Girder User Manual

© 2008 Promixis, LLC

Girder

What can Girder and Promixis do for you?

by Ron Bessems

Girder is a powerful automation application that fits a wide variety of needs. From automating or remote controlling a Windows application to industrial and commercial automation to home automation Girder is up to the task.

Promixis offers superior products and services geared towards automating your environment and putting you in control.

Girder User Manual

© 2008 Promixis, LLC

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: November 2008 in Santa Barbara, CA

Special thanks to:

Everyone who over the years has contributed to Girder, NetRemote and Promixis, from the early beginnings on AVS forums to the current active members on our forum (you know who you are!) a big thank you goes out to all of you.

Table of Contents

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

© 2008 Promixis, LLC

You can't always control the wind, but you can control your sails.

Anthony Robbins

© 2008 Promixis, LLC

Part I

1 Introduction

Welcome

Welcome to Girder, the award winning Windows automation utility from Promixis. As the most powerful and feature rich utility in its class, Girder offers limitless possibilities. From home theater automation, PC-based media players, business applications and IT functions, to home automation, Girder can do it all. Plus Girder's online library contains hundreds of Plugins, pre-configured Actions, and support for leading third party hardware devices and software applications.

Control your PC using Remote Control Devices

Girder works with leading IR and RF remotes including.

- NetRemote from Promixis running on a Pocket PC.
- USB-UIRT.
- SnapStream Firefly.
- ATI Remote Wonder.
- Streamzap.
- IR Trans.

Or you can create on-screen controls operated with a standard mouse.

- Link Girder Actions to desktop or folder icons.
- On-Screen Display (OSD) features can be used to produce large-text menus.
- Run NetRemote from Promixis on the Girder PC to produce sophisticated graphical control panels.
- Use the built-in Web server to build control applications which run in any web browser (Girder Pro).

Automate Your Home Theater PC and Digital Home Entertainment

For Home Theater PCs, Girder controls Windows programs (media/DVD players) and hardware devices (sound cards, X10 and serial devices such as projectors, and receivers). Girder is the behind the scenes glue providing full control of your multimedia experience.

- Supports media and home theater applications such as Winamp, Windows Media Player, PowerDVD, WinDVD and ZoomPlayer.
- From start up to mid-movie play & pause to shut-down, Girder can do it all.
- Controls home theater hardware such as lighting, receivers and projectors via serial, network or IR control.
- Setup custom controls for your home theater devices, tuners and cable/satellite TV set top boxes.
- Use Girder with your remote to play music from your digital media library, select playlists, crank the volume and play digital DJ for your next party!

Automate Your Home

- Use Girder to automate your home. Control your lighting, security system, climate settings and more. Girder is less expensive and more flexible than most dedicated home automation packages!
- New to Girder 5, the Device Manager allows control of devices across multiple technologies (such as X10 and Insteon) in a consistent way, automatically generating control panels for

© 2008 Promixis, LLC

web browsers and Promixis NetRemote (Girder Pro).
- Girder hosts Lua, an advanced scripting language, that can be used to easily create complex and powerful automation applications.
- Use Girder on multiple PCs to automate your home network (Girder Pro).

Simplify Business & IT Task Automation

Automate repetitive or scheduled tasks using Girder. Contact Promixis for attractive volume pricing on departmental and multi-seat licenses.

Girder Grows with You

The more you use Girder, the more possibilities you will see. Using our extensive plug-in library, you will be able to control more every day. That is because the plug-in list expands constantly as we, along with the large Girder community, build new features and controls. Just download a new plug-in and you will be off and running with new Girder capabilities.

Using This Manual

Girder rewards a little learning with a lot of power. Automation and Home Automation in particular are quite complex problems and will take some effort on your part to solve, but rest assured Girder can do it all.

We recommend you have a look at the Getting Started section first. It will explain some basic concepts of Girder.

To get more detailed description of the various components of Girder take a look at Girder Concepts, Events Reference and the User Interface Guide.

Examples are included in the Examples section.

Girder Licence Versions

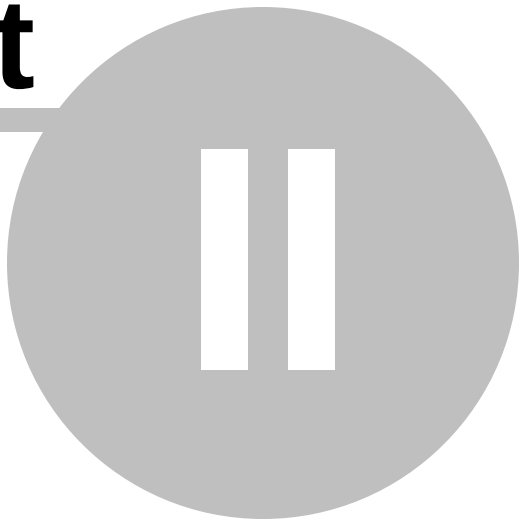
Some features are limited to the Pro and Whole Home Pro versions of Girder. Some of these are marked in green (Girder Pro) or black (Girder Pro). These features will be available during the trial period, but will be disabled when a Standard licence is installed. The Whole Home Pro (WHP) version has the same features as the Pro version with the addition that you may run as many copies as you need in your home for non-commercial use.

Version Control

The What's New (Since Girder 5.0) section identifies changes since Girder 5.0 which are covered in this manual. The manual may lag behind Girder minor versions, in which case you should consult the Promixis online forum for updated release notes.

© 2008 Promixis, LLC

Part



2 Getting Started

To understand how Girder works you must first understand it's core principle, Girder is an event processor. It deals with events and actions. Events trigger actions, the sun goes down (event) turn on the ligths (action).

Events could be for example:

- Someone presses the doorbell
- A motion detector is triggered
- A scheduled event is triggered
- The sun sets
- The sun rises

A few examples of actions are:

- Turn on a light
- Start an application
- Turn on some music
- Change channels on your TV

All of these can be combined in infinite ways and you can have multiple events trigger one action or one event trigger multiple action. On top of that Girder uses a scripting language to deal with more complex automation tasks.

While Girder has great power and depth, it will does not take long to get started by linking a supported remote to a supported multimedia application. Later, you can use the more advanced features of Girder to fine tune your setup and to control applications that are not directly supported.

Three important steps are covered in the following sections.

1. Setting up Girder.
2. Adding a Supported Remote.
3. Adding a Supported Application.
4. Controlling Consumer Electronics.

When you have completed these steps you will be able to control the application using the remote.

Tip: Even if your remote or application is not specifically supported, it is still likely that you can accomplish your intentions using the more advanced features of Girder which are described in subsequent sections.

2.1 Setting up Girder

Installing Girder

You can download the latest version of Girder from www.promixis.com, or it may have been supplied to you on CD. In either case, installation is a simple matter of running the installer program and following the prompts.

Tip: When Girder starts, it may trigger warning messages from security programs. You should verify that these are due to Girder and tell your security program to ignore them in future.

