. The mimic tutorial continues where this tutorial leaves off,so it is suggested that is done next.

→ Chapter 6.1.2
Mimic Tutorial

1. Open the Signal Generator module through the menu: **Utilities→Signals→Signal Generator**

→ Chapter 2.2.2
Oscilloscope

2. Open the oscilloscope module through the menu: **System→Oscilloscope**
3. Set the correct parameters for the desired biphasic square wave in the signal generator GUI as in Figure 6.1.1:
   - Set the signal type to **Biphasic Square Wave** using the pull down menu under **Signal Type**
   - To output the signal every 5s, set **Delay** to 5
   - To set each phase to be 1s long, set **Width** to 1
   - To set the amplitude to 4V, set **Amplitude** to 4
   - Save changes by clicking the **Modify** button
4. Set up the oscilloscope to visualize the signal by right clicking on the oscilloscope and selecting **Properties**
   - Make sure the **Channel** Tab is currently selected
   - Select **Signal Generator** under the channel pulldown menu (This probably will already be selected)
   - Select **Output** in the following pulldown menu on the right
   - Select **Signal Waveform** in the following pulldown menu on the right (This probably will already be selected)
   - Activate this channel by hitting the toggle button **Active**
   - Change the scale to **1 V/div** in the Scale pulldown menu
   - Click the **Apply** button to save the changes
   - Select the **Display** tab
   - Change the time scale to **1 s/div**

- Change the refresh rate to 50 for a smoother looking output
- Click the **Apply** button to save the changes

5. *Optional* Output the signal generator signal through the analog output of a data acquisition board
    - Open the connector module through the menu: **System→Connector**
    - In the Output Block, select **Signal Generator** under block and **Signal Waveform** under Channel
    - In the Input Block, select your data acquisition board (i.e. /dev/comedi0 or NI-PCI6259) and the desired channel (i.e. Analog Output 0)
    - Connect the two by toggling the arrow button
    - Make sure the desired channel is active through the System Control Panel

→ Chapter 2.2.1
System Control Panel

6. Start the signal by untoggling the **Pause** button
7. The biphasic square wave should now be seen on the oscilloscope as in Figure 6.1.1

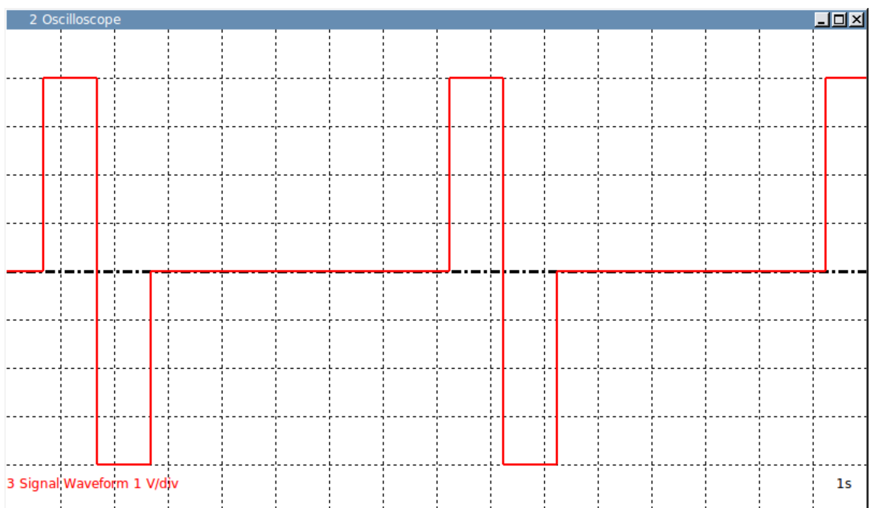

Figure 6.2: The oscilloscope is used to visualize a biphasic square wave being output by the Signal Generator Module

6.1.2 Mimic

Overview

→ Chapter 6.1.2
Mimic Tutorial

The Mimic module is the simplest signal generator available in RTXI. Mimic's main function is to apply a gain and/or offset to a signal it receives. By itself, Mimic can also be used output a continuous signal.

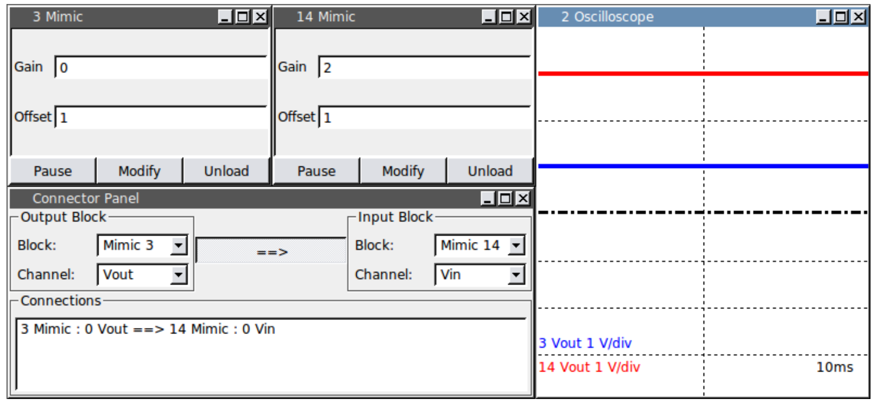

Figure 6.3: Two Mimic modules are connected with their outputs displayed on the oscilloscope. Mimic-3, as shown in the blue oscilloscope trace, is outputting an offset of 1. As shown in the connector, Mimic-3's output (Vout) is connected to Mimic-14's (Vin). Mimic-14 applies the set gain and offset to Mimic-3's signal (1*2+1), resulting in an output of 4 shown in red. A tutorial is provided to replicate this figure.

Input Channels

Vin : Gain and offset applied to this input to calculate output

Output Channels

Vout : Output calculated by multiplying input signal by gain and adding offset

Parameters

Gain : Factor by which input is multipled

Offset : Factor added to the input. If no input is connected (i.e. input = 0), offset solely determines output

Tutorial

This tutorial is meant to be performed after the Signal Generator tutorial. The mimic module will be used to add a gain of 0.5 and offset of 1 to the biphasic square wave output by the signal generator.

1. Use the signal generator to output a biphasic square wave, where each phase is 1s long and has an amplitude of 4. Output this signal to the oscilloscope. These steps are covered in the Signal Generator tutorial.

