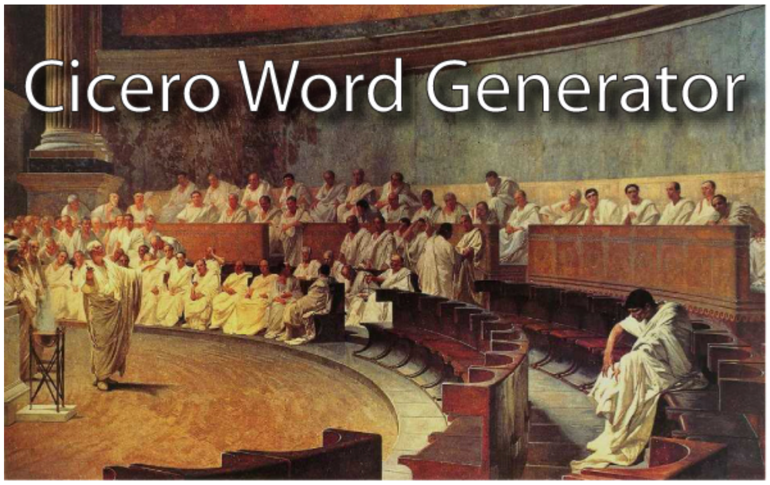


Cicero Word Generator

Technical and User Manual

Aviv Keshet, June 7, 2008

Introduction

This document describes the Cicero Word Generator project, a collection of software applications for controlling atomic physics experiments. The project was undertaken at the MIT Center for Ultracold Atoms, beginning in 2007.

The purposes of this document are:

- To describe the architecture of the software, and the fundamentals of how it operates.
- To instruct users in configuring and operating the software.
- To provide details on the inner workings of the software, of interest to those who would add features or modify the software.

When text appears in `Courier`, this indicates that the text is the name of an object or function as it appears in the software source code. When text appears in **Boldface**, this indicates that the text refers to a field or text label in the software's user interface, or to a literal string such as a file name.

License and Warranty Information

The following copyright information applies to all the applications in the Cicero Word Generator suite:

```
Copyright (C) 2008, Aviv Keshet

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.

For the full text of the of the GNU General Public License, see
<http://www.gnu.org/licenses/>.
```

This Cicero Word Generator suite makes use of the .Math library to parse and evaluate user-entered equations. The following copyright notice applies to the .Math library:

```
Copyright (c) 2001-2004, Stephen Hebert
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:
```

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the .Math, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Hardware and software support and requirements

- Cicero is designed primarily to drive National Instruments output hardware. It should be compatible with any output hardware which uses the NIDaqMx driver library. It has been tested with the following output cards:
    - Analog: PCI-6713, PXI-6713
    - Digital: PXI-6533, PXI-6434, PCI-6533
- The NIDaqMx library does NOT RUN on Windows Vista. Thus, the hardware layer of Cicero (ie Atticus, as explained below) requires Windows XP or earlier, with XP being the recommended choice. In principle, the UI layer of Cicero may run on Vista, but this is not tested. Atticus is known to run on Windows 2000, and Cicero is known not to run on Windows 2000.
- The hardware layer can consume substantial amounts of memory for generation of output buffers. It is thus recommended to have 4GB of memory on computers with output cards.
- For the software to run, the .NET Framework 2.0 must be installed. This is not installed by default with new Windows XP installations, but can be installed via Windows Update.
- It is recommended to install the National Instruments Measurement Studio package. This contains a full suite of drivers for all National Instruments devices, as well as the National Instruments Measurement & Automation

Explorer (aka MAX) which is useful in getting information about installed devices.

Installation Instructions

Once the National Instruments drivers and .NET Framework are installed, all of the applications in this collection are standalone programs, that do not require running an install program. The software is generally distributed as raw source code + compiled binaries. To install, for example, the Cicero client, go to [BaseDirectory]\Cicero\WordGenerator\bin\Release\ and copy all the contents of this directory to a new directory on your computer where you would like to store the application. The hardware server program installation procedure is similar, residing in Cicero\AtticusServer\...

To update your installation to reflect a new version of the software, simply copy over the contents of the new version's directory over the contents of the directory on your computer. However, be sure not to completely erase your old directory, since this directory contains the configuration settings files that you have made when configuring the software. Only overwrite the contents of the directory, not the directory itself.

What is Cicero?

Cicero was built as a replacement for “Word Generator II” (or WG2), written by Chris Kuklewicz around 1999. WG2 is capable of designing a sequence to be run on a fixed number of National Instruments digital and analog output cards. The user interface, fairly intuitively, allows the user to set the digital output values at discrete time slices of settable duration (called Words in that program). Analog and GPIB-controlled microwave ramps can be started at the beginning of these Words (by starting a Group).

While this powerful interface allowed for fairly straightforward design of very complex control sequences, there were limitations to WG2. The principle limitations were:

- Unmaintained code, difficult or impossible to upgrade with new features.
- Fixed hard-coded hardware configuration, limiting the number and type of channels that could be used.
- Incompatibility with newer National Instruments output hardware.
- No support for “batch mode” runs, in which parameters are scanned through a number of values.

The groundwork for Cicero was laid by Widagdo Setiawan, with the XGenerator project. Unfortunately, XGenerator was not finished by the time of Widagdo’s graduation, and it was determined that a re-write would be more efficient than a continuation of the project. Much of the architecture of Cicero was inspired by the architecture of XGenerator, and Cicero would certainly have been far inferior if not for the lessons of XGenerator.

The user interface for sequence design in Cicero is based on the same concepts as that of WG2. A number of UI features have been added, most prominently the ability to make any numerical parameter of the sequence into a scannable variable. Further changes will be elaborated upon below.

How it Works

There are two main applications which are used in designing and running experiment control sequences:

- Cicero: the user interface (“client”) for editing sequences.
- Atticus: the back-end (“server”) software which translates the sequences to output buffers, and passes these output buffers to the output hardware.

In addition, an application named Elgin allows for browsing the run logs generated by Cicero every time a sequence is run, and assists in data analysis.

Cicero and Atticus communicate with each other via .NET Remoting, a software library which seamlessly allows applications running on separate computers to share data. In fact, one Cicero application can simultaneously communicate with several Atticus servers, allowing multiple computers to have their output hardware controlled by one user interface. The rationale behind this client/server model is precisely that it allows for multiple output hardware computers. This enables the control system to scale to (in principle) arbitrarily high numbers of outputs without being limited by the resources of a single computer.

It is instructive to expand further on the roles of the two pieces of software. Cicero is in essence an editor, which edits two types of objects:

- `SettingsData`: this object stores all of the data which, while it should be editable from within the user interface, will not vary from sequence to sequence. Examples are the names and descriptions of all of the channels, the locations of the servers to connect to, and the mappings from channels (`LogicalChannel`) to their hardware address (`HardwareChannel`).
- `SequenceData`: this objects stores all of the sequence-specific data not stored in the settings object described above. This includes the values of all of the digital channels during each of the `TimeStep`s, all of the `AnalogGroup`s used in the sequence and the `Waveform`s that each of these uses, the names and values of all of the `Variable`s used in the sequence, etc.

While it should not be necessary to do so under normal operation, within Cicero it is possible to directly browse and edit these objects hierarchically. This can be accomplished by selecting the **Advanced -> Sequence Explorer** or **Advanced -> Settings Explorer** menu items within Cicero. Of course, using the normal user interface is a much more natural way to accomplish most sequence and settings editing.

At any time, it is possible to load or save a sequence or settings object. This functionality is accessed through **File** menu. In addition, this menu can be used to save a sequence or settings object as the default startup object, which will cause it to be automatically loaded whenever Cicero is started. This is a natural course of action, especially for the settings object which is unlikely to require frequent changes.

The other main role of Cicero is to communicate with the various Atticus servers in order to execute a run, or to fetch a collection of available `HardwareChannel`s.