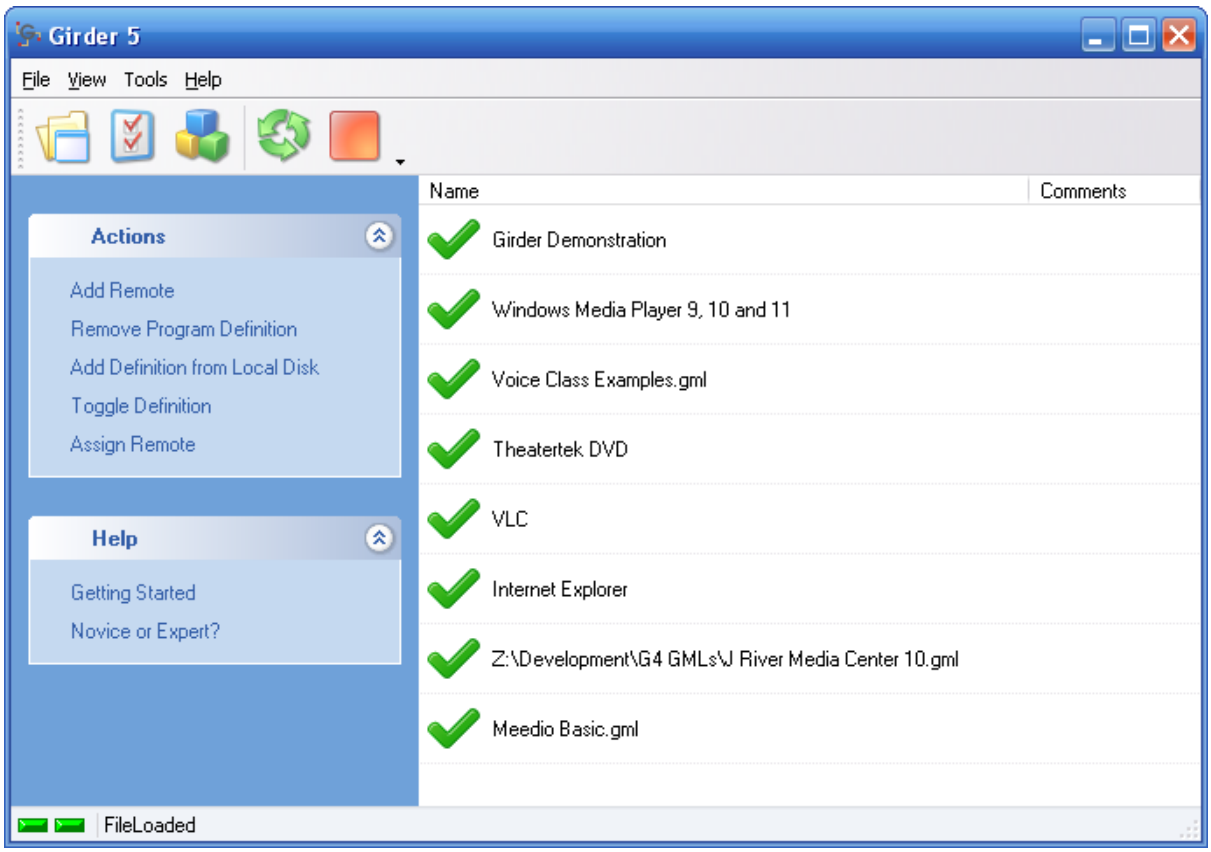
© 2008 Promixis, LLC

Starting Girder for the first time

If you take the default options during installation, Girder will be started automatically. If not, start it from the desktop icon or the start menu.

Notice the Girder task bar notification area icon.

And the Girder user interface window.



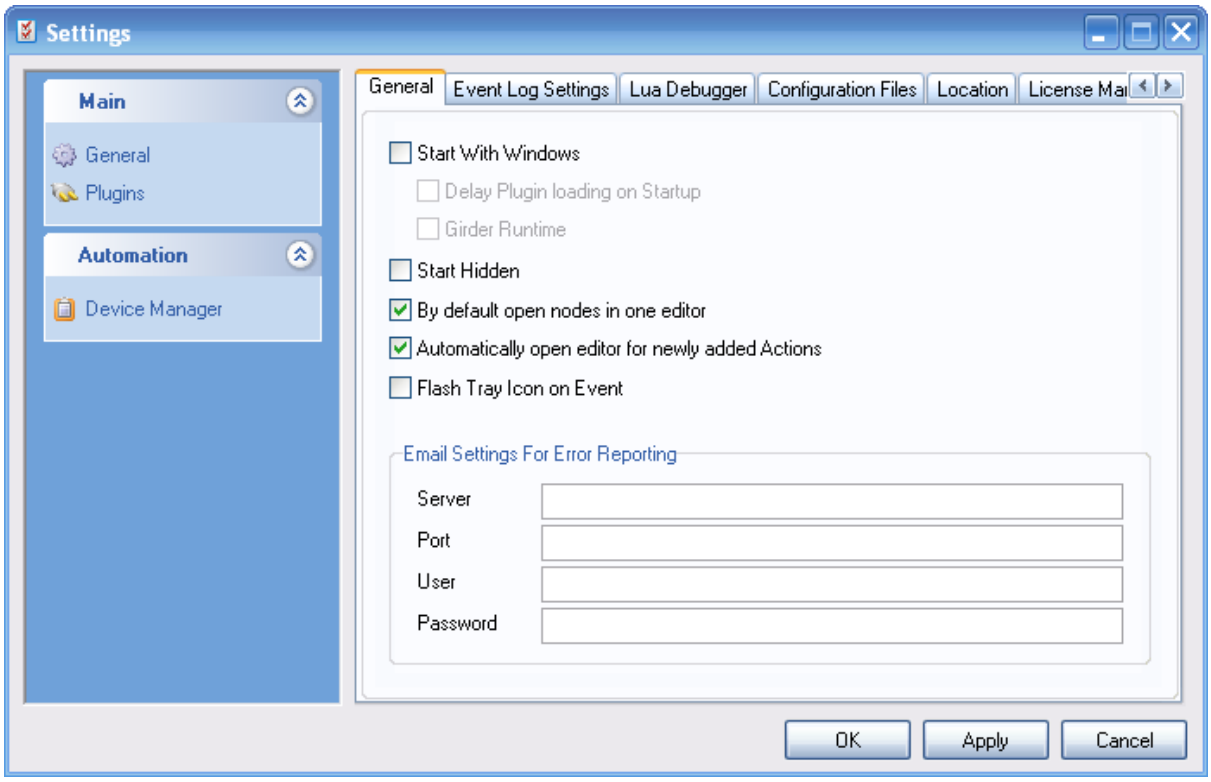
The user interface is in **Novice** mode which we will use throughout this section. You can switch between **Novice** and **Expert** mode using the **View** menu. When you close the user interface window, Girder continues to run in the notification area.