→ Chapter 6.1.1
Signal Generator Tutorial

2. Open up a Mimic module through the menu: **Utilities→Signals**$rightarrow$**Mimic**

→ Chapter 2.2.4
Connector

3. Open the Connetor Panel through the menu: **System→Connector**
4. Connect the output of the Signal Generator module to the input of Mimic

- In the output block on the left side of the **Connector Panel**, select the Signal Generator module. The channel option should default to its only option, **Signal Waveform**

- In the input block on the right side of the **Connector Panel**, select the other Mimic module. The channel option should default to its only option, **Vin**

5. Set up the oscilloscope to visualize the output of the Mimic module, in conjunction with the output of Signal Generator, by right clicking on the oscilloscope and selecting **Properties**

    - Make sure the **Channel** Tab is currently selected

    - Select the Mimic module under the channel pulldown menu

    - Select **Output** in the following pulldown menu, and make sure **Vout** is the selected output

    - Activate the channel by hitting the toggle button **Active**

    - Change the scale to **1 V/div** in the Scale pulldown menu

    - Change the color to **Blue**

    - Click the **Apply** button to save the changes

6. Set the **Gain to 2** and the **Offset** to 1

7. Untoggle the **Pause** button

8. The output should now appear on the oscilloscope as in Figure 6.1.2

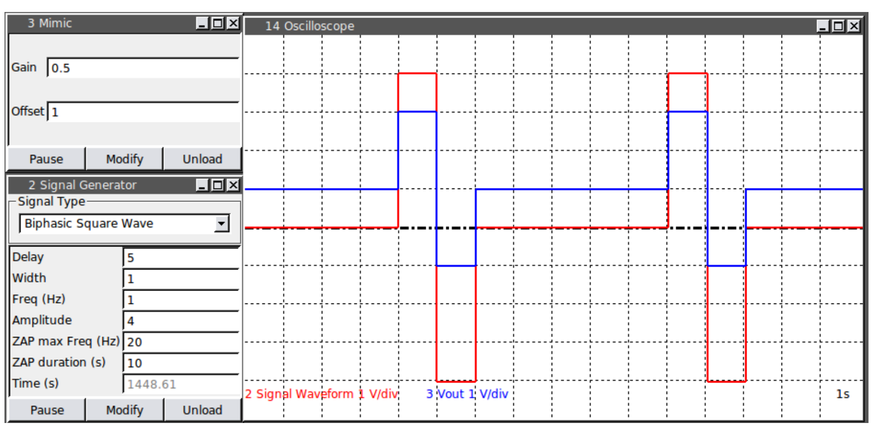

Figure 6.4: The Signal Generator module is outputting a biphasic square wave (red trace). This output is connected to the input of the Mimic module. The mimic module is applying a gain of 0.5 and an offset of 1 to the biphasic square, and outputs this modified signal (blue trace).

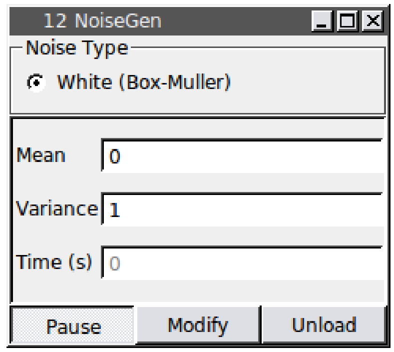

Figure 6.5: The noise generator module outputs Gaussian white noise.

6.1.3 Noise Generator

Overview

The noise generator module can continuously generate Gaussian white noise computed using the Box-Muller method.

Output Channels

Noise Waveform : Noise output

Parameters

Mean : Mean value of noise output

Variance : The given variance used in noise calculated

6.1.4 Wave Maker

Overview

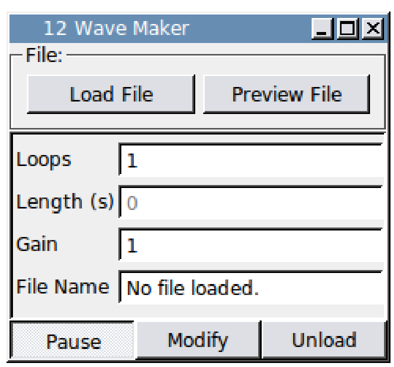

Figure 6.6: The wave maker module allows the output of a pre-recorded signal from an ASCII file.

The wavemaker module loads data from an ASCII formatted file. It samples one value from the file on every time step and generates an output signal. The module computes the time length of the waeform based on the current

→ Chapter 2.2.1
System Control Panel

real-time period, set through the (System Control Panel. User-generated modules can be tested using the wavemaker module, by simulating real-time acquisition of data.

Output Channels

Output : Values read from the ASCII file

Parameters

Loops : Number of times to repeat the waveform, looping back to the beginning

Gain : Multiplicative gain to apply to the waveform values

6.2 Filter Utilities

RTXI provides modules to filter incoming signal, which are most commonly used for analog signals received through a data acquisition board.

6.2.1 FIR Filter

Overview

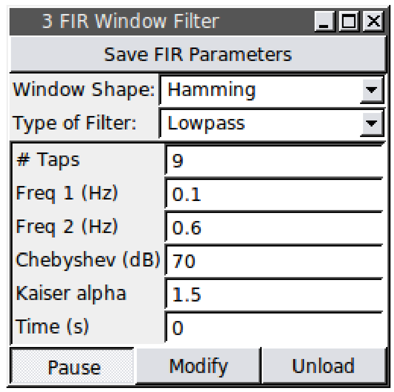

Figure 6.7: The wave maker module allows the output of a pre-recorded signal from an ASCII file.

The Finite Impulse Response Filter module creates an in-line FIR filter that can be applied to any signal in RTXI. Given the desired number of filter taps (filter order + 1), it computes the impulse response for a lowpass, highpass, bandpass, or bandstop filter using the window method. For a lowpass or highpass filter, the module uses the first frequency as the cut-off frequency. For a bandpass or bandstop filter, both input frequencies are used to define the frequency band. The module initially computes an ideal FIR filter to which you can apply a Triangular (or Bartlett), Hamming, Hann, Kaiser, or Dolph-Chebyshev window. The Hann window is not to be confused with the Hanning window (see MATLABs hann() vs. hanning() functions). To apply no window to the filter, choose the Rectangular filter. The Kaiser and Chebyshev windows each take a parameter that determines the attenuation of the sidelobes in the filter. The algorithms only accept an odd number of filter taps. If an even number is entered, the module will automatically add 1 to the number of filter taps.

Input Channels

Input : Signal to be filtered

Output Channels

Output : Filtered input signal

Parameters

Window Shape :
- Rectangular

- Triangular (Bartlett)
- Hamming
- Hann
- Chebyshev
- Kaiser

Type of Filter :
- Highpass
- Lowpass
- Bandpass
- Bandstop

\# Taps :

Freq 1 (Hz) :

Chebyshev (dB) :

Kaiser alpha :

States

Time (s) : Time elapsed, in seconds, since filter was started

6.2.2 IIR Filter

Overview

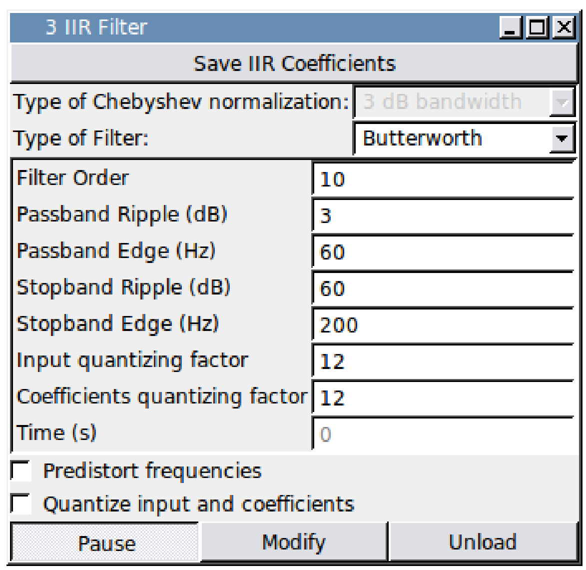

Figure 6.8: The infinite impulse response filter module can be used to filter any signal in RTXI.