The user interface for Atticus is fairly bare by comparison, allowing the user to configure all of the hardware parameters (to be explained in much greater detail below). These settings are all stored in a `ServerSettings` object which is saved to the file **AtticusServerSettings.set** in the same directory as the Atticus executable. Atticus detects which `HardwareChannel`s are available on the computer it is running on, and allows the user to browse them and exclude specific channels if necessary (circumstances where this is recommended will be described later).

The main role of Atticus is to listen for messages from Cicero, and to turn sequence and settings data into output buffers and put these output buffers to use. To initiate each run of the experiment, Cicero sends the following series of messages to Atticus:

1. A `SettingsData` object for the current run.
2. A `SequenceData` object for the current run.
3. A `generateBuffers` command, which causes Atticus to turn the small sequence data object into a full-size output buffer for all of the channels that reside on the given server. Depending on the size of the buffer, this step can be time consuming, on the order of several seconds.
4. An `armTasks` command, which instructs Atticus to prepare any hardware-triggered or externally-clocked buffers to be triggered. (the details of triggering and clocking will be explained below).
5. A `generateTrigger` command, which actually begins the run.

Note that if multiple Atticus servers are attached to a single Cicero client, then each server will receive each message at nearly the same time, and none of the servers will receive the next message until each of the servers has finished handling the previous one. This ensures the possibility of synchronizing outputs from multiple computers.

At the end of the run, Atticus receives a `runSuccess` query, so that Cicero can determine whether the run executed successfully or not.

Tasks

This section will give an overview of the types of outputs supported by Atticus, and the relevant features of these outputs to keep in mind when configuring the system.

The `SequenceData` object that is edited in Cicero and passed over the network to Atticus is a high-level description of the sequence to be run. The sequence is described in terms of durations of words, values of ramps, and other user-editable values. However, the data format for the output hardware is a low-level description of the sequence, just a buffer or an array of values to be output at particular times. Such a buffer, along with the hardware-specific configuration settings for that buffer, is called a `Task`.

There are two classes of `Task` – Software Timed and Hardware Timed. To run a software timed task, a computer's CPU runs a thread which polls the CPU's clock at regular intervals. Whenever the CPU determines that it is time to output the next command or data point, it does so. However, this form of timing is *nondeterministic*. In other words, the exact time when output will occur cannot be controlled to better than a few milliseconds, because of variations in CPU load and CPU clock inaccuracies. Thus, Software Timed tasks are not suitable for use on outputs which require fast and tight synchronization with other outputs.

Hardware Timed tasks, on the other hand, are deterministic. A buffer of outputs or commands is loaded to the onboard memory of an output card. The card also has a `SampleClock` signal input. Every time an edge is received on the `SampleClock` signal, the card immediately (~ns) outputs the next value of its buffer. By sharing a sample clock between several output cards, it is thus possible to synchronize them *deterministically.*

In practice, all of the analog and digital outputs from Atticus are provided by National Instruments output cards, which support sophisticated hardware timing. On the other hand, serial and GPIB devices are all run with software timing.

Hardware Timing features of NI output cards

National Instruments cards offer a fair bit of flexibility in configuring their timing (more flexibility than we need). This section will discuss the basics of the timing features supported by Cicero, and their potential uses.

As mentioned above, NI cards make use of an output buffer and a sample clock to generate their outputs. Each time an edge is received on the sample clock, the card goes to the next sample in its buffer. Cards are capable of synthesizing their own sample clock at a wide range of fixed frequencies, by dividing down from an onboard master clock (usually running at 10MHz).

NI cards on a single computer share a bus for communicating with other cards, and sharing timing signals. With PCI cards, this is knows as the rtsi bus (channels rtsi0 to rtsi7). With PXI cards, it is the PXI_Trig bus (PXI_Trig0 to PXI_Trig7). Most of the timing signals that a card uses can either be routed to, or routed from, one of the channels of this shared bus. In addition, each card will have some number of PFI channels which allow routing of external signals (through the same cable that connects your card to the outside world) into or out of this timing bus.

There are a number of ways to synchronize several cards. Cards on the same PXI or rtsi bus can share a sample clock over the bus. Or, they can share a master clock and a start trigger signal, and each generate their sample clock independently (the advantage of this method being that it allows you to run different cards at different clock rates). If cards are not on the same bus (for instance, cards in separate computers), then a card on the “sending” end can route its sample clock to a PFI channel, this can be wired by a coax cable to a PFI channel on the “receiving” card, and the receiving end PFI channel can be routed to the channel’s sample clock.

Variable Timebase clocks

The methods discussed above all involve synchronizing a set of output cards through the use of one or a few fixed frequency clocks. However, this method has one major disadvantage. The disadvantage comes from the wide separation of timescales involved in a typical atomic physics experimental sequence.

In a typical BEC experimental sequence, for instance, you may want to have several seconds of MOT loading time or evaporation time during which none of the digital or analog channels have changing values. But then, you also want an imaging pulse during which you change a number of outputs in say 50us. Thus, if running your experiment with a fixed frequency clock synchronizing all of the output cards, you have no choice but to clock all the cards at 50us throughout the whole sequence, even during the long stretches of time when nothing is changing. As a result, you must generate long redundant buffers for these long operations, just so that your short operations will be fast enough. This increases the buffer generation time, and decreases the best time resolution you can achieve. If you were to attempt to go to ~1us time resolution, these requirements quickly become too much to bare.

An alternative method is to synchronize all the cards with a variable frequency clock (known throughout this program as a variable timebase), which only generates clock pulses when the card outputs actually need to change. This provides a great saving in buffer memory usage, generation time, and a large improvement in achievable time resolution. Variable Timebase features are implemented in Atticus. Configuration details will be discussed below, but the concepts will be explained here.

To use this feature, one digital output is specified as the variable timebase output channel. This channel is routed to the PFI channels of at least one card on each computer (or each RTSI / PXI bus) in use. The cards are configured to use this variable timebase as their sample clock.

Before the server generates buffers for the output channels, it calculates the behavior of the variable frequency clock, based on when output channel need to be changing. This list of `VariableTimebaseSegment`s is then used when calculating the output buffers, and is used to synthesize a variable timebase clock which is output on the specified variable timebase output channel.

It is worthwhile to note that whenever the value of *any* channel changes, there is a variable timebase clock pulse for *all* of the cards. If analog groups are running, then while they are running the clock rate comes from the minimum time resolution of the running analog groups. Thus, when using a variable timebase it is somewhat important to keep the underlying timing mechanism in mind when designing an experimental sequence. When not necessary, do not create extremely long-duration analog groups with a high clock frequency, as this effectively invalidates the advantages of using a variable timebase.

Finally, there are some hardware considerations when using a variable timebase. The synthesized variable timebase clock is itself a Task, which must be clocked with a frequency that is twice as high as the highest variable timebase output frequency. While saving on generating large buffers for all the other channels, using a variable timebase does require one large buffer to be generated (the buffer for the synthesized clock). The NI 6533 digital output card has a very small on-board buffer of just 16 samples (relying on frequent refreshes of the on-board buffer from system memory). Thus, this card is unsuitable for generating the variable timebase clock. The NI 6534 on the other hand has a sizable on-board buffer, and is quite suitable. For best performance, the variable timebase output should be the first output channel on the card, and the next 15 outputs should not be used. This means sacrificing half of the channels of the card. The reason behind this is that each half of the card can be clocked independently, but all the channels within one half must NOT be clocked independently. If you want to use the other 15 channels on the same half of the card as the variable timebase output, then the increased memory required for this limits somewhat the maximum length buffers you can generate.

This variable timebase scheme has been tested as able to run ~60s sequences with 1us time resolution, on a computer with 4GB of memory. Longer sequences may require a reduction in time resolution in order not to run out of memory.

Configuring Atticus

This chapter will give a step-by-step guide to configuring a new installation of Atticus.