- To shut Girder down completely, right-click the notification icon and click on **Exit Girder**.
- To show the user interface, right-click the notification icon and click on **Show Girder**.

Configuring Girder

Bring up the Settings dialog (**File** menu or toolbar button) and click the **Generic** settings group and then the **General** tab.

© 2008 Promixis, LLC



You may want to select **Start With Windows** and **Start Hidden** to ensure that Girder runs at all times but does not show its user interface when your computer starts.

It is a good idea also to visit the **Location** tab and personalize this information which is used by Girder for the Weather application and to determine sunrise and sunset times.

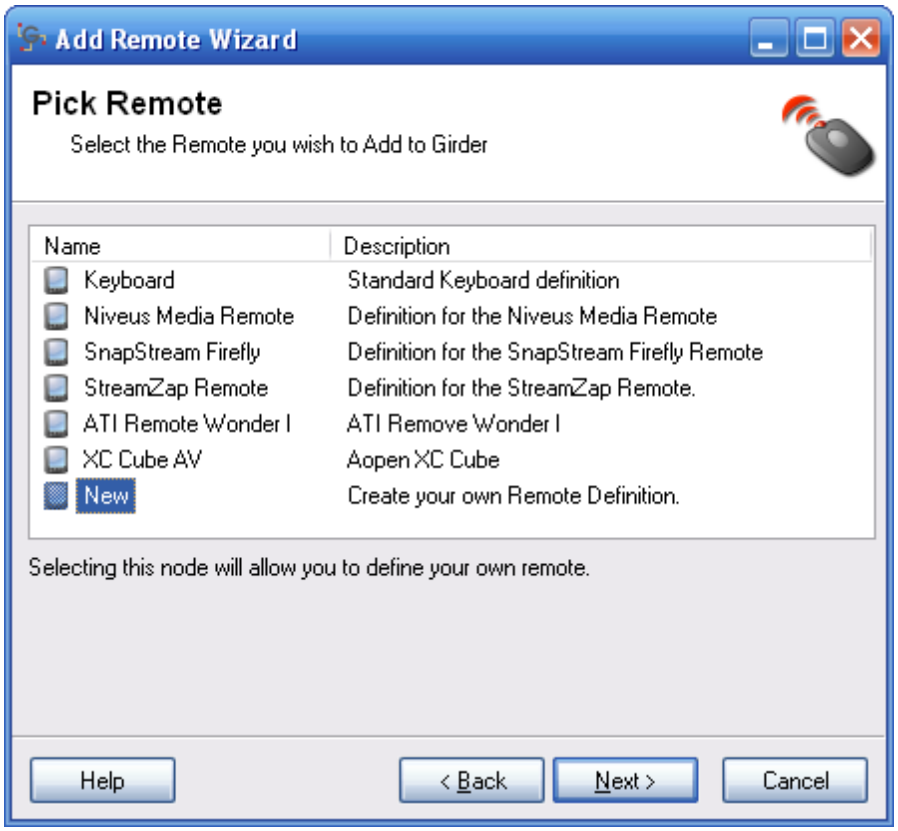
2.2 Adding a Supported Remote

Using the Add Remote Wizard

Note the add remote wizard adds a remote as an INPUT device for Girder, this is not the place to add actions that send out IR signals. This is done by the IR Profile Editor.

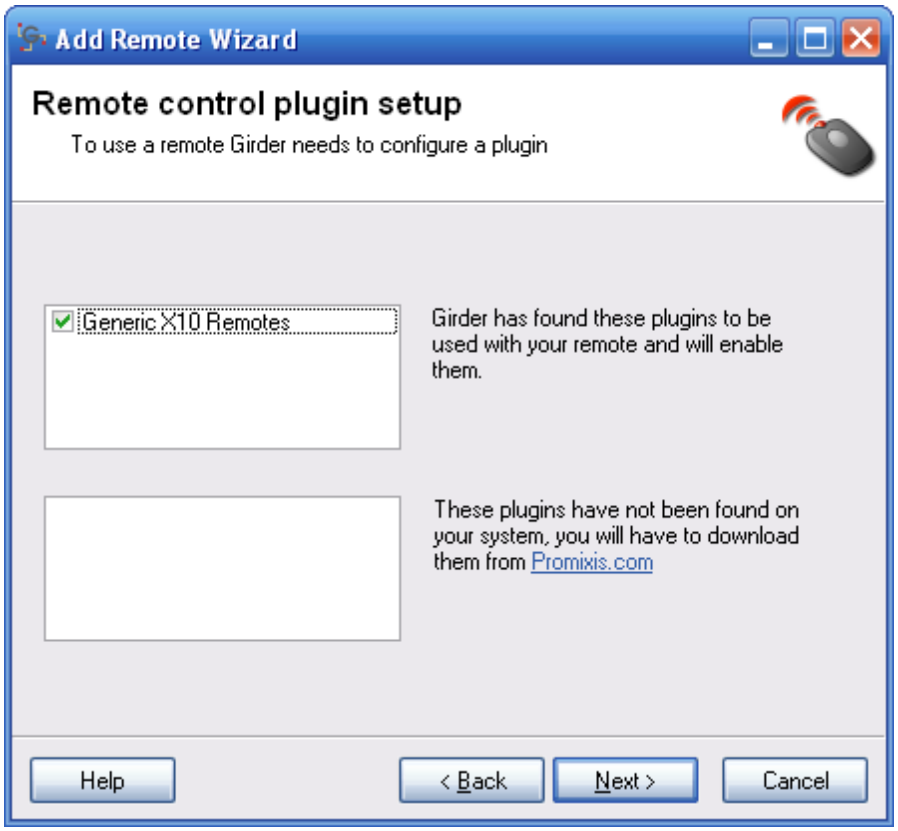
From the Event Mapping Editor click on **Add Remote** in the **Actions** task box, or click "Add Input Remote Wizard" from the main Girder window Tools menu and press **Next** to step the wizard on to the list of supported remotes. Highlight your remote in the list.

© 2008 Promixis, LLC



Press the **Next** button. If your remote requires a Plugin (or sometimes more than one) to operate, a step will appear to allow you to install or enable it.