This module computes coefficients for three types of filters. They require the following parameters:

Butterworth: passband edge The Butterworth filter is the best compromise between attenuation and phase response. It has no ripple in the pass band or the stop band, and because of this is sometimes called a maximally flat filter. The Butterworth filter achieves its flatness at the expense of a relatively wide transition region from pass band to stop band, with average transient characteristics.

Chebyshev: passband ripple, passband edge The Chebyshev filter has a smaller transition region than the same-order Butterworth filter, at the expense of ripples in its pass band. The filter minimizes the height of the maximum ripple. If you use a Chebyshev filter, you should also choose the type of normalization to apply.

Elliptical: passband ripple, stopband ripple, passband edge, stopband edge An Elliptical (Cauer) filter has a shorter transition region than the Chebyshev filter because it allows ripples in both the stop and pass bands, giving a much higher rate of attenuation in the stop band. Elliptical filters give better frequency discrimination, but have a degraded transient response.

You may save the computed coefficients and the filters parameters to a file.

Input Channels

Input : Signal to be filtered

Output Channels

Output : Filtered input signal

Parameters

Filter order : An integer for the desired order for the filter

Passband Ripple (dB) :

Passband Edge (Hz) :

Stopband Ripple (dB) :

Stopband Edge (Hz) :

Input quantizing factor : The number of bits to quantize the input signal to

Coefficients quantizing factor : The number of bits to quantize the filter coefficients to

States

Time (s) : Time elapsed, in seconds, since filter was started

6.3 Patch Clamp Experimental Suite

The patch clamp experimental suite is a group of modules designed to aid an electrophysiologist during patch clamping. The suite includes modules for interfacing with patch clamp amplifiers, monitoring electrode resistance during gigaseal formation, and running voltage clamp experiments.

6.3.1 Amplifier Control Modules

AM Systems Model 2400

Axon Axopatch 1-D

Axon Axopatch 200 Series

Axon Multiclamp 700 Series

6.3.2 Membrane Test

6.3.3 Clamp Protocol

Appendix

1. Licensing Information
2. DYNAMO Scripting Language
3. Information for Developers

.1 Licensing Information

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program.

If not, see "http://www.gnu.org/licenses".

.1.1 GNU General Public License

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. ¡http://fsf.org/¿ Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a ”modified version” of the earlier work or a work ”based on” the earlier work.

A ”covered work” means either the unmodified Program or a work based on the Program.

To ”propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To ”convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays ”Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The ”source code” for a work means the preferred form of the work for making modifications to it. ”Object code” means any non-source form of a work.

A ”Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The ”System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A ”Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The ”Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

¡one line to give the program's name and a brief idea of what it does.¿ Copyright (C) ¡year¿ ¡name of author¿

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see ¡http://www.gnu.org/licenses/¿.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

¡program¿ Copyright (C) ¡year¿ ¡name of author¿ This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see ¡http://www.gnu.org/licenses/¿.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read ¡http://www.gnu.org/philosophy/why-not-lgpl.html¿.

.1.2 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. ¡http://fsf.org/¿ Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

.2 DYNAMO Scripting Language

Content in this section is adapted from the DYNAMO Reference Manual Edition 1.9.7 and the Dynamical Language Reference Manual Edition 1.0.0, both written by Ivan Raikov at the Georgia Institute of Technology in 2005.

.2.1 Using DYNAMO with RTXI

DYNAMO models can be edited using your choice of text editor and compiled into RTXI models from within RTXI by selecting the menu item Modules → Load DYNAMO Module. The DYNAMO model is first parsed into C++ header and implementation files, which are then compiled into shared object libraries similar to other RTXI modules. After the initial compilation, the model can be loaded using the menu item Modules → Load User Module, without repeating the parsing step. User-specified variable names and equations are not preserved in the resulting CPP files so changes to your model should be made to the original DYNAMO module. If changes are made, the DYNAMO module should be re-parsed and compiled. If you started RTXI from the terminal, the progress of the parsing and compilation steps, as well as any errors in your DYNAMO syntax, will be displayed there.

There should already be a working DYNAMO script located in /usr/bin, but the following instructions will allow you to compile the DYNAMO translation script from scratch in Ubuntu 8.10/9.04/9.10.

```
$ sudo apt-get install mlton
$ cd /rtxi/dynamo
$ mllex dl.lex
$ mlyacc dl.grm
$ mlton dynamo.mlb
$ sudo cp dynamo /usr/bin
```

.2.2 Running DYNAMO from the terminal

The DYNAMO translator is run with the command `dynamo` followed by names of files to be translated, and options that specify the type of output. For example:

```
dynamo --matlab morris-lecar.dynamo
```

This command specifies that the DYNAMO translator should translate file `morris-lecar.dynamo` to MATLAB code. Below is a summary of all options accepted by the DYNAMO translator:

Table 1: DYNAMO translator options

Options(s)	**Description**
`-h, --help`	describes the options
`-version, --release`	show version information
`-o [FILE], --output[=FILE]`	set the prefix of output file(s)
`-e [FILE], --error-output[=FILE]`	redirect error messages
`--eqdfg[=FILE]`	output the equation DFG
`--exdfg[=FILE]`	output the expression DFG
`-r [FILE], --mrci[=FILE]`	output MRCI model
`-x [FILE], --rtxi[=FILE]`	output RTXI model
`-m [FILE], --matlab[=FILE]`	output Matlab model
`--simulink[=FILE]`	output Simulink model

.2.3 DYNAMO Syntax

A *model description* describes the dynamical system to be studied. It consists of *declarations* and *equations*, which have a certain mathematical meaning, and are not to be confused with assignment statements in programming languages. These equations do not have to be entered in any particular order; they are automatically sorted by the translator so that functions are computed before the state equations that use them.

Concepts

A *user-visible quantity* is something the user can access, change (provided that it makes sense), and which may have a documentation associated with it. DYNAMO allows all user-visible quantities to be manipulated by the user, and in that sense it does not distinguish between quantities with different mathematical meanings (such as constants, states, functions, etc.)

The DYNAMO model description language is always case sensitive. The name 'A' is different from 'a'. The language has a set of reserved words, which are always written in capital letters.

The names given to user-visible quantities must obey the following rules: an identifier consists of a sequence of alpha-numerical characters, the first of which is a character, and the underscore '_' is counted as a character.

Once an identifier is used to declare a state, parameter etc., it may not be used for any other purpose.

Comments may be written anywhere. The syntax is as in the C and C++ programming languages: either enclosed between '/*' and '*/', or from '//' and to end-of-line.

Structure of a DYNAMO Model

A DYNAMO model file consists of a declaration section followed by a time block. The declaration section consists of a number of declarations. Every declaration is ended by a semicolon. The first declaration has to be a declaration of the model, such as:

```
MODEL system_name
```

where *system_name* follows the rules for an identifier name.