1. Start Atticus.
2. Select a **Server Name**, and enter it into the appropriate text box. The server name will be incorporated into the names of the `HardwareChannel`s exported by the server.
3. Familiarize yourself with the installed output hardware. This can be done by running the National Instruments software "Measurement and Automation Explorer" or MAX. This software is available as part of the National Instruments Measurement Studio software package, or with the drivers of your output hardware.
4. Discover your hardware-timed channels.
    a. In the **Hardware Settings** section of the Atticus window, you should see a box containing a list of the detected output devices (**Dev1, Dev2**, etc.). Note: It is possible within MAX to rename these devices. However, renaming them is not recommended; Atticus relies on the NI output hardware to have names that begin with Dev.
    b. Select a device to see its configuration settings appear in the property editor box to the right of the device list. If no devices appear in the list, press the **Refresh Hardware** button.
    c. Under the heading **Global**, examine the value under **Device Description**. For National Instruments output cards, this will give you the part number of the output card, so that you can identify which card corresponds to which device number. (This can also be done with MAX, as described above).
    d. Set **DeviceEnabled** to true, if you wish to use this card.
    e. If this card has analog channels that you wish to use, set **AnalogChannelsEnabled** to **True** under the **Analog** heading. Similarly for digital channels. Most cards tested only support either analog OR digital output, not both simultaneously. If you attempt to use both types of channels and this is not supported by the card, you will receive an exception to this effect if you try to run a sequence that uses both types of channels on this card.
    f. Detect the card's channels, by pressing the **Refresh Hardware** button. The list of channels towards the bottom of the **Hardware Settings** section should now contain the channels available on this card.
    g. Some cards have channels which are not actually useful. For instance, the digital output card PXI-6753 has 4 8-bit ports of channels which support hardware timing, but also an additional 4-

bit port of channels which only support software timing, and are thus basically useless for our applications. Thus, to ensure that these channels do not get used by accident, you can select these useless channels from the list, and click the **Exclude Channel(s)** button to remove them from the list of exported channels. To recover these channels in the future, if necessary, you can edit the **Excluded Channels** list under the **Hardware** heading of the **Server Settings** property box.

5. Configure hardware timing / synchronization
    a. As discussed above, the easiest way to synchronize various tasks is to have them share a sample clock. Each NI card can either produce a sample clock by dividing down a higher frequency "master clock", or get its sample clock from another source. Multiple cards with different sample clock rates can even be synchronized, by having each card divide down its own sample clock, but all from a shared "master clock". This document will explain how to set up timing and synchronization for either a shared sample clock, or for a shared variable timebase.
    b. **Shared Sample Clock**.
        1. Chose one of the cards on one of the computers to be the master card (say **Dev1**). This card will be the source of the sample clock. We will use a 50000 Hz (50 kHz) clock rate in this example.
        2. Edit the device settings for **Dev1** to reflect the following:

Setting	Value
MasterTimebaseSource	OnBoardClock
MySampleClockSource	DerivedFromMaster
SampleClockRate	50000
UsingVariabletimebase	False
SoftTriggerLast	True
StartTriggerType	SoftwareTrigger

        3. Route the sample clock signal to the timing bus. If you have Measurement & Automation Explorer installed (aka MAX, by National Instruments) you can browse the possible connections available for this device. For instance, with a PXI 6713 analog output card as Dev1, we would connect /Dev1/ao/SampleClock to /Dev1/ao/PXI_Trig0 . For PCI cards, or for other types of cards, consult MAX to see the corresponding sample clock source port, and available rtsi timing ports. To route the signal, edit the **Connections** settings in the **Server Settings** property browser. Click **Add** to add a connection. Set the **SourceTerminal** to **/Dev1/ao/SampleClock** and the **DestinationTerminal** to **/Dev1/PXI_Trig0** . Now the sample clock timing signal is being exported from this device to shared timing port PXI_Trig0.

4. If you are using additional computers, or additional cards which are not on the same timing bus as this master card, then route /Dev1/ao/SampleClock to /Dev1/PFI0 , run a wire or coaxial cable from this card's PFI0 channel to the PFI0 channel on a card on the other computer, and on the destination computer route /DevX/PFI0 to /DevX/PXI_Trig0 (where X is the device number of the card on the destination computer, and assuming that the destination card is using a PXI bus. If rtsi, make the appropriate substitutions). *Note: if the length of this cable running the timing signal is greater than a few feet, it may be advisable to add a 50 ohm terminator to the receiving end of the timing signal. There is a known instance of a long cable leading to reflections / distortion of the timing signal, causing intermittent timing glitches.*
5. For each of the remaining cards, use the following device settings:

Setting	Value
MasterTimebaseSource	
MySampleClockSource	External
SampleClockExternalSource	PXI_Trig0 **or** rtsi0 **as appropriate**
SampleClockRate	50000
UsingVariabletimebase	False
SoftTriggerLast	False
StartTriggerType	SoftwareTrigger

c. **Variable Timebase**

1. Read the above instructions for the basics of routing signals.
2. The variable timebase output channel should be a channel on the large-buffer NI digital output card (6534). For convenience, we use the first channel of this card, and assume that this card is Dev1 on a PXI timing bus. We will use a fundamental clock speed of 1MHz. This means that sequences running with this variable timebase will have 1us time resolution (but with a minimum time between clock edges of 2us).
3. Under **Server Settings**, set **VariableTimebaseOutputChannel** to **/Dev1/port0/line0** . This specifies that the first channel of Dev1 will be used as the variable timebase output.
4. Set **VariableTimebaseMasterFrequency** to 1000000.
5. Exclude /Dev1/port0/line0 from the list of exported channels (see section on discovering channels for

instructions). For best performance, consider excluding all of the channels on port0 and port1.

6. Run a wire or coaxial cable from the /Dev1/port0/line0 output to the PFI0 input of one card on each of the computers / timing busses in use. On these “receiving cards”, route their PFI0 channel to rtsi0 or PXI_Trig0 as appropriate.
7. For **each device** including the one generating the variable timebase clock, set the following device settings:

Setting	Value
MasterTimebaseSource	
MySampleClockSource	External
SampleClockExternalSource	PXI_Trig0 **or** rtsi0 **as appropriate**
SampleClockRate	1000000
UsingVariabletimebase	True
SoftTriggerLast	False
StartTriggerType	SoftwareTrigger

6. Tweaking buffer performance
    a. Most of the details of the communication between the computer and the output cards and handled by the NI drivers. This includes the re-filling of the cards’ onboard buffers when necessary. Usually this works fine, however some of the cards are prone to buffer underruns if clocked at above ~50kHz (the card runs out of data in its on board buffer, and doesn’t get it re-filled before the next sample clock). These underruns are reported as such by Atticus. Some of the cards (in my experience, the Analog output cards) support giving the drivers some hints or tweaks about what data transfer mechanism to use, and with these tweaks applied the cards can perform at higher rates. Other cards don’t support these tweaks, and the drivers will report an error if you attempt to apply them.
    b. The tweaks are applied by editing the device settings for the card you want to tweak. In my experience, the best tweaks are as follows. For each analog output card, set **UseCustomAnalogTransferSettings** to **True**, **AnalogDataTransferMechanism** to **Interrupts**, and **AnalogDataTransferCondition** to **OnBoardMemoryHalfFullOrLess**. This effectively makes the analog cards a little more aggressive in refilling their buffers, reducing the likelihood of buffer underruns.
7. Configure Software Timed Devices
    a. GPIB Devices
        1. Whenever Atticus refreshes its hardware, it searches for connected GPIB devices on any of its GPIB ports. If a

device is detected, the channel number for the GPIB device appears in the Devices list. The device description is set to the string returned by the device in response to a "*IDN?\n" command, which is the standard query for a device to identify itself.