© 2008 Promixis, LLC



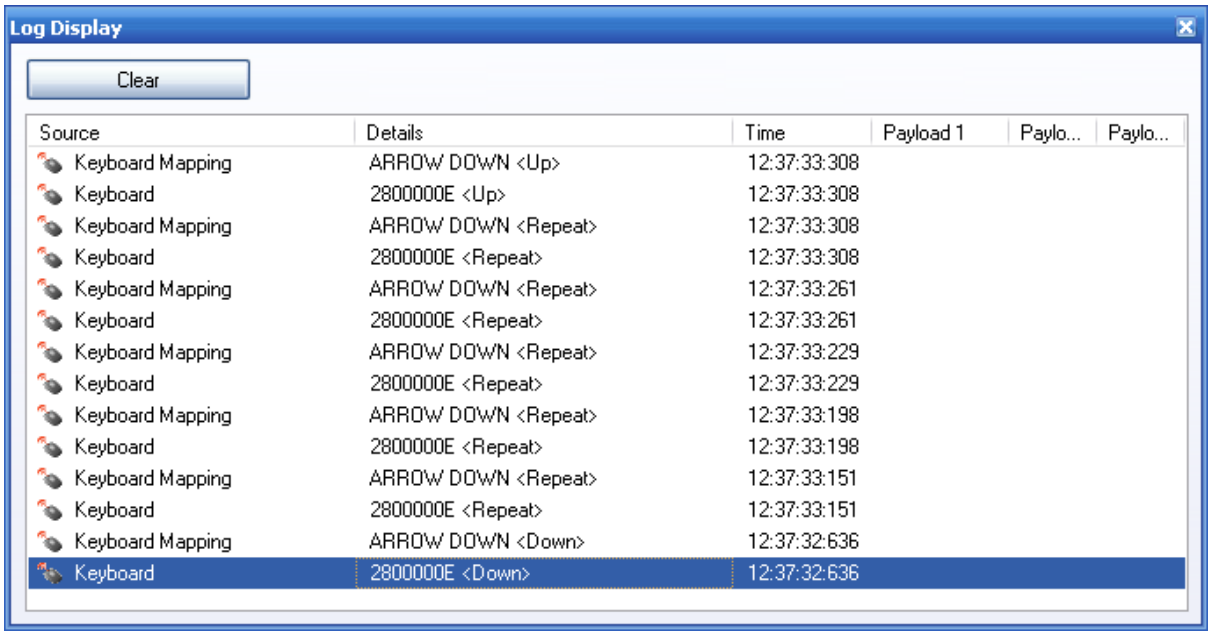
Usually, you can just press **Next** again to enable the Plugin(s), but if there is an entry in the lower box, you will need to Cancel the wizard, exit completely from Girder, download and install the Plugin, restart Girder and re-run the Wizard. Once all required Plugins are in the upper box, you can proceed.

The next step allows you to test the remote and provides some troubleshooting tips in the event it does not work.

At the final step, simply press **Finish** to complete the operation.

To test this, bring up the Logger window (**View** menu or **F4**). When you operate your remote, you should see a "raw" event, immediately followed by a "mapping" event. The event generated when you press the "down" button will be different for each type of remote, but the mapped event, "ARROW DOWN", is always the same (although you can change which button produces "ARROW DOWN" if you want to).

© 2008 Promixis, LLC



If the remote is not generating events, it is possible that the default settings of the Plugin do not match your system. Bring up the Settings dialog (**File** menu), select the **Plugins** group and look for the settings tab for your particular Plugin. More help on settings for individual Plugins is given in the Plugins Reference.

Audio Mixer and Mouse Controls

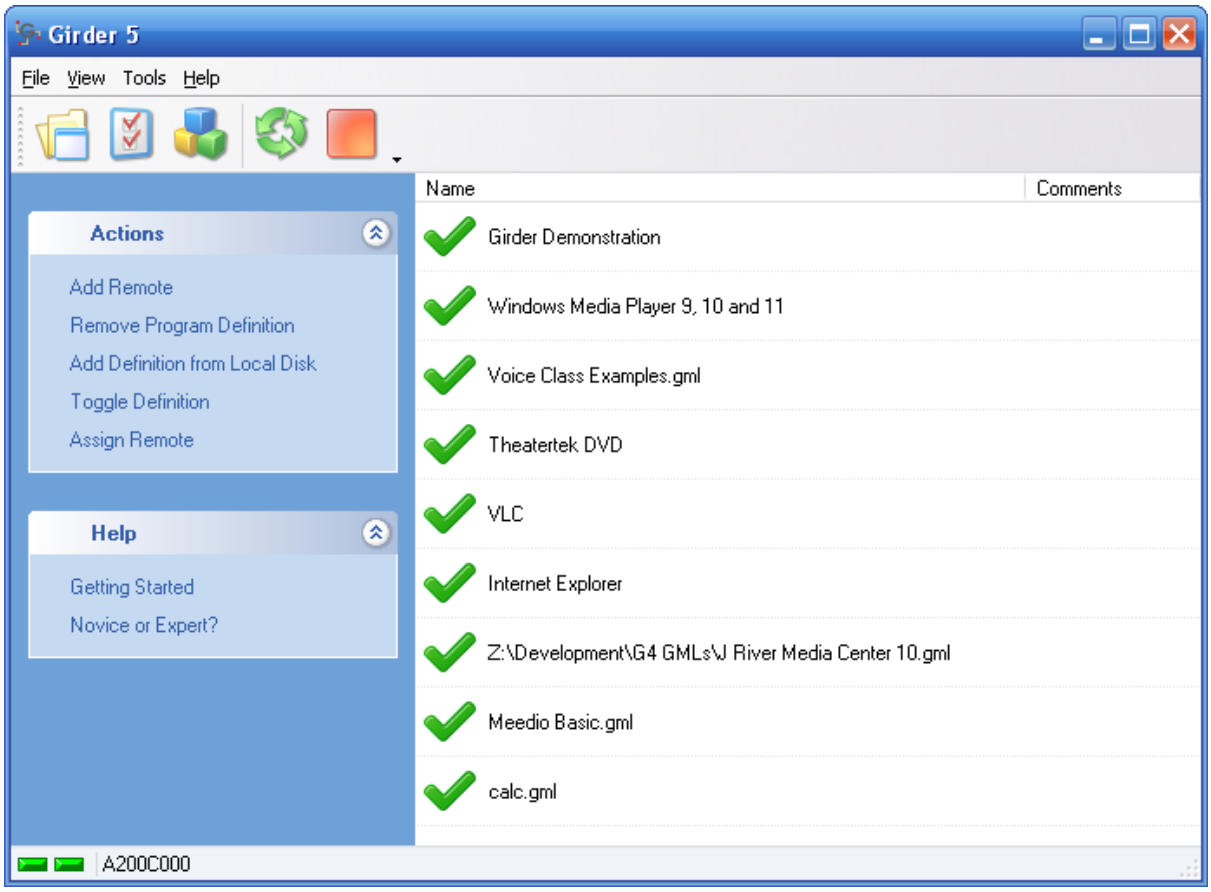
If your remote has volume controls or a mouse mode, these should work at this point. If not, revisit the Settings dialog and make sure the "Audio Mixer (Sound)" and the "Mouse Control" Plugins are enabled. Each of these has a settings tab with some adjustments you can make. If the pointer is difficult to control accurately, you can adjust the dynamics on the **Mouse Controls** tab of the **Plugins** settings group.

2.3 Adding a Supported Application

Download an Application Definition file (GML file) from promixis.com. Then click on **Add Definition from Local disk** in the Actions task box. Find the file you downloaded and open it.