After the system declaration, there follow a number of declarations of states, parameters, and functions. There has to be exactly one time declaration.

Declarations

The declaration section specifies the names and initial values of all quantities in the dynamical system. Every declaration is ended by a semicolon. The first declaration has to be a declaration of the system, which simply states the name of the model for informative purposes.

After the model name, there follows a number of declarations for *parameters*, *states*, *external states* and *functions*. The declaration section concludes with exactly one *time declaration*.

Parameters are constants during integration. The syntax for declaring a parameter is

```
PARAMETER name = default_value ``description''
```

where *description* is optional. The description string is there for convenience and is not read by any program. It is always optional, so it can be omitted.

Vector parameters are constant vector values. The syntax for declaring a vector parameter is

```
VECTOR PARAMETER name = ( element0, element1, ... )
   ‘‘description’’;
```

The parameter declared in this manner will have a vector value initialized with the scalar elements supplied by the user. The size of the vector is equal to the number of initialization elements given, and remains constant throughout the simulation.

States are the components of the dynamical system whose values change over time and are computed by a difference or differential equation. There are several different kinds of states. *Scalar states* can only contain a scalar value and are declared with the keyword `STATE`:

```
STATE name = initial condition ‘‘description’’;
```

where *name* is the name of the state as the user sees it and *initial condition* is the default initial condition, a real constant.

For example, in the following declaration,

```
STATE x = 0.1 "gating variable for inward conductance";
```

`x` is the name of the state, and 0.1 is the default initial condition.

The above declaration will create a state variable which is integrated using an equation in the time block, described later in this section. The default method for integration is Euler's method. DYNAMO also supports a method we call *multiply-add-update*, in which the state variable being integrated is multiplied and added with the values returned by two functions dependent on `dt`. The method of integration can be specified with the `METHOD` attribute of the state definition, as follows:

```
STATE name = initial condition METHOD method_name;
```

where `method_name` can be either `euler` or `mau`, indicating Euler or multiply-add-update, respectively. Thus, our example can be changed to:

```
STATE x = 0.1 METHOD "mau" "gating variable for inward
   conductance";
```

Integer states are exactly the same as scalar states, only they can only have an integer initial value, and only the integer part of their equation is assigned to them.

Vector states are states that can only hold a vector value of fixed size:

```
VECTOR STATE name = ( initial0, initial1, ... ) ‘‘description’’;
```

The state declared above will have an initial value that is a vector initialized with the scalar elements supplied by the user. The size of the vector is equal to the number of initialization elements given, and remains constant throughout the simulation. Vector states cannot be assigned equations that return a vector of size different than that of their initial value.

Discrete states are state quantities whose value can only be one of several enumerated (discrete) values:

```
DISCRETE STATE status = ( inactive, threshold, active
   ) ``description'';
```

The names of the possible values are supplied by the user, and are implicitly assigned integer values starting from one and incrementing by one. These default integer values can be overridden as follows:

```
DISCRETE STATE status = ( inactive=5, threshold=10,
   active=20 );
```

External states are states whose value is either obtained through the data acquisition board (*external input*), or whose value is being output to the data acquisition board (*external output*). They are declared as:

```
EXTERNAL INPUT Vin1, Vin2; EXTERNAL OUTPUT Vout;
```

The input state can then be used in equations and expressions; the output state may not be used in expressions, and it must be assigned a value. The values of these external state variables are in terms of the units provided by the data acquisition board, usually volts. The order in which external input and output states are declared determines their assignment to physical channels of the data acquisition board. For example, in the above declaration, state *Vin1* will be assigned to input channel 0, and state *Vin2* will be assigned to input channel 1. Had they been declared in reverse order, then state *Vin1* would have been assigned to input channel 1, and state *Vin2* would have been assigned to input channel 0.

Functions are quantities that are statically dependent on other quantities in the system—unlike state equations, their equations are not permitted to use the previously computed value of the quantity. There are *scalar functions* which return a scalar value, and *vector functions* which return a vector value.

```
STATE FUNCTION name "description";
```

and

```
VECTOR FUNCTION name "description";
```

For all systems, there is only one "time", to be declared with the declaration `TIME`. The syntax is

```
TIME name;
```

The time variable that was declared with the above statement can be used anywhere in the model equations. Its value is a real number that represents the number of milliseconds elapsed since the beginning of the simulation. At each computational step it is incremented by `dt`.

The declarations section may also contain *function lookup tables.* These define a function whose value is computed by interpolating datapoints in a table indexed by a state variable. This feature can greatly speed up computation. An example lookup table definition is shown below.

```
TABLE FUNCTION F1(v) = (1 + tanh(v)), LOW = -10.1, HIGH
   = 10.1, STEP = 0.1, DEPENDENCY = F2;
```

The various syntactic components of this statement have the following meanings:

- `TABLE FUNCTION F(v)`—declaration of a function called `F1`, which has one argument, `v`. Note that the function argument is only to be used inside the function expression; it is *NOT* (or doesn't have to be) the name of the variable used for looking up datapoints in the function table.
- `(1 + tanh(v))`—the actual function expression. See Appendix .2.3, for details on arithmetic expressions in DYNAMO. Note the use of our function argument.
- `LOW=-10.1,HIGH=10.1,STEP=0.1`—the lower boundary of interpolation datapoints, the upper boundary of interpolation datapoints, and the interval for datapoints between the two boundaries. DYNAMO will compute datapoints starting at the lower boundary and reaching to the upper boundary using the given step.
- `DEPENDENCY=F2`—the name of the dependency. This can be a function, state, parameter, etc. At run-time, the value of this quantity will be computed first, then it will be given as an input to the interpolation function.

Time Blocks

A time block describes the equations which are in effect during the named time. Dynamic equations are all in the `AT TIME t`-block (assuming that the system time is called `t`). The equations in a time block are, if possible, sorted in order so they can be sequentially executed. If the equations contain a circular dependence, the sorting will fail. The DYNAMO translator can not solve algebraic loops (It can be claimed that in this case, the user has not written complete and consistent equations for the system). Other sanity tests (like that every derivative is assigned to exactly once) are also performed.

Function expressions are specified in the following manner:

```
f1 = sin((1 + a) / 5)
```

The above statement will specify that at each iteration, the quantity *f1* will have the value computed with the given expression. *f1* then can be used in the expressions of other functions, differential equations, etc:

```
f2 = sin(f1 * 12)
```

On the right hand side, almost any scalar C expression is allowed: The exponential operator, denoted either ∗∗ or ^, has been added to the C syntax. The equation, which may run over several lines, is terminated with a semicolon. Further, the sequencing operator (the comma) is not allowed, since an expression sequence can hardly be an "equation". See .2.3, for a complete description of the possible arithmetic expressions.

Standard mathematical functions are available with their usual names (log, cos, atan, etc@dots). See .2.3, for a list of all DYNAMO-supported mathematical functions.