2. Most of the DeviceSettings parameters are meaningless for GPIB devices. The only relevant parameter is SampleClockRate. This parameter sets the rate at which commands are output during ramps (for instance, when ramping the voltage and frequency of a microwave synthesizer, for forced evaporation of a BEC). In my experience, Agilent microwave synthesizers begin to behave strangely (ignore a random subset of their received commands) if clocked beyond ~17 Hz.
3. If you wish to run voltage / frequency ramps on this GPIB device, then Atticus needs to know the commands for setting the voltage and frequency. This support is built-in for devices whose ID string contains the substring "ESG-4000B" or "N5181". For other devices, you need to tell Atticus how to formulate the commands. To do this, you must add a new **GpibRampCommandConverter**. These can be added to the GPIB subsection of the server settings object. Consult the manual for your gpib device, to determine the commands for setting the amplitude and frequency of its output. The commands output by Atticus are a string combining a prefix and postfix string to go before and after a numerical value (in Volts for amplitude, in Hz for frequency). For instance, to set a certain brand of synthesizer to 1 GHz, Atticus might use the command "FREQ 1000000000 Hz\r\n". In this case, the prefix is "FREQ ", the postfix is " Hz\r\n", and the number is 1000000000. The field **DeviceIdentifierSubstring** should contain a substring which is contained in the Device Description for the GPIB device, so that Atticus knows which device to apply this ramp converter to. Note: the patterns "\r" or "\n" in the prefix or postfix string are converted by Atticus into the special characters that they conventionally represent (Carriage Return and Line Feed respectively). Nearly all devices require commands to be terminated by one or both of these characters, so the pattern "\r\n" should be included at the end of the postfix string. Some devices are pickier, and want one special character but not the other.
4. Device settings for GPIB devices will persist even if the device is disconnected or turned off. If, at a later time, a different GPIB device with the same GPIB address is

attached to this GPIB port, then the device settings for may be inappropriate for the device, and the device description string may not get refreshed. To resolve this issue, delete the device settings object for the disconnected device, and click refresh hardware to discover the newly connected device properly.

b. RS232 Devices
    1. At present, there are no special configuration parameters required for RS232 channels. Some devices are picky about the baud rate or other serial settings of the data they receive. There is a mechanism for setting these appropriately within Atticus, but a more convenient way is to edit the properties of the output port the usual windows way. MAX provides a good user interface for doing so.

c. Triggering / synchronization of software timed tasks
    1. As discussed above, software timed tasks do not have a deterministic timing mechanism. The default behavior is for the software timed tasks to be started when Atticus finishes processing the “generate triggers” command from Cicero (as discussed near the beginning of this document). This generally works fairly well, and results in the software timed tasks being started within a few ms of the hardware timed tasks. However, for deployments of the software which run on multiple computers, it is possible for the “generate triggers” command to take vastly different amounts of time for different computers to process (especially when one of the computers is running a variable timebase task – these tasks, because they consume so much memory, take longer to start, and make the generate triggers command take longer to handle). This can result in unacceptably long software timing offsets of up to a second.
    2. Thus, there exists a mechanism to more closely tie the starting of the software timed tasks to the starting of the hardware timed tasks (hardware timed tasks on multiple computers already have a natural synchronization mechanism, in that they will share a sample clock). This involves creating a thread on Atticus during its handling of the arm tasks command which constantly polls the current buffer position of one of the hardware timed tasks. As soon as the buffer position of this task moves, Atticus knows that the hardware timed tasks have started, and starts the software timed tasks. This results in software timed tasks that, even on multiple computers, can be synchronized to the hardware timed tasks on the order of a few ms.

3. To use this feature, go to server settings and set **SoftwareTaskTriggerMethod** to **PollBufferPosition**, and **DeviceToSyncSoftwareTimedTasksTo** to the string identifier for the output card whose task Atticus should poll for this synchronization (e.g. **Dev1**, **Dev2**, etc). Note that you must specify a card that actually has channels bound to it and that is being used in the sequence. Otherwise, there will not be a task corresponding to this id string to synchronize to.

Configuration and Basic Usage of Cicero

This chapter will give a step by step description of how to configure a new installation of Cicero, and use its basic sequence editing and running features. It is assumed that the reader has at least one configured Atticus server.

1. Start Cicero.
2. When started for the first time, Cicero may prompt you to select a ClientStartupSettings file. Click cancel on the file open dialog to instruct Cicero to create this object anew.
3. Connect to servers. (This step is necessary to use output hardware. However, if you just want to start Cicero to get a flavor of how the interface works, it can be skipped)
    a. Press F11, click on the **Server Manager** button, or **Server Manager** from the **Tools** menu to start the Server Manager.
    b. Open the Servers collection, by clicking on Servers, and then on the small … button.
    c. Add a new server with the Add button. Set **ServerEnabled** to True, and set the **ServerAddress** to the IP address of the computer running Atticus (or to localhost, if running the server on the same computer as the client).
    d. Repeat as necessary, if more servers are being used.
    e. Close the server collection editor, and click on **Connect To Enabled Servers** to attempt to connect. If connection is successful, **ConnectedServersCount** will reflect the number of servers that were successfully connected.
    f. If Atticus is running, but Cicero is unable to connect to it, it may be one of the following reasons:
        i. You did not click the **Connect** button on the server, and thus the server is not accepting connection requests.
        ii. The IP address for the server is entered incorrectly.
        iii. Either the Client or the Server is not connected to the network, or there is a firewall running on one of the computers interfering with their connection. (Cicero and Atticus communicate via TCP on port 5678, so the firewall problem can be alleviated by opening this port to TCP traffic).
    g. Whenever Cicero is started, you will need to start the Server Manager and click **Connect To Enabled Servers** if you intend to use the output hardware.
4. Create and bind channels.

   a. Start the **Channel Manager**, either with the labeled button or through the Tools menu.
   b. The **Channel Manager** allows you to easily add or edit channels of the various channel types supported by Cicero.
      i. Whenever a channel is created, it is assigned a Logical ID #.
      ii. Within each channel type, no two channels can have the same ID#.
      iii. The SequenceData objects that are later created with Cicero store their sequence information based on the ID# of the channel they apply to, so changes to the channel ID#s will cause changes in the way Sequences are output.
      iv. When editing a sequence, data for the various channels will be displayed sorted by the channel ID#.
   c. To bind a logical channel to a hardware channel, select the **Hardware Channel** drop down menu either when adding or editing a given channel. This will give a list of all the known and unused hardware channels. Select **Unassigned** to leave this channel un-bound. It is useful to have **Unassigned** channels to act as placeholders for future channels, or to temporarily stop using certain hardware. If the drop down list of hardware channels is empty, then either:
      i. No servers are connected.
      ii. The servers that are connected do not have any channels the selected channel type, or all the channels of that type are already bound.
   d. Channels can be given a name, as well as a description. In most of the user interface elements which display the channel name, holding the mouse over the channel name will cause a tool-tip to display the full channel description.
   e. When the channel manager is closed, labels for the newly created channels should appear in the appropriate channel label panels.
5. Save the server and channel configuration, by saving the Settings data. Select **Save Settings as Default** from the file menu to save these to the default settings file, or save them to a different file with the other menu options.
6. Start making your first Sequence.
   a. Right-click somewhere in the empty region to the right of the "Analog Group:" label, to make the TimeStep panel context menu appear. Add some new timesteps.
   b. The digital value boxes will appear gray. This indicates that these channels have no entry in the sequence data object. To give these channels their sequence data entries, click the **Populate Sequence** button. The digital boxes will turn beige, to indicate that they have the value "false". Clicking on the boxes toggles the digital value at that point. Clicking and dragging horizontally allows for toggling one channel's value at several different timesteps.