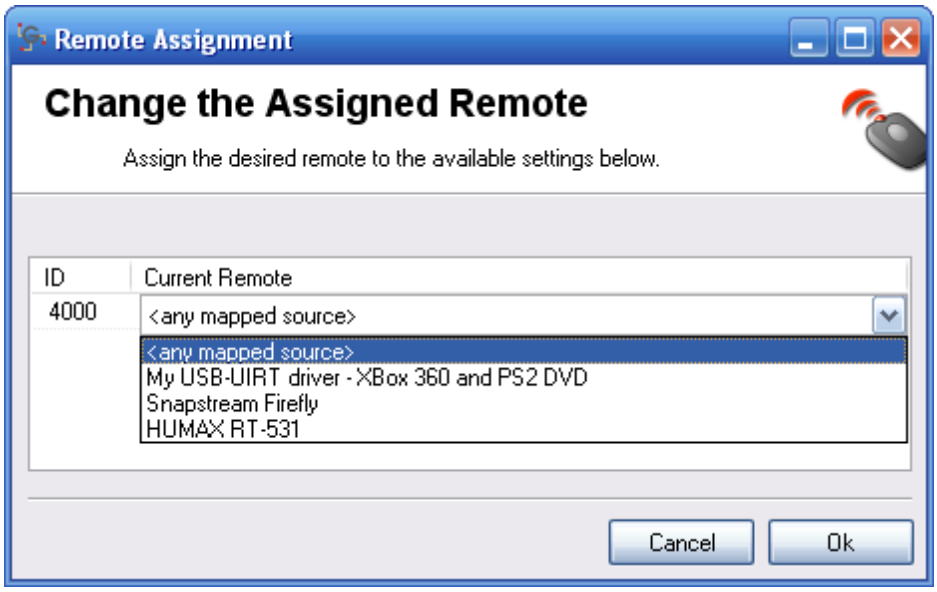
Returning to the main user interface window, the definition shows up in the list with a green tick indicating that it is active.

© 2008 Promixis, LLC



Assigning the Remote to the Application

Highlight the definition in the list and click on **Assign Remote**.



© 2008 Promixis, LLC

If there is more than one row, this allows you to assign different remotes to different groups of functions in the program (for example, a keyboard for typing functions and a push-button remote for transport functions).

Click on a row to highlight it, wait a little and then click again to produce a drop-down list of available mapped remotes. Select the remote that you previously added.

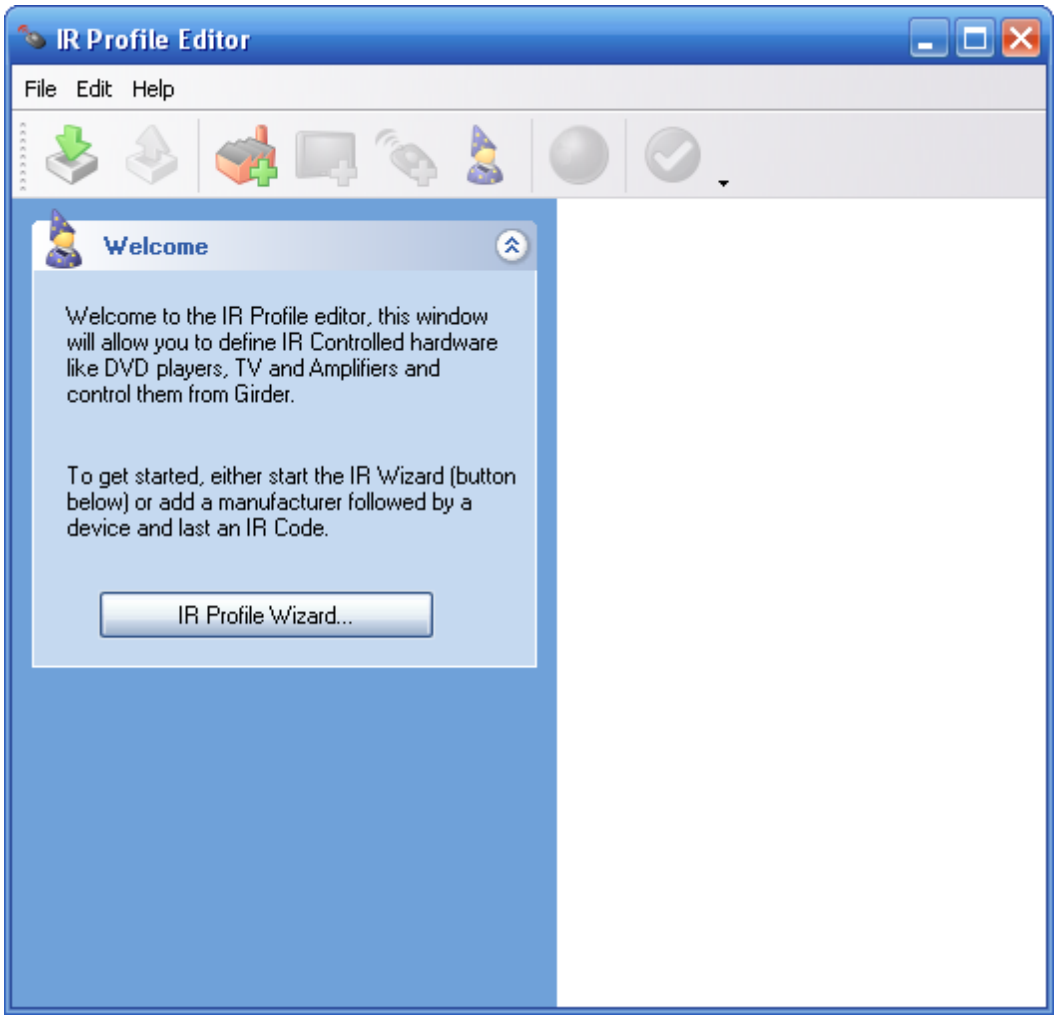
Press OK to reassign the application to your remote.

You should now be able to control the application using the remote.

2.4 Controlling Consumer Electronics

Girder can control almost any consumer electronics device that uses IR (limited by the IR transmitter you own), IR is short for Infra Red. It a type of light that we cannot see and is used in 99% of the consumer electronics available. In case you are wondering what consumer electronics are, this can be anything you own, a TV, Amplifier, Tuner, DVD or BluRay player to name a few.

The configuration for this is handled in the IR Profile Editor. You can find this under View->IR Profile Editor on the main Girder interface. The easiest way to add a new IR Profile is to use the wizard, you can do it by hand but that would take a bit more work.



© 2008 Promixis, LLC

So let's start by clicking on the IR Profile Wizard button. Or Edit->IR Profile Wizard. The wizards opens on it's welcome page. Simple click next and select the manufacturer of your device from the list. If your manufacturer does not appear click on the "My manufacturer does not appear in the list" button and give the new manufacturer a name. If the manufacturer did appear you will now get a list of devices that we have available for this manufacturer. It is sorted by device type, if you can find your device try the search box. If it is still not available you'll have to add your own by selecting the "My Device is not in the list I'll make my own" radio button.