Differential equations are specified in the following form:

```
d(state) = right-hand side;
```

Here `d( )` denotes the differential operator, and `d(x)` should here be interpreted as dx/dt. On the right-hand side, the same rules apply as for function expressions.

In cases where the desired method of integration requires more than one equation (such as the multiply-add-update method), the equations are written as a comma-separated list enclosed in brackets. Thus:

```
d(state) = [ exp1, exp2 ];
```

For difference equations, the dynamic equation takes the form

```
q(x) = right-hand side;
```

Here we think of `q` as the forward shift operator: `q(x(t)) = x(t + 1)`.

A time block may also contain a block of arbitrary C code. It should occur first in the time block. It will be executed before the equations. There is presently no possibility to put raw C code *after* the equations are executed. (However, one way to circumvent this would be to use function calls, e.g. of the type *d(z) = f(z) + function();* where `function` always returns `0`.)

Declarations of states etc.: are also allowed in the time block, as long as a quantity is declared before it is referenced. This feature is necessary for certain machine generated system descriptions (e.g.: using macros). It is not the recommended practice to take advantage of it in manually written system descriptions.

One *time* is predefined: `START`. `AT TIME START` contains equations and/or C-code to be executed before the main integration. For example, functions can be set to their initial values in this section. If an error is detected the C statement *return -1;* should be executed. This will stop the simulation.

Dynamic equations (differential equations and difference equations) are only allowed in the `AT TIME t` block. Algebraic equations may occur only in the `AT TIME t` and the `AT TIME START` block.

Expressions and Operators in the Modeling Language

An *expression* is any sequence of operators and operands in the C programming language that produces a value. The simplest expressions are parameter, function, and state names, which yield values directly. Other expressions combine operators and subexpressions to produce values.

An expression within parentheses has the same value as the expression without parentheses would have. Any expression can be delimited by parentheses to change the precedence of its operators.

All declared quantities can be used in conjunction with C operators to create more complex expressions. The following table presents the set of C operators.

Table 2: Expressions and Operators in DYNAMO

<table>
<tr><th>Operator</th><th>Example</th><th>Description/Meaning</th></tr>
<tr><td>+ [unary]</td><td>+a</td><td>Value of a</td></tr>
<tr><td>- [unary]</td><td>-a</td><td>Negative of a</td></tr>
<tr><td>~</td><td>~a</td><td>One's complement of a</td></tr>
<tr><td>++ [prefix]</td><td>++a</td><td>The value of a afte increment by one</td></tr>
<tr><td>++ [postfix]</td><td>a++</td><td>The value of a before increment by one</td></tr>
<tr><td>-- [prefix]</td><td>–a</td><td>The value of a after decrement by one</td></tr>
<tr><td>-- [postfix]</td><td>a–</td><td>The value of a before decrement by one</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>+ [binary]</td><td>a + b</td><td>a plus b</td></tr>
<tr><td>- [binary]</td><td>a - b</td><td>a minus b</td></tr>
<tr><td>* [binary]</td><td>a * b</td><td>a times b</td></tr>
<tr><td>/</td><td>a / b</td><td>a divided by b</td></tr>
<tr><td>%</td><td>a % b</td><td>Remainder of a/b</td></tr>
<tr><td>>></td><td>a >> b</td><td>a, right-shifted b bits</td></tr>
<tr><td><<</td><td>a<< b</td><td>a, left-shifted b bits</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td><</td><td>a < b</td><td>1 if a ¡ b; 0 otherwise</td></tr>
<tr><td>></td><td>a > b</td><td>1 if a ¿ b; 0 otherwise</td></tr>
<tr><td><=</td><td>a <= b</td><td>1 if a ¡= b; 0 otherwise</td></tr>
<tr><td>>=</td><td>a >= b</td><td>1 if a ¿= b; 0 otherwise</td></tr>
<tr><td>==</td><td>a == b</td><td>1 if a equal to b; 0 otherwise</td></tr>
<tr><td>!=</td><td>a != b</td><td>1 if a not equal to b; 0 otherwise</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>& [binary]</td><td>a & b</td><td>Bitwise AND of a and b</td></tr>
<tr><td>|</td><td>a | b</td><td>Bitwise OR of a and b</td></tr>
<tr><td>^</td><td>a ^ b</td><td>Bitwise XOR (exclusive OR) of a and b</td></tr>
<tr><td>&&</td><td>a && b</td><td>Logical AND of a and b (yields 0 or 1)</td></tr>
<tr><td>||</td><td>a || b</td><td>Logical OR of a and b (yields 0 or 1)</td></tr>
<tr><td>!</td><td>!a</td><td>Logical NOT of a (yields 0 or 1)</td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>?:</td><td>a ? e1 : e2</td><td>Expression e1 if a is nonzero; Expression e2 if a is zero</td></tr>
<tr><td>=</td><td>a = b</td><td>a, after b is assigned to it</td></tr>
<tr><td>+=</td><td>a += b</td><td>a plus b (assigned to a)</td></tr>
<tr><td>-=</td><td>a -= b</td><td>a minus b (assigned to a)</td></tr>
<tr><td>*=</td><td>a *= b</td><td>a times b (assigned to a)</td></tr>
<tr><td>/=</td><td>a /= b</td><td>a divided by b (assigned to a)</td></tr>
<tr><td>%=</td><td>a %= b</td><td>Remainder of a/b (assigned to a)</td></tr>
<tr><td>>>=</td><td>a >>= b</td><td>a, right-shifted b bits (assigned to a)</td></tr>
<tr><td><<=</td><td>a <<= b</td><td>a, left-shifted b bits (assigned to a)</td></tr>
<tr><td>&=</td><td>a &= b</td><td>a and b (assigned to a)</td></tr>
<tr><td>|=</td><td>a |= b</td><td>a OR b (assigned to a)</td></tr>
<tr><td>^=</td><td>a ^= b</td><td>a XOR b (assigned to a)</td></tr>
</table>

The C operators fall into the following categories:

- Unary operators, which take a single operand.
- Postfix operators, which follow a single operand.
- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and resolves to the value of either the second or third expression, depending on the result of the evaluation of the first expression.

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

```
x = 8 + 4 * 2; /* x is assigned 16, not 24 */
```

The previous statement is equivalent to the following:

```
x = 8 + ( 4 * 2 );
```

Using parenthesis in an expression alters the default precedence. For example:

```
x = (8 + 4) * 2; /* (8 + 4) is evaluated first */
```

In an unparenthesized expression, operators of higher precedence are evaluated before those of lower precedence. Consider the following expression:

```
A + B * C
```

The identifiers B and C are multiplied first because the multiplication operator (*) has higher precedence than the addition operator (+).

A useful construction is the ternary *? :* operator. A good example of its use may be

```
step = t < t0 ?  0 :  1;
```

This expression states that `step` has the value 0 if `t < t0`, else the value 1.