c. The analog preview panel will appear to have a red hatched background, with grey channel separator lines and flat white channel preview lines. Red indicates that the given timestep is disabled. Enable some timesteps to see this background turn green. Click on the preview panel to see a cursor indicating the value of a given channel at a certain time.
d. Make your analog channels do something more interesting. Go to the Analog tab of the main window, and rename the analog group “Unnamed” to something more descriptive by editing the text box next to the **Rename** button, and then clicking rename. Turn on a few channels in this analog group by clicking on the buttons labeled **Continue** to make them read **Enabled**. Waveform viewers will appear for the enabled channels. Click on a waveform to select it, and enable the waveform editor. Create a piecewise linear waveform with five data points (use the down arrow to add interpolation points), lasting 10 seconds (duration), and with some interesting shape. Select some other channel waveforms and explore some of the other interpolation types.
e. Go back to the Sequence tab. In one of your timesteps, click on the analog group drop-down list and select your newly edited analog group. The analog preview pane will update to reflect this change. This panel updates automatically if the checkbox labeled **Auto** (next to the **Update** button) is selected. Turn this off if auto-updating is causing an unacceptable performance lag. To force an update, click the update button.
f. Channels that have a waveform that has just started in a given timestep will have a black background in the preview panel. Channels that are continuing a waveform that was started in a previous timestep will have a green background. A given analog group only affects the channels that are **Enabled** in that group. The channels that are **Continue** in that group will not be affected by starting the group, but will continue doing what they were doing before.
g. Drag the boundary between the analog preview panel and the digital grid up and down, depending one which are you more interested at a given moment.
h. Create and add GPIB and RS232 Groups in a similar way to Analog Groups. RS232 and GPIB Groups both involve creating a set of commands to give to a device. GPIB Groups also allow turning a Amplitude / Frequency ramp into a series of commands. To create a GPIB group with a ramp, enable the desired channel, and select **A+F ramp** from the nearby drop-down. Waveforms for the amplitude and frequency will appear, and can be edited. Other data formats are **Raw**, which allows you to enter a raw string command, and **Parameter** which allows you to enter one or several (right click on the text boxes to insert or delete) numerical based

commands with a string prefix and postfix. When designing these text commands, remember to insert appropriate line termination characters for the device you are using (usually “\r\n”). RS232 Groups support **Raw** and **Parameter** data in the same way.

i. Run the sequence. Click the **Run Iteration 0** button or press F9. If all is well, the sequence will run and a progress bar will appear. Congratulations! Otherwise, a hopefully descriptive message will tell you what problems to resolve before trying to run a sequence.
j. Save the sequence with the **Save Sequence As** item in the **File** menu.

Features of Cicero

This section attempts to describe all of the features of Cicero beyond basic sequence editing.

Variables

Any numerical parameter (timestep duration, ramp value, etc) in a sequence can be either entered manually, or bound to a variable. To bind a numerical parameter to a variable, right click on the number box to see a drop down list of the existing variables.

To create a variable, go to the **Variables** tab. Click the **Add Variable** button. (If this button is disabled, it is because the lists are locked. Click the **Unlock Lists** button towards the bottom right of the window).

Variables have a name and a value. Values can be entered manually, bound to a list, or derived from other variables using equations. Duplicate variables with the same name are allowed, though not recommended.

To make a variable list-driven, right click on its value number box to get a drop-down list of lists to bind the variable to. To make the variable driven by an equation, click the **Derived?** check box, and enter the equation in the text box below. Variables can be referenced in equations by name in this text box, but only if their name contains no spaces or reserved characters. If the equation is formatted in a readable way, the equation value will be given below the text box. Otherwise, a message indicating the problem will appear, and the variable’s value will be set to zero. To see a list of supported operators and functions, right click on the text box and select the Help option.

In addition to the editable variables, there are two special variables **IterationCount** and **IterationNum**. **IterationCount** is the total number of iterations in the given list configuration, and **IterationNum** is the iteration number for the current iteration (spanning between 0 and IterationCount-1 inclusive).

Lists

Lists allow for batch processing, by allowing the value of a variable or set of variables to run through a pre-defined list of values. If lists are locked, then to edit them you need to unlock them. Up to 10 Lists can be individually enabled or disabled using the checkboxes above their labels. When data is entered in an enabled list, the list label will have a green background if the data is valid, and red if invalid. Reasons for invalid data include non-numerical text or empty lines. The total number of lines in a list will be given at the bottom of the list.

Multiple lists can be used at once, and can either be scanned “in parallel” or “in series”. To scan in series means that, for example, List 1 will take its first value while List 2 takes its first value. Then on the next run, List 1 will take its second value while List 2 will take its second value. To scan in parallel means that List 1 will take its first value while List 2 runs through each of its values.

Once List 2 has reached the end, it will start from the beginning with List 1 taking its second value, and so on.

To chose whether lists are scanned in parallel or in series, use the buttons labeled "," or "X". A "," between two lists means that they will be scanned in series, "X" in parallel. If two lists are to be scanned in series, they must have the same length, and you will not be able to lock the lists unless they do. If you attempt to lock lists, but there is a reason why it can't be done, this reason will be given below the lock button.

Variables get their values updated from the lists whenever the current iteration number is set. The iteration number ranges from 0 to the total number of iterations minus 1. The default running behavior, accessed by the F9 button, is to run iteration number 0. An arbitrary iteration number can be run by setting the iteration number using the **Set Iteration** button (either in the Variables tab or the Sequence tab) and then clicking the **Run Current Iteration** button. To run through all of the iterations, use the **Run List** button. To run through all of the iterations from the current iteration onward to the last one, use the **Continue List** button. To run through all the iterations, in a random order, use the **Run List in Random Order** button. There is not currently support to continue a random run or pick up where you left off in one, but this is a likely future feature.

Calibration Shots

When running through a batch list of iterations, it may be desirable to occasionally have a shot with some baseline sequence. This is supported through the Calibration Shots feature, accessible in the Variables tab. When enabled and loaded, you can specify whether to run the Calibration Shot as the first shot of a list run, the last shot, or after every Nth shot. To specify the sequence data corresponding to this calibration shot, use the **Load Sequence** button in the calibration shot configuration panel, and select a sequence file. This file gets loaded into memory, and incorporated as the calibration shot for the currently open sequence file. The calibration shot is always run with its iteration number set to 0. The calibration shot sequence object is saved as part of the currently open sequence. NOTE: Future changes to the sequence file which the calibration shot was loaded from have no affect on previously loaded calibration sequences. The calibration sequence is stored as an object within the parent sequence object, not as a reference to a file or file name.

Digital Colors

Colors used in the digital grid are stored in the Settings Data object. They can be set by using the **Settings Explorer**, and editing the DigitalGridColors field. To use a color other than the list of named colors, select color type custom, and right click on one of the color swatches.

To revert back to the default color scheme, remove all entries from the list of colors.

Dwell Word

After any sequence is run (or if a sequence is aborted), the output hardware reverts to outputting a set of dwell values on its analog and digital output hardware. These values come from the sequence's dwell word, which is the first enabled timestep in the sequence.

Output Now

Aside from running a sequence, output hardware can also be accessed by using various "Output Now" features. A timestep can be output, by right clicking on it and selecting **Output Now**. Outputting a timestep in this way causes the analog and digital output channels to output the value that they would have at the end of the selected timestep. Under some circumstances, it is desirable to have analog channels ignore these output now requests (such as when the analog values are used as the setpoints of large power supplies, or of high power lasers). Instead of outputting the value of the selected timestep, these channels can instead output their dwell value, as determined from the dwell word.

This option can be configured on a per-channel basis by editing the logical channel for a given analog channel in the **Channel Manager**, or globally for all analog channels by going to the Settings Explorer and setting the **OutputAnalogDwellValuesOnOutputNow** field.

If timestep output now is successful, a message to this effect appears in the status bar at the bottom of the window, and the background for the timestep turns grey. If the timestep does not change color, this is a sign that some error prevented the timestep from being output. You can examine the message log tab to determine what the error was.