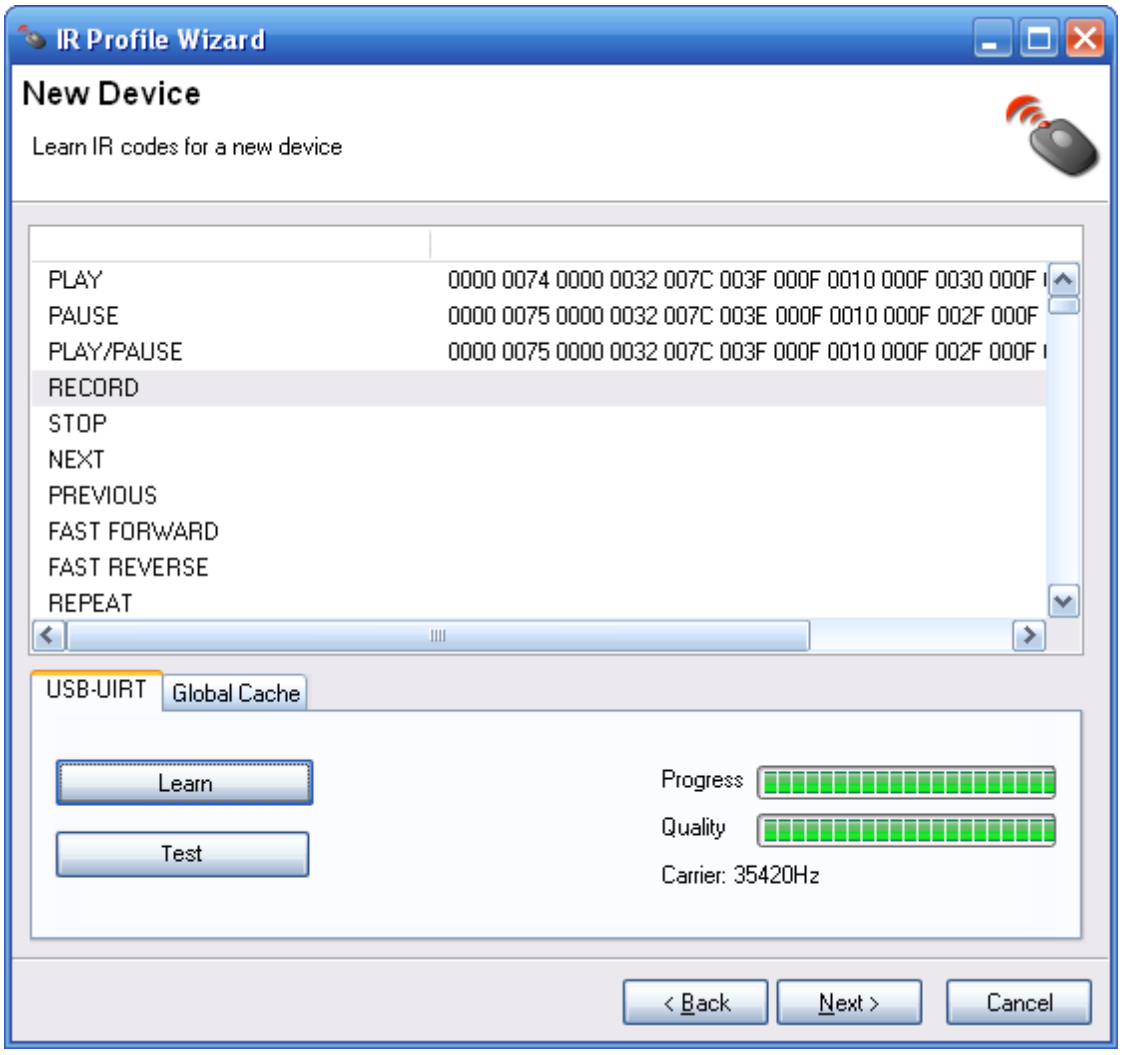
Predefined profile available.

If a predefined profile is available we will have to tell it what IR device we want to use to send out the IR signals, the name and location of the device. The latter two combined have to be unique in the system as they will be used to identify the device. Use descriptions like "Huge Plasma TV" and "Living Room" here. Simply finish the wizard from here to get your new device.

Predefined profile not available.

If a predefined profile is not available we'll have to learn the IR codes for the device. The wizard helps us here too. First we'll have to specify what device type we are trying to setup. This will be used later on by the graphical user interface to create an appropriate view of this device. The next step is to actually teach all the IR commands, currently only the USB-UIRT is supported but more device are on the way. Simply click on the button (PLAY, PAUSE, etc) of choice, press "Learn" and press the button on the remote about 1-2 inches away from the USB-UIRT. Once done the code will appear in the list. After you have enter all the buttons you wish to use press next and configure the IR output device, the name and location.

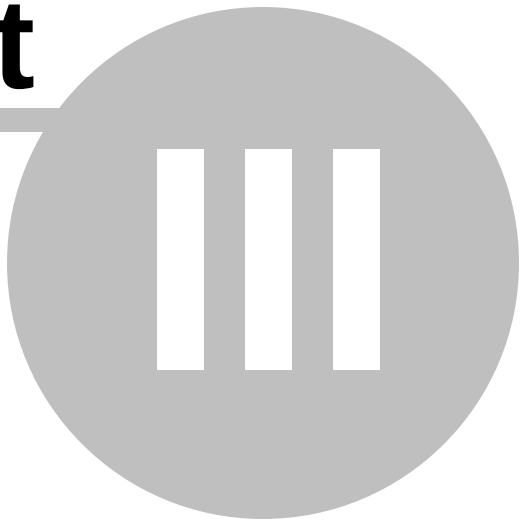
© 2008 Promixis, LLC



A great way to start using this functionality is by loading the iPhone or webbrowser Device Manager interface.

© 2008 Promixis, LLC

Part



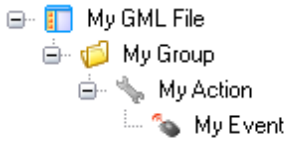
3 Girder Concepts

This section describes the concepts used by Girder and provides detailed procedures for working with these concepts to create automation applications.

- The Girder Tree.
- Girder GML Files.
- Girder Groups.
- Girder Actions
- Girder Conditionals.
- Girder Events.
- Event Mapping.
- Girder Macros.
- Girder Macro Events.
- Girder Scripts.
- Girder Plugins.

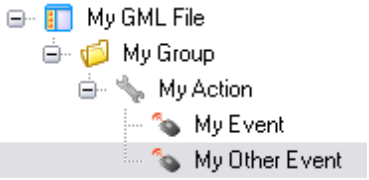
3.1 The Girder Tree

Girder uses a tree structure to associate Girder Actions with Girder Events. The simplest case looks like this -



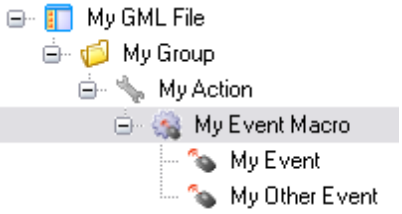
When Girder sees My Event, it executes My Action.