Table 3: Mathematical Functions in DYNAMO

Function	**Description/Meaning**
`asin`	Arc sine of x
`atan`	Arc tangent of x
`atan2`	Arc tangent of two variables
`acos`	Arc cosine of x
`abs`	Absolute value of an integer x
`ceil`	Smallest integral value not less than x
`cos`	Cosine of x
`cosh`	Hyperbolic cosine of x
`cube`	x cubed
`exp`	e raised to the power of x
`floor`	Largest integral value not greater than x
`fabs`	Absolute value of a floating-point number x
`log`	Natural logarithm of x
`log10`	Base-10 logarithm of x
`pow`	x to the yth power
`sin`	Sine of x
`sinh`	Hyperbolic sine of x
`sqrt`	Square root of x
`sqr`	x squared
`tanh`	Hyperbolic tangent of x
`tan`	Tangent of x

.3 DYNAMO Example

```
/* This model is used to calculate the membrane potential assuming some initial
   state.  The calculation is based on sodium ion flow, potassium ion flow and
   leakage ion flow.  (Hodgkin, A. L. and Huxley, A. F. (1952) "A Quantitative
   Description of Membrane Current and its Application to Conduction and
   Excitation in Nerve" Journal of Physiology 117:  500-544)

*/

SYSTEM Hodgkin_Huxley;

PARAMETER C_m = 1.0 "uF/cm^2";

// Maximum possible sodium conductance
PARAMETER g_Na = 120.0 "mS/cm^2";
// Maximum possible potassium conductance
PARAMETER g_K = 36.0 "mS/cm^2";
// Maximum possible leakage conductance
PARAMETER g_L = 0.3 "mS/cm^2";
// Sodium membrane potential
PARAMETER E_Na = 50.0 "mV";
// Potassium membrane potential
PARAMETER E_K = -77.0 "mV";
// Leakage membrane potential
PARAMETER E_L = -54.4 "mV";
// The time range during which I_stim will be applied to the system
PARAMETER t_on = 0 "Beginning time for I_stim";
PARAMETER t_off = 10 "Ending time for I_stim";
// The magnitude of the stimulus current
PARAMETER I_stim_mag = 10;

STATE V = -65.0;

STATE h = 0.9;
STATE m = 0.1;
STATE n = 0.1;

STATE FUNCTION I_stim; // Stimulus current
// Ionic currents across the membrane
STATE FUNCTION I_Na "Na+ current:  I_Na (V, m, h)";
STATE FUNCTION alpha_m "alpha_m(V)";
STATE FUNCTION beta_m "beta_m(V)";
STATE FUNCTION alpha_h "alpha_h(V)";
STATE FUNCTION beta_h "beta_h(V)";
STATE FUNCTION I_K "K+ current:  I_K (V, n)";
STATE FUNCTION alpha_n "alpha_n(V)";
STATE FUNCTION beta_n "beta_n(V)";
STATE FUNCTION I_L "Leak current:  I_L (V)";
```

```
// This will have the value of V (total membrane potential)
EXTERNAL OUTPUT Vout1;

TIME t;

AT TIME t:

alpha_m = (0.1* (V + 40))/(1 - exp(-(V + 40)/(10)));
beta_m = 4 * exp(-(V + 65)/(20));
alpha_h = 0.07 * exp(-(V + 65)/(20));
beta_h = 1/(1 + exp(-(V + 35)/(10)));
alpha_n = (0.01 * (V + 55))/(1 - exp(-(V + 55)/(10)));
beta_n = 0.125 * exp(-(V + 65)/(80));
// I_stim is 1V during the specified time range (t_on -- t_off),
// 0V otherwise
I_stim = (t > t_on) ?  (t < t_off) ?  (1.0 * I_stim_mag):  0.0:  0.0;

I_Na = g_Na * cube(m) * h * (V - E_Na);
I_K = g_K * sqr(sqr (n)) * (V - E_K);
I_L = g_L * (V - E_L);

/* Integration of the four state variables.  */
d(V) = - (I_Na + I_K + I_L - I_stim) / C_m;
d(m) = alpha_m * (1 - m) - beta_m * m;
d(h) = alpha_h * (1 - h) - beta_h * h;
d(n) = alpha_n * (1 - n) - beta_n * n;

Vout1 = V;
```

.4 Information for Developers

.4.1 RTXI Architecture

RTXI is designed to run experiments that require high-frequency periodic execution. At the heart of this design is the real-time (RT) thread. The RT thread is essentially a standard Linux thread, with two important caveats: (i) it runs with the highest priority afforded by the real-time enabled kernel and (ii) it executes periodically and then sleeps for a designated (short) period of time.

In addition to the RT thread, RTXI runs a user-interface (UI) non-real-time thread. The UI thread is also a standard Linux thread and runs in the same process address space as the RT thread. The UI thread is responsible for handling user input in the form of command-line arguments and graphical user-interface (GUI) events. Because the UI and RT threads share an address space, they can interact with each other through data structures that are stored in that shared address space. It is through the manipulation of these data structures that the UI thread is able to act as a mediator between the user interacting with the GUI and the RT thread, which repeatedly wakes up and executes the user-selected modules loaded into RTXI.

Modules are function-specific code that can be used in combinations to build custom experiment protocols and interfaces, thereby eliminating the need to code all aspects of each experiment protocol from scratch. Often, users will have multiple modules working in parallel during a single RTXI session. Typically, those modules will need to share data and information. RTXI provides an event delivery system that allows modules to signal the occurrence of user-defined events (such as detected neuronal spikes) and then send data to other modules that are listening for such an event. All core system features in RTXI are actually written as modules and they are initially loaded according to a configuration file, `rtxi.conf`. This bootstraps RTXI into a state where users can perform basic tasks such as configuring system settings and the DAQ card, acquire and save experimental data, and load additional custom user modules.

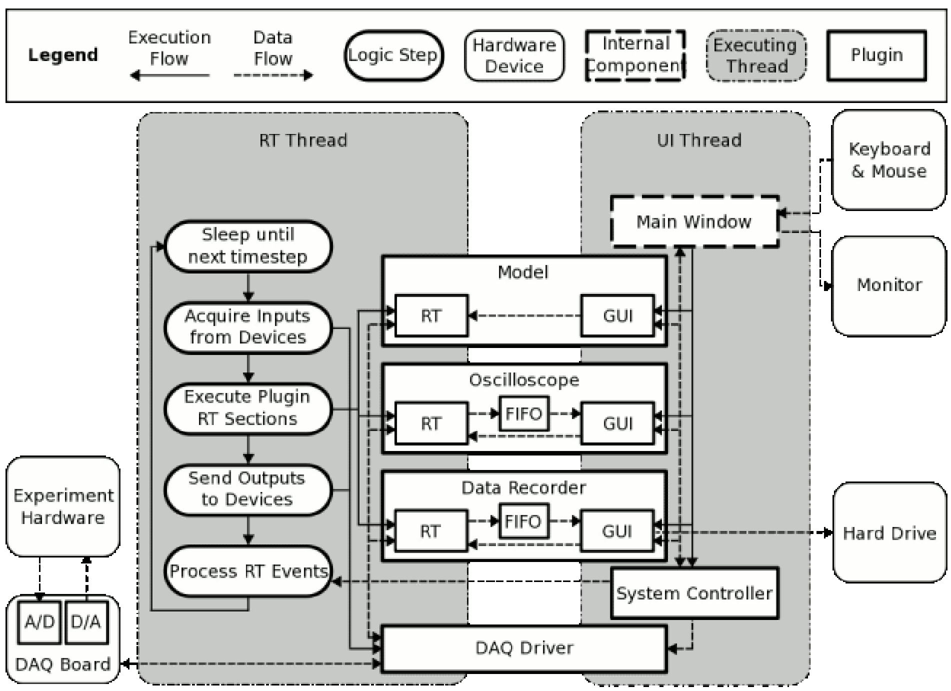

Figure 9: RTXI has a two-thread architecture. On every cycle, the real-time thread wakes and performs standard tasks such as sampling inputs to the DAQ card and outputting any signals. It also executes the real-time code from any loaded modules, which are dynamically linked to RTXI. The real-time thread then goes to sleep until the next cycle begins. All modules span the real-time and user interface threads.