Note: There is no guarantee of synchronization between various channels when using output now. All of the channels on a given card will update simultaneously, but different output cards will update at different times. Thus it is possible, for short periods of time (~ms) while the outputs are changing, for the collection of output channels to be in neither the new state nor the old state, but some transitional state with some channels in the new state and some in the old state. This should be kept it mind if, for safety or other reasons, you are for instance relying on channel a to be false whenever channel b is true. (It is anyway extremely unwise to rely on this unless you use external circuitry to enforce these safety conditions, for a wide variety of reasons beyond this output now quirk).

Running output now on a timestep DOES NOT output gpib or rs232 commands. To output these, go to the gpib or rs232 tab, select the desired group, and click the **Output Now** button. Note, the output now feature for gpib Amplitude + Frequency ramps is not supported (as these cannot be fulfilled by outputting just a command, but require a series of commands over a extended length of time). Channels with this data type will be ignored during a gpib output now.

Overrides

Any analog or digital channel can have its value overridden in the override tab. When a channel is overridden, its value is determined by the override value rather than by the sequence object. Override information is stored in the settings object, rather than the sequence object.

Changing the override status or override value of a digital channel will automatically re-output the last timestep to be “output now”. Changes in analog channels require clicking on the re-output button.

Hotkeys

A number of features in Cicero can have hotkeys bound to them. The most commonly useful is to bind a hotkey to a timestep. Whenever this hotkey is pressed, the given timestep is output using the “output now” feature. To set a hotkey for a timestep, right click on the timestep and navigate to **the Set Timestep Hotkey** context menu option. Select the text box and type a alphanumeric character. The hotkey for this timestep will now be set to CTRL+[character], and the character will be displayed in curly brackets next to the timestep number.

Hotkeys can also be assigned in a similar way to digital and analog overrides.

Pulses

In many circumstances, it is desirable to have certain digital events pre-triggered, occurring a specified amount of time before other events. For instance, when creating an imaging pulse to take a picture of an atom cloud, it may be necessary to open a mechanical shutter several ms before the image is taken, and then to flash on the light with an AOM for a short period of time. This can be somewhat awkward to do with a digital-value-per-timestep sequence design, especially if this shutter opening has to take place during some other operation completely unrelated to imaging your atoms.

Cicero has an additional way to specify digital output values, using a feature called “Pulses”. A Pulse is a description of digital data that allows for digital values to be changed in the middle of timesteps, with user specified amounts of pretrigger or delay from the timestep they are inserted into. These pulses are “written on top of” the normal digital data.

To define a pulse, go to the pulses tab and click the Create Pulse button. A pulse has a name, a start condition, a stop condition, and a pulse value. The pulse can be stopped or started at a time specified relative to the start of the timestep it is inserted into (**TimestepStart**), the end of the timestep (**TimestepEnd**), or according to the duration of the pulse (**Duration**). If either the start or stop condition of the pulse is set to **Duration**, then the other must not be. Cicero uses the start and stop condition of the pulse to determine when the pulse is active, and writes the pulse’s **Pulse Value** onto the digital channel outputs during this active time of the pulse.

For example, to create a pulse that will force the value of a digital channel to false for the 10 ms preceding the timestep that it is placed into, do the following. Set **Start Condition** to **Duration**, **End Condition** to **TimestepStart**, **Pulse Duration** to 10ms, and **Pulse Value** to false.

To create a pulse that will make a digital output true, will start 2ms after the start of the timestep it is in, and end 3ms before the end of the timestep it is in, set **Start Condition** to **TimestepStart**, **Start Pretrig/Delay Enabled** to true, **Start Pretrig/Delay** time to 2ms, **Start Delay** to true, **End Condition** to **TimestepEnd, End Pretrig/Delay Enabled** to true, **End Pretrig/Delay** time to 3ms, **End Delay** to false, and **Pulse Value** to true.

To insert a pulse into the sequence, right click on the digital box you wish to apply it to (in the sequence tab digital grid, corresponding to the channel and timestep you want the pulse to apply to), to see a drop down list of available pulses.

There are a wide variety of possible uses that these pulses can have, beyond just shutter pretriggers for imaging. Be creative. But keep in mind that Pulses are somewhat more resource intensive than normal digital data, so do not use them when they are completely unnecessary.

NOTES:

- Pulses do not affect the output now behavior of digital channels, they are ignored when using the output now feature.
- When using a variable timebase as the clock for your sequence, there is a possibility that the start or stop of pulses can be displaced by as much as 2 times the period of the underlying master clock. This will occur if the pulse tries to change the value of a digital channel too close to a time when the channels are changing anyway (due to an analog ramp, or the beginning of a timestep, etc.). This is more likely to happen if pulses are used in close proximity to high time resolution analog ramps. Occurrence of these displacements should be fairly rare, and unless you are very sensitive to the timing of these pulses, it should not make much difference.

Analog Group Time Resolution

In the analog group editor tab, there is a field to enter the time resolution of the analog group being edited. This field is only meaningful if you are using a variable timebase clock. In such a case, the time resolution given here will set the rate at which the variable timebase clock will run while the analog group is active. A group is considered active if any of its channels still has changing data. If multiple analog groups are running at the same time, the lower value of the time resolution is used. To take full advantage of the benefits of the variable timebase clock, avoid the temptation to set extremely fast time resolution for long running analog groups. However, make sure to keep this time resolution in mind when designing analog groups that have fast ramps. The default time resolution for an analog group is relatively large (1ms), and this may be unsuitable for certain ramps.

Common Waveforms

There are a set of waveforms that are editable but that do not belong to a specific analog group. These are called the **Common Waveforms**, and are selectable and editable from the Common Waveforms tab.

A special interpolation type supported by Cicero, which allows you to combine various common waveforms, is the Combine Waveforms interpolation type. This allows you to, for example, make an exponentially damped sinusoid by multiplying a sinusoidal waveform with a exponential one. Keep in mind that the combine waveforms interpolation type uses substantially more resources than the normal interpolation types, and thus should not be used when the normal interpolation types can do the job.

To use a common waveform in an analog group, enable that channel, and select the desired common waveform from the drop down list to the right of its enable/continue button. The waveform for this channel will be displayed in gray, to indicate that it is a common waveform rather than a waveform editable from within thin analog group.

To copy a waveform from and analog (or gpib) group to the set of common waveform, select the waveform, then right-click on the on the waveform editor and select the copy to common waveforms context menu item.

Saving and Inserting Subsequences

On occasion it may be desirable to take a part of one sequence and insert it into another. To do this, open the sequence that you would like to copy from. By right clicking on timesteps and selecting the **Mark** menu item, mark the timesteps that you would like to copy. Marked timesteps will appear with a orange-pink background color. Under the file menu, select Save Marked Timesteps to create and save a sequence made up of the selected timesteps (this will include all the variables, common waveforms, analog groups, gpib groups, rs232 groups, and pulses that were used in those timesteps).

In the destination sequence, select Insert Sequence from the file menu to insert a sequence from a file into the currently open sequence. Note: Inserting a sequence may cause duplicate groups, pulses, variables etc. with the same name, if the inserted sequence included such items with the same name as the open sequence. It then may be necessary to manually re-name, or to manually re-assign groups and remove groups to eliminate this potential confusion.

Run Logging

Every time a sequence successfully runs, it generates a run log file which contains the full sequence and settings objects that were used for that runs (this includes the values of all the variables), as well as a timestamp indicating when it was run. These files accumulate in the **RunLogs** subdirectory of the Cicero installation. RunLogs can be opened and batch-examined with the Elgin program. Elgin is currently not described at any length in this program, but its features are straightforward enough that hopefully they can be deduced by a new user.

Sequence “Modes”

A `SequenceMode` is a data structure that stores the enabled/disabled and hidden/visible state of each of the `TimeStep`s in a sequence. Modes can be saved and loaded using the mode drop-down list box and its neighboring buttons.