You can associate more than one Event with an Action.



When Girder sees *either* My Event *or* My Other Event, it executes My Action.

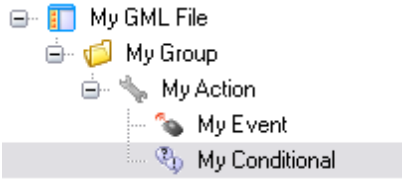
Alternatively, you can associate a Girder Macro Event with an Action and then associate the Events with the Macro Event.



© 2008 Promixis, LLC

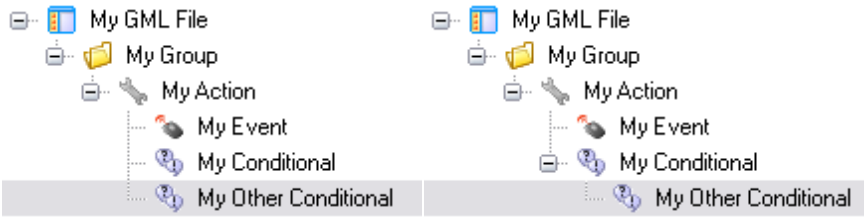
Now Girder will only execute My Action if it sees *both* My Event *and* My Other Event *in that order*.

You can make the execution of an Action conditional on some test like the presence of a Window by attaching a Girder Conditional to the Action.



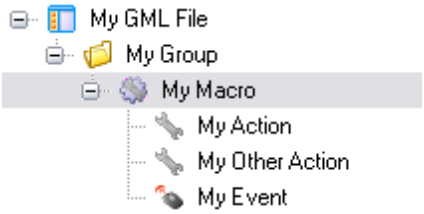
Now Girder will execute My Action when it sees My Event, but *only if* My Conditional is true.

You can attach a Conditional to any type of node in the Girder Tree to disable or enable the node and all nodes under it. A **list** or **ladder** of conditionals combine with **OR** logic while a **cascade** or **staircase** of conditionals combine with **AND**.



In the case on the left, My Action will be enabled if *either* My Conditional *or* My Other Conditional is true. On the right, My Action will be enabled if *both* My Conditional *and* My Other Conditional are true. Examples of more complex Conditional structures are given in the Girder Conditionals topic.

If you want Girder to execute a sequence of Actions in response to an Event, associate the Actions with a Girder Macro.



Now when Girder sees My Event it will *first* execute My Action and *then* My Other Action.

All the above strategies for attaching Events, Event Macros, and Conditionals to Actions also work in just the same way for Macros.

A Girder Script is a special type of action which runs a sequence of instructions written in the Lua programming language when it is executed (see Lua Language Reference).

© 2008 Promixis, LLC

3.2 Girder GML Files

Girder stores parts of its tree on disk in files with the extension .GML. Girder can have as many of these files as you want open and active. Nodes in the Girder Tree which represent GML files have no parent node.

GML File Procedures

- To create a new GML file and insert it in the tree, use New on the file menu, the Create New File toolbar button or the Ctrl-N shortcut key. Supply the path and filename when prompted and press OK. You can then click on the name in the tree to edit it. The name of the node does not have to be the same as the name of the file.
- To remove a file from the tree, leaving it on your disk, highlight its node and use Close on the file menu or context menu or Ctrl-F4.
- To save changes to a file without closing it, highlight its node and use Save on the context menu or Ctrl-S.
- To open an existing file and insert it in the tree, use Open on the File menu, the Open GML File toolbar button or Ctrl-O. Then navigate to the file using the file picker and press OK.
- To download a Program Definition GML file from the Internet and insert it in the tree, use Software Updates on the Tools menu or F10. Select the appropriate tab and then double-click the application you want.
- To save part of an open GML file as a new GML file, highlight a Group node and use Export on the File menu.
- To insert a GML file as a Group within an open GML file, highlight the parent node in the tree and use Import on the File menu.

3.3 Girder Groups

Girder Groups are a way of adding structure to a complex tree enabling you to hide and show specific parts of the tree. They also enable parts of the tree to be enabled and disabled as a unit either manually from the General task box or automatically using Girder Conditionals. Groups may be nested within groups to any depth.

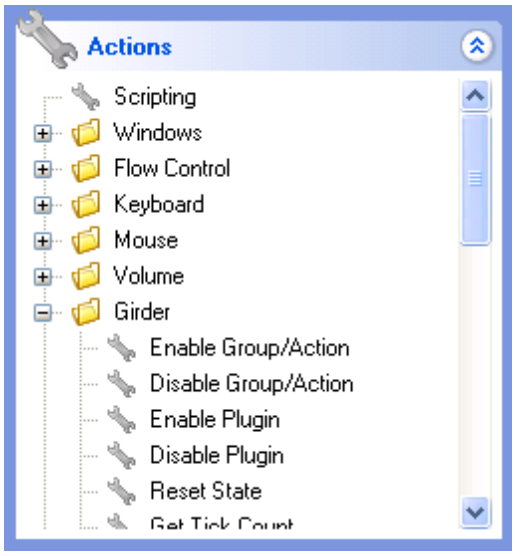
Group Procedures

- To add a group, first highlight the parent File or Group. Press the Add Group toolbar button, Shift-F1 or use the Add Group menu item on the Edit menu or the context menu. Click on the new group name and edit it to something meaningful.
- To delete a group (and all its children), use Delete on the context menu or the Edit menu or press Ctrl-Del.
- To enable or disable a Group, highlight it and use the Enabled checkbox on the General task box.
- To add documentation comments to a Group, highlight it and type the comments in the large text box at the bottom of the General task box.

3.4 Girder Actions

Girder Actions carry out specific functions on your computer. A wide variety of actions are provided as standard and more may be added using Girder Plugins or the Tree Script Plugin. The available actions are listed on the Actions task box.

© 2008 Promixis, LLC



There are two ways of adding an Action to the tree. You can highlight the parent node (which must be either a Group or a Macro) and double-click on the required Action type in the Actions task box. Alternatively, you can drag the Action's icon from the Actions task box into position in the tree.

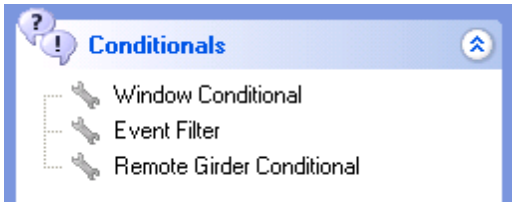
In either case, the Action Editor will appear to allow you to configure the Action. The editor is different for each Action type and is described in the Actions Reference topic.

Action Procedures

- To edit an action, either double-click on it or use Edit or Edit in New Window on the context menu.
- To test the effect of an Action, highlight it and press the Test Action toolbar button or F5.
- To delete an action (and any attached Events, Event Macros or Conditionals), use Delete on the context menu or the Edit menu or press Ctrl-Del.