Core system modules are not derived from the `DefaultGUIModel` class, however, and there are several different ways of implementing functionality at that level.

.4.2 Software Requirements

RTXI is a combination of several open source software initiatives:

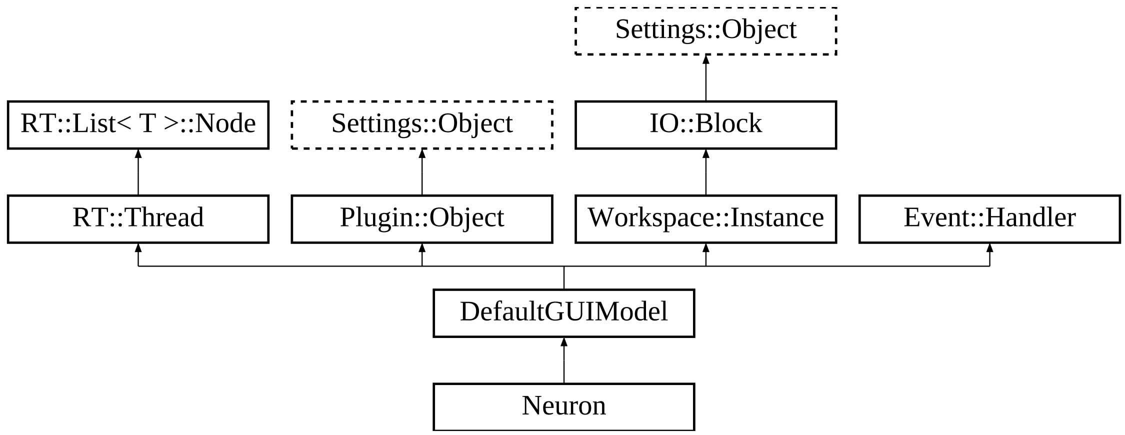

Figure 10: The `Neuron` module is a class derived from `DefaultGUIModel`, which inherits features such as hard real-time execution and event handling, the ability to generate and accept signals, and the ability to have metadata automatically captured by the Data Recorder to HDF5 data files.

1. Linux,
2. the Real Time Application Interface for Linux (RTAI),
3. COMEDI,
4. the Qt user interface framework, and
5. HDF5.

Linux: Linux is a generic term referring to Unix-like computer operating systems based on the Linux kernel. Their development is one of the most prominent examples of free and open source software collaboration; typically all the underlying source code can be used, freely modified, and redistributed, both commercially and non-commercially, by anyone under licenses such as the GNU General Public License. Desktop use of Linux has become increasingly user-friendly and popular in recent years. Typically, Linux is packaged into different distributions that include the Linux kernel and all of the supporting software required to run a complete system. These distributions may include modified versions of ”vanilla” Linux source code and common applications, such as the vim text editor. The RTXI Live CD and manual installation notes are based on the Ubuntu desktop distribution, which is popular with new Linux users. It features a complete desktop environment with common productivity software and GUI applications for system administration.

RTAI: RTAI (http://www.rtai.org) provides real-time extensions to the official Linux kernel to make hard real-time applications possible. This is achieved by patching the kernel and introducing additional modules to handle task scheduling, capture system interrupts, etc. This requires that the kernel be recompiled and manual installation instructions are provided here. RTAI also provides several benchmark tests for evaluating your system's real-time performance.

RTAI is not the only option for installing a real-time Linux kernel but it is the method used by the RTXI Live CD and described in the manual installation notes. RTXI also supports the Xenomai interface.

→ Chapter ??
COMEDI Support

COMEDI: The COMEDI project develops open-source drivers, tools, and libraries for data acquisition on Linux platforms. COMEDI supports a variety of common data acquisition module boards. Most RTXI users use DAQ cards by National Instruments, but any DAQ card that is supported by COMEDI should work with RTXI. RTXI can also handle multiple DAQ cards with a simple modification to the RTXI configuration file. A list of compatible DAQ cards is available in section 2.1.2.

→ Chapter 4.3
Custom GUI Modules

Qt: Qt is a cross-platform user-interface framework distributed by Nokia and used in RTXI under the LGPL license. This framework provides classes for developing sophisticated GUIs using a signals-and-slots mechanism similar to that of RTXI modules. RTXI uses Qt v3.3.8.

→ Chapter 3.8
Saving Data &
HDF5 Files

HDF5: HDF5 is a versatile data model that can represent complex data objects and a wide variety of metadata and allows you to quickly extract subsets of data. It is incorporated into RTXI through the Data Recorder module, which streams data to a HDF5 file along with the parameters of all modules connected to the Data Recorder. You can store multi-channel experimental recordings, instrument metadata, and browse images in a single file, making it possible to capture the entire collection of information about a single experiment. The format is completely portable and several tools are available for interacting with data in this format. HDFView is a free visual tool for browsing and editing HDF5 data structures and is available for Windows, Mac and Linux. MATLAB also has native functions for working with HDF5 files and we provide a tool that optimizes HDF5 files produced by RTXI for importing into MATLAB. We have developed a standardized hierarchical structure that will allow you to write MATLAB scripts that are compatible with all RTXI-generated HDF5 files.

RTXI depends on several additional Linux libraries that are typically available through the software repositories for each popular distribution of Linux:

1. GNU Scientific Library (GSL)
2. Boost libraries
3. Qt 3.3.8
4. HDF5

.4.3 Development Roadmap

Bibliography

[1] Bettencourt, J., Lillis, K., Stupin, L., and White, J. A., "Effects of imperfect dynamic clamp: Computational and experimental results," *J Neurosci Methods*, vol. 169, pp. 282–289, Oct 2008. Journal article.

[2] Brizzi, L., Meunier, C., Zytnicki, D., Donnet, M., Hansel, D., D'Incamps, B., and Vreeswijk, C. V., "How shunting inhibition affects the discharge of lumbar motoneurones: a dynamic clamp study in anaesthetized cats," *J Physiol*, vol. 558, pp. 671–83, Jul 2004. 0022-3751 (Print) Journal Article.

[3] Butera, R. J. and McCarthy, M., "Analysis of real-time numerical integration methods applied to dynamic clamp experiments," *J Neural Eng*, vol. 1, pp. 187–94, Dec 2004.