These are located right next to the timesteps panel, in the sequence page. When a mode is selected from the list, it is loaded, meaning that all of the sequence's timesteps are enabled or disabled and are shown or hidden based on the data stored in this mode. Whenever the Store button is clicked, the current state of all the timesteps is saved into the currently open mode (and the name of the mode is changed, if the name textbox has been updated).

The modes feature allows a single sequence object to be quickly switched between various functionalities. For example, by enabling and disabling the right timesteps, the same sequence object can both create a cloud of atoms out of a magnetic trap, or out of a subsequent optical trap, or after a subsequent evaporation, etc. Using a single sequence object to perform all these "checks" en route to the final result can allow you to re-use the parts of the sequence that would be common to all these procedures, and avoid needing to tediously manually update all your sequence files whenever some detail in these shared steps was changed.

The modes feature also allows for more sophisticated batch runs than what could be achieved using just normal list scanning + calibration shots. This is accomplished through use of a special named variable. If a variable with the name **SeqMode** exists, then before each run, Cicero will set the sequence mode to the mode corresponding to the nearest integer value of **SeqMode** (0 for the first item in the list of modes, 1 for the second, etc). Thus this allows for a batch run in which the sequence mode changes between individual shots.

Miscellaneous

Help! Bugs!

If (gasp!) you should run across a bug when using Cicero, do not fret. Send an email describing the bug and the circumstances under which it occurred to akeshet AT mit DOT edu. If the bug was accompanied by an exception window, giving a stack trace of the exception, paste this text into the email as well, as this information makes tracking down and fixing bugs much easier.

Why do the buffers not seem fully used up?

A monitoring feature in Atticus is that at the end of each run, Atticus will report for each of its output tasks the size of the buffer for that task, as well as the number of samples that were actually generated. The perceptive user may note that, when using a variable timebase:

- The buffers do not appear to get completely used up.
- Different tasks report having generated a different number of samples.

The reason that some of the buffers do not get completely used up is that the NI drivers require buffer sizes to be a multiple of 4, so extra unused samples are added to the end of buffers to make them reach this condition, but these extra samples are not used. However, certain output cards also will only report the number of samples generated rounded up to the nearest multiple of 4 (in my experience, these are the digital output cards) whereas other cards will show the exact number of generated samples. Thus tasks on different cards may claim to have generated a different number of samples.

These monitoring features are quite useful in tracking down and debugging timing problems. If a sequence is run repeatedly, the number of samples generated should be the same each time. If not, this may indicate that the timing signal is being degraded. If a variable timebase is being generated on one computer and then shared over a long cable to another computer, it may be necessary to add a terminator at the receiving end to eliminate intermittent timing glitches.

File Formats

All the files saved by Cicero are saved using .NET binary serialization. This means that it is fairly straightforward to write a program using a .NET compatible language (such as C#) that will open, manipulate, and save any Cicero object, should the need to do custom processing of these files arise. To see examples of how to load and save files, see the code for the Storage class.

Code Overview

This is intended as a brief overview of the source code of Cicero, for those who would upkeep the software or add new features. The information contained here is of course only a sketch. But the source code is, for the most part, well documented, so it should be possible to find one's way around after a bit of experience.

The Visual Studio "solution" for the source code contains several main "projects":

Atticus: This project contains all of the UI code for the server, and all of the hardware side code.

Cicero: This project contains all of the UI code for Cicero, as well as some code for sending those sequences to the server.

DataStructures: This project does not contain any executable. Instead, it contains class definitions for all of the classes use to describe and store sequences. In addition, it contains some of the code used to communicate between client and server, and a few auxiliary classes shared by multiple projects but without a unifying theme.

Elgin: This is a simple RunLog viewing program, to view the RunLogs produced by Cicero. It may eventually be extended to do things like batch processing.

Some of the key important files, perhaps worth looking at first in getting acquainted with the code:

Atticus/ServerRuntime/AtticusServerRuntime.cs: This file is the main server object. It receives requests over .NET remoting (interface defined by ServerCommunicator), detects hardware channels, creates buffers and triggers.

Atticus/ServerRuntime/DaqMxTaskGenerator.cs: This class contains a number of static methods used to generate digital and analog output tasks out of sequence objects.

Atticus/AtticusServer.cs: Entry point for the Atticus executable.

Atticus/Form1.cs: Top level user interface for Cicero. Best viewed with Visual Studio designer.

Cicero/Controls/MainClientForm.cs: The top-level user interface for Cicero. Most of the guts of the user interface resides in sub-controls of this. The easiest way to understand this code is by viewing it with Visual Studio's designer.

Cicero/Program.cs: entry point for Cicero.

Cicero/Controls/RunForm.cs: A user interface object which displays a progress bar during a run. RunForm also contains the client-side code for orchestrating a sequence run, making the appropriate calls to ServerManager to send sequence objects, settings objects, and start runs.

Cicero/Storage.cs: Contains static references to the SettingsData and SequenceData objects being edited, as well as methods for loading and saving.

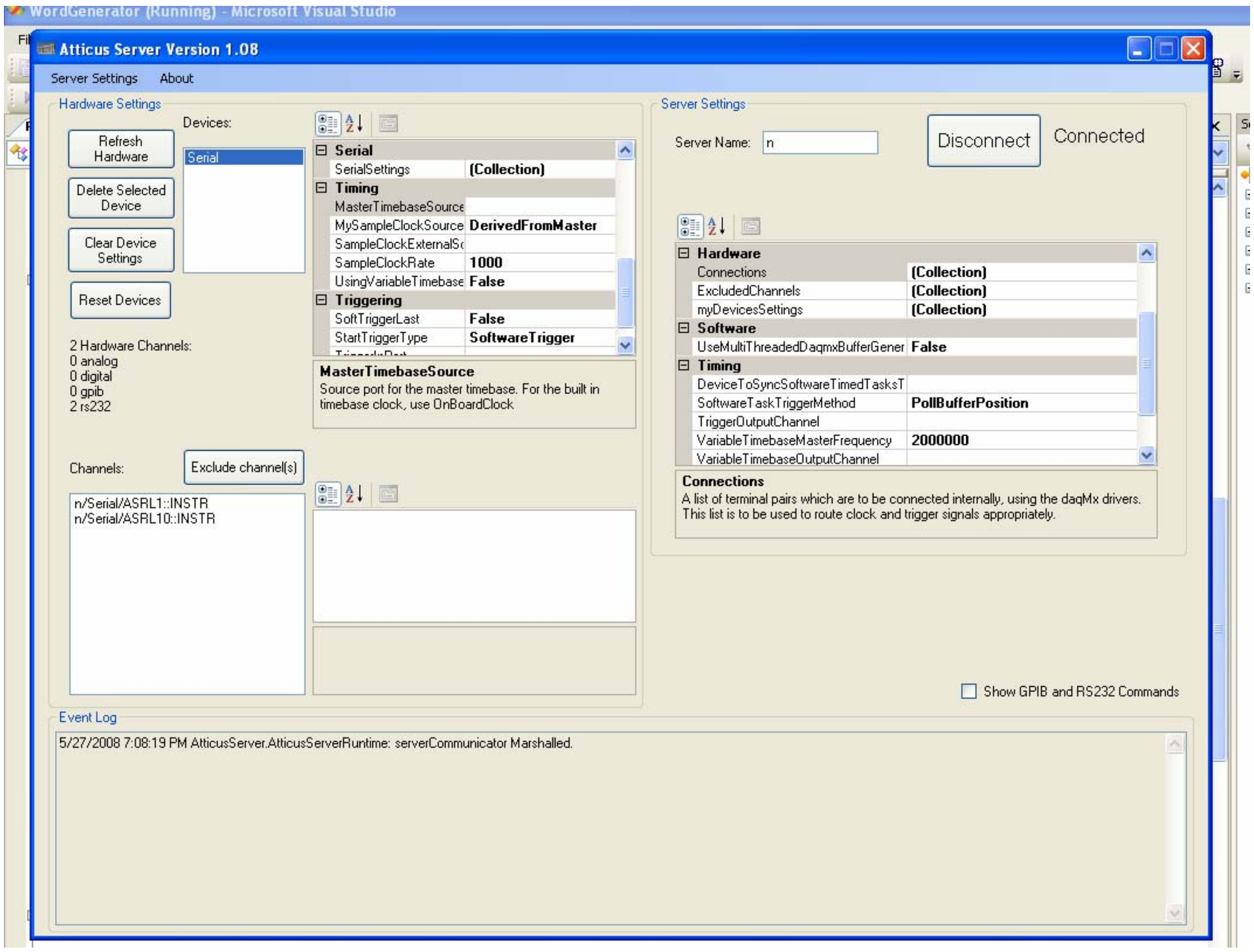
DataStructures/SequenceData/SequenceData.cs: The class which stores a sequence. This class is effectively "edited" by Cicero, and then sent (along with SettingsData) to the servers, which turn the SequenceData object into output buffers. This class contains the code used to make the various types of buffers out of sequence objects.

DataStructures/SettingsData/SettingsData.cs: The class which stores "settings". This includes mapping from channel ID integers to Hardware channels (thorough LogicalChannelManager), addresses of servers

DataStructures/Communication/ServerManager/ServerManager.cs: Contains a lot of client-server communication code.

Screenshots

Atticus



2 Hardware Channels:
0 analog
0 digital
0 gpib
2 rs232

n/Serial/ASRL1::INSTR
n/Serial/ASRL10::INSTR

Serial	
SerialSettings	**(Collection)**
Timing	
MasterTimebaseSource	
MySampleClockSource	**DerivedFromMaster**
SampleClockExternalS	
SampleClockRate	**1000**
UsingVariableTimebase	**False**
Triggering	
SoftTriggerLast	**False**
StartTriggerType	**SoftwareTrigger**

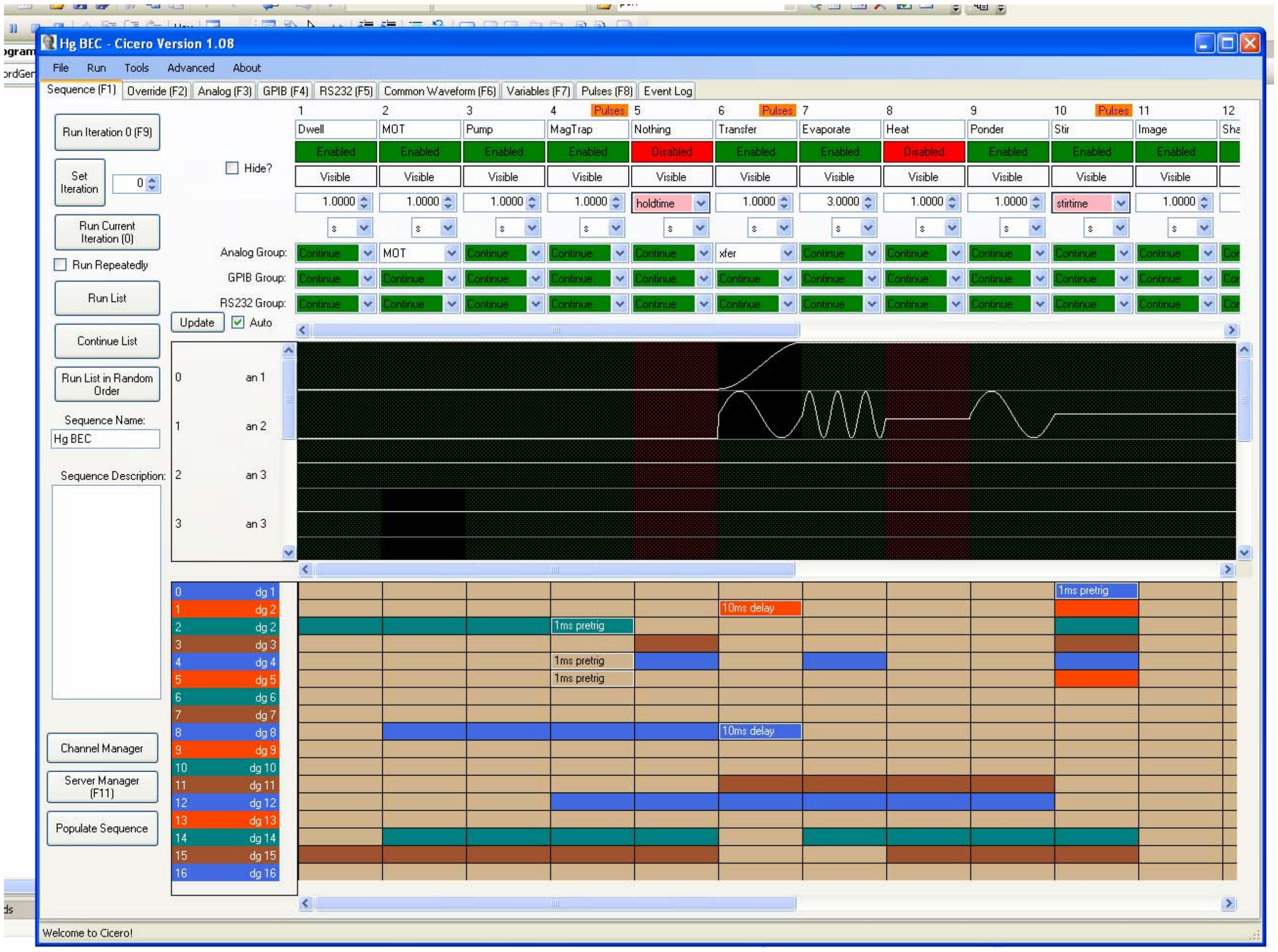
MasterTimebaseSource
Source port for the master timebase. For the built in timebase clock, use OnBoardClock

Hardware	
Connections	**(Collection)**
ExcludedChannels	**(Collection)**
myDevicesSettings	**(Collection)**
Software	
UseMultiThreadedDaqmxBufferGener	**False**
Timing	
DeviceToSyncSoftwareTimedTasksT	
SoftwareTaskTriggerMethod	**PollBufferPosition**
TriggerOutputChannel	
VariableTimebaseMasterFrequency	**2000000**
VariableTimebaseOutputChannel	

Connections
A list of terminal pairs which are to be connected internally, using the daqMx drivers. This list is to be used to route clock and trigger signals appropriately.

```
5/27/2008 7:08:19 PM AtticusServer.AtticusServerRuntime: serverCommunicator Marshalled.
```

Cicero



Sequence Name: Hg BEC

Set Iteration: 0

	1	2	3	4	5	6	7	8	9	10	11	12
				Pulses		Pulses				Pulses		
	Dwell	MOT	Pump	MagTrap	Nothing	Transfer	Evaporate	Heat	Ponder	Stir	Image	Sha
	Enabled	Enabled	Enabled	Enabled	Disabled	Enabled	Enabled	Disabled	Enabled	Enabled	Enabled	
	Visible	Visible	Visible	Visible	Visible	Visible	Visible	Visible	Visible	Visible	Visible	
	1.0000	1.0000	1.0000	1.0000	holdtime	1.0000	3.0000	1.0000	1.0000	stirtime	1.0000	
	s	s	s	s	s	s	s	s	s	s	s	
Analog Group:	Continue	MOT	Continue	Continue	Continue	xfer	Continue	Continue	Continue	Continue	Continue	Co
GPIB Group:	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Co
RS232 Group:	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Continue	Co

0	dg 1
1	dg 2
2	dg 2
3	dg 3
4	dg 4
5	dg 5
6	dg 6
7	dg 7
8	dg 8
9	dg 9
10	dg 10
11	dg 11
12	dg 12
13	dg 13
14	dg 14
15	dg 15
16	dg 16