[4] Butera, R. J., Wilson, C. J., Negro, C. A. D., and Smith, J. C., "A methodology for achieving high-speed rates for artificial conductance injection in electrically excitable biological cells," *IEEE Trans Biomed Eng*, vol. 48, pp. 1460–70, Dec 2001. 0018-9294 (Print) Journal Article.

[5] Chance, F., Abbott, L. F., and Reyes, A., "Gain modulation from background synaptic input," *Neuron*, vol. 35, no. 4, pp. 773–782, 2002.

[6] COMEDI, *COMEDI: Linux Control and Measurement Device Interface*, Updated 2008, Accessed December 3, 2010, Available: http://www.comedi.org/.

[7] Culianu, C. A. and Christini, D. J., "Real-time experiment interface system: Rtlab," (Milan, Italy), 3rd Real-Time Linux Workshop, November 26-29 2001.

[8] Culianu, C. A. and Christini, D. J., "A sample data acquisition and control application using rtlab," (Boston, MA), 4th Real-Time Linux Workshop, December 6-7, 2002 2002.

[9] Destexhe, A., Rudolph-Lilith, M., Fellous, J., and Sejnowski, T. J., "Fluctuating synaptic conductances recreate in vivo-like activity in neocortical neurons," *Neuroscience*, vol. 107, no. 1, pp. 13–24, 2001. 0306-4522 (Print) Journal Article.

[10] Destexhe, A., Rudolph-Lilith, M., and Pare, D., "The high-conductance state of neocortical neurons in vivo," *Nat Rev Neurosci*, vol. 4, no. 9, pp. 739–751, 2003.

[11] Dorval, A. D., Christini, D. J., and White, J. A., "Real-time linux dynamic clamp: a fast and flexible way to construct virtual ion channels in living cells," *Ann Biomed Eng*, vol. 29, pp. 897–907, Oct 2001. 0090-6964 (Print) Journal Article.

[12] Economo, M. N., Fernandez, F. R., and White, J. A., "Dynamic clamp: alteration of response properties and creation of virtual realities in neurophysiology," *J Neurosci*, vol. 30, pp. 2407–13, Feb 2010.

[13] Goaillard, J.-M. and Marder, E., "Dynamic clamp analyses of cardiac, endocrine, and neural function," *Physiology (Bethesda)*, vol. 21, pp. 197–207, Jun 2006. 1548-9213 (Print) Journal Article.

[14] Golowasch, J., Goldman, M. S., Abbott, L. F., and Marder, E., "Failure of averaging in the construction of a conductance-based neuron model," *J Neurophysiol*, vol. 87, pp. 1129–31, Feb 2002.

[15] Hughes, S., Lorincz, M., Cope, D., and Crunelli, V., "Neureal: An interactive simulation system for implementing artificial dendrites and large hybrid networks," *J Neurosci Methods*, vol. 169, pp. 290–301, Nov 2008. Journal article.

[16] Jaeger, D. and Bower, J., "Synaptic control of spiking in cerebellar purkinje cells: Dynamic current clamp based on model conductances," *J Neurosci*, vol. 19, no. 14, pp. 6090–6101, 1999.

[17] Kumar, A., Schrader, S., Aertsen, A., and Rotter, S., "The high-conductance state of cortical networks," *Neural Comput*, vol. 20, pp. 1–43, Jan 2008.

[18] Milescu, L. S., Yamanishi, T., Ptak, K., Mogri, M. Z., and Smith, J. C., "Real-time kinetic modeling of voltage-gated ion channels using dynamic clamp," *Biophys J*, vol. 95, pp. 66–87, Jul 2008.

[19] Politecnico di Milano - Dipartimento di Ingegneria Aerospaziale, *RTAI - the RealTime Application Interface for Linux*, Updated February 2010, Accessed December 3, 2010, Available: http://www.rtai.org.

[20] Preyer, A. J. and Butera, R. J., "The effect of residual electrode resistance and sampling delay on transient instability in the dynamic clamp system," *Conf Proc IEEE Eng Med Biol Soc*, vol. 2007, pp. 430–3, 2007.

[21] Preyer, A. J. and Butera, R. J., "Causes of transient instabilities in the dynamic clamp," *IEEE Trans Neural Syst Rehabil Eng*, vol. 17, pp. 190–8, Apr 2009.

[22] Prinz, A. A., Abbott, L. F., and Marder, E., "The dynamic clamp comes of age," *Trends Neurosci*, vol. 27, pp. 218–24, Apr 2004. 0166-2236 (Print) Journal Article Review.

[23] Raikov, I., Preyer, A. J., and Butera, R. J., "Mrci: a flexible real-time dynamic clamp system for electrophysiology experiments," *J Neurosci Methods*, vol. 132, pp. 109–23, Jan 2004. 0165-0270 (Print) Journal Article.

[24] Robinson, H. and Kawai, N., "Injection of digitally synthesized synaptic conductance transients to measure the integrative properties of neurons," *J Neurosci Methods*, vol. 49, pp. 157–65, Sep 1993.

[25] Rudolph, M. and Destexhe, A., "A fast-conducting, stochastic integrative mode for neocortical neurons in vivo," *J Neurosci*, vol. 23, pp. 2466–76, Mar 2003.

[26] Rudolph, M. and Destexhe, A., "Tuning neocortical pyramidal neurons between integrators and coincidence detectors," *J Comput Neurosci*, vol. 14, pp. 239–51, Jan 2003.

[27] Salinas, E. and Sejnowski, T. J., "Impact of correlated synaptic input on output firing rate and variability in simple neuronal models," *J Neurosci*, vol. 20, pp. 6193–209, Aug 2000.

[28] Sceniak, M. P. and Sabo, S. L., "Modulation of firing rate by background synaptic noise statistics in rat visual cortical neurons," *J Neurophysiol*, vol. 104, pp. 2792–2805, Aug 2010.

[29] Sharp, A., O'Neil, M., Abbott, L. F., and Marder, E., "Dynamic clamp: computer-generated conductances in real neurons," *J Neurophysiol*, vol. 69, pp. 992–5, Mar 1993. 0022-3077 (Print) Journal Article.

[30] Steriade, M., "Impact of network activities on neuronal properties in corticothalamic systems," *J Neurophysiol*, vol. 86, pp. 1–39, Jul 2001.

[31] Tiesinga, P., Jose, J., and Sejnowski, T. J., "Comparison of current-driven and conductance-driven neocortical model neurons with hodgkin-huxley voltage-gated channels," *Phys Rev E*, vol. 62, no. 6, pp. 8413–8419, 2000.

[32] Wilders, R., "Dynamic clamp: a powerful tool in cardiac electrophysiology," *J Physiol*, vol. 576, pp. 349–359, Jul 2006. 0022-3751 (Print) Journal article.

[33] Wolfart, J., Debay, D., Masson, G. L., Destexhe, A., and Bal, T., "Synaptic background activity controls spike transfer from thalamus to cortex," *Nat Neurosci*, vol. 8, no. 12, pp. 1760–1767, 2005.