3.5 Girder Conditionals

Girder Conditionals control whether the parent node is enabled or not based on a specific test. See the Conditionals Reference for details of Conditionals provided as standard. More may be added using Girder Plugins or the Tree Script Plugin. The available Conditionals are listed on the Conditionals task box.



There are two ways of adding a Conditional to the tree. You can highlight the parent node and double-click on the required Conditional type in the Conditionals task box. Alternatively, you can drag the Conditional's icon from the Conditional task box onto a node in the tree.

In either case, the Conditional Editor will appear to allow you to configure the Conditional. The

© 2008 Promixis, LLC

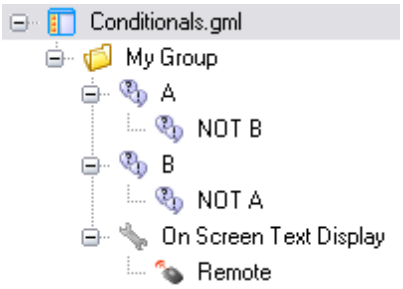
editor is different for each Conditional type and is described in the Conditionals Reference topic.

Function of Conditionals in the Girder Tree

A Conditional on a **File Node** or **Group Node** enables or disables all the Actions within and below that node. A Conditional on an **Action Node** or **Macro Node** prevents or allows that node to be triggered by events. A Conditional on an **Event Node** or **Macro Event Node** prevents or allows triggering of Actions by that node. A Conditional on another **Conditional** creates a logical **AND** combination such that *both* Conditionals must be true to enable the node to which the first is attached. However two or more Conditionals on the same node (including on another Conditional) creates an **OR** combination such that the node is enabled if any of the attached Conditionals is true. All Conditionals have an Invert or **NOT** option on their editor. These three operations: AND, OR and NOT, are a complete set which allows any logical combination to be created.

Example:

Suppose you want to enable the actions within a Group when one of two alternate windows is present, and disable them when neither is present and when both are present. In formal logic this operation is called "exclusive or" or XOR. It can be expressed as **(A AND NOT(B))OR(B AND NOT(A))**, which can be represented in the Girder Tree like this.



Tip: Remember that a cascade or staircase of conditionals combine with AND logic, while a list or ladder combine with OR logic.

Conditionals Procedures

- To edit a conditional, either double-click on it or use Edit or Edit in New Window on the context menu.
- To delete a conditional, use Delete on the context menu or the Edit menu or press Ctrl-Del.

3.6 Girder Events

Girder Events are generated by Girder itself or by Girder Plugins in response to control actions such as a keystroke on the keyboard or remote controller or a range of other happenings on the computer. The available Events are documented in the Events Reference and the Plugins Reference.

For each event that occurs Girder records the following information.

1. The device number of the Plugin that generated the event (EventDevice in Lua).
2. An Event String which distinguishes between different events generated by the same Plugin (EventString in Lua).
3. A modifier applicable to Keyboards and Button devices, which is one of Down, Repeat or Up (or may be absent).

© 2008 Promixis, LLC

4. The time of the event to millisecond resolution.
5. Up to four additional string parameters which are only available as Lua variables pld1..pld4.

Event nodes in the tree can recognize unique values of 1 and 2 plus any set combination of 3. They can alternatively be set to recognize on 1 only (any message from a specific device).

Note: There is a possible problem of language here. "Event" can refer either to the *occurrence* or to the *node* in the Girder Tree that recognizes the occurrence and triggers Girder Actions. If there is any ambiguity, the term "Event Node" will be used to refer to the recognizer.

Event Procedures

- To add an Event Node using the Logger, see the Logger topic in the User Interface Guide section.
- To add an Event Node using the Toolbar, first highlight the parent node in the Girder Tree, then press the Add Event button. See the Event Editor topic in the User Interface Guide section to complete this procedure.

3.7 Event Mapping

Events generated by devices like the keyboard or a remote controller are referred to as **Raw Events**. Event Nodes may be set to recognize these directly, but that means that the GML file will only work with that specific control device.

Event Mapping in Girder introduces the concept of the **Mapping Device**. Usually each remote will have its own Mapping Device which generates Events drawn from a set of **Predefined Events** in response to the Raw Events from the remote itself. This means that, for example, any remote having an up arrow button will produce **Event String** "ARROW UP" when it is pressed.

If you have two remotes, the "ARROW UP" Event will still be distinguishable by **Event Device** because each remote will have its own Mapping Device. This means you can have the two remotes control different applications, or you can set the Event Nodes to recognize **Predefined Events (all remotes)**.

Girder provides a Remote Assignment Editor for making bulk changes to the Event Device of all Event Nodes in a GML file or in a Group. This enables you to decide which of your remotes to use with standard Application Definition GML files.

Supported remotes are provided with a Mapping Device, but you may want to change the key assignments or define a new Mapping Device from scratch. The Event Mapping Editor allows you to do this.

3.8 Girder Macros

Girder Macros are containers for Girder Actions that allow a sequence of Actions to be treated as a single Action from the viewpoint of triggering, enabling etc.

Macro Procedures

- To insert a Macro in the tree, highlight the parent Group and press the Add Macro button on the toolbar, or press Shift-F3, or use Add Macro on the context menu or the Edit menu. Click on the Macro name and edit it to something meaningful.
- To delete a Macro and all its child nodes, use Delete on the context menu or the Edit menu or press Ctrl-Del.
- To test a Macro, highlight it and press the Test Action toolbar button or F5.
- To enable or disable a Macro, highlight it and tick or un-tick Enabled in the General task box.

© 2008 Promixis, LLC

- To document the purpose of a Macro, highlight it and enter text into the large textbox at the bottom of the General task box.

3.9 Girder Macro Events

Girder Macro Events are containers for Events that enable a sequence of Events to be recognized and to trigger an Action or a Macro in the same way as a single Event.

Note: Each Event in the Macro Event sequence must occur within five seconds of the previous one. Otherwise the entire Macro Event resets to look for the first Event again.

Macro Event Procedures

- To insert an Macro Event in the tree, highlight the parent Action or Macro and press the Add Macro Event button on the toolbar, or press Shift-F6, or use Add Macro Event on the context menu or the Edit menu. Click on the Macro Event name and edit it to something meaningful.
- To delete a Macro Event and all its child Events, use Delete on the context menu or the Edit menu or press Ctrl-Del.

3.10 Girder Scripts

Girder uses the Lua scripting language to provide virtually unlimited power once the capabilities of the available Actions runs into limitations. The Lua Language Reference describes this language.

Scripting Procedures

- To insert a Scripting Action, drag it from the Actions task box to the Girder Tree. Enter the script in the Action Editor.

3.11 Girder Plugins

Girder can be extended using Girder Plugins which are Windows DLL files written to conform to the Girder API. Many Plugins are shipped with Girder and more can be downloaded from www.promixis.com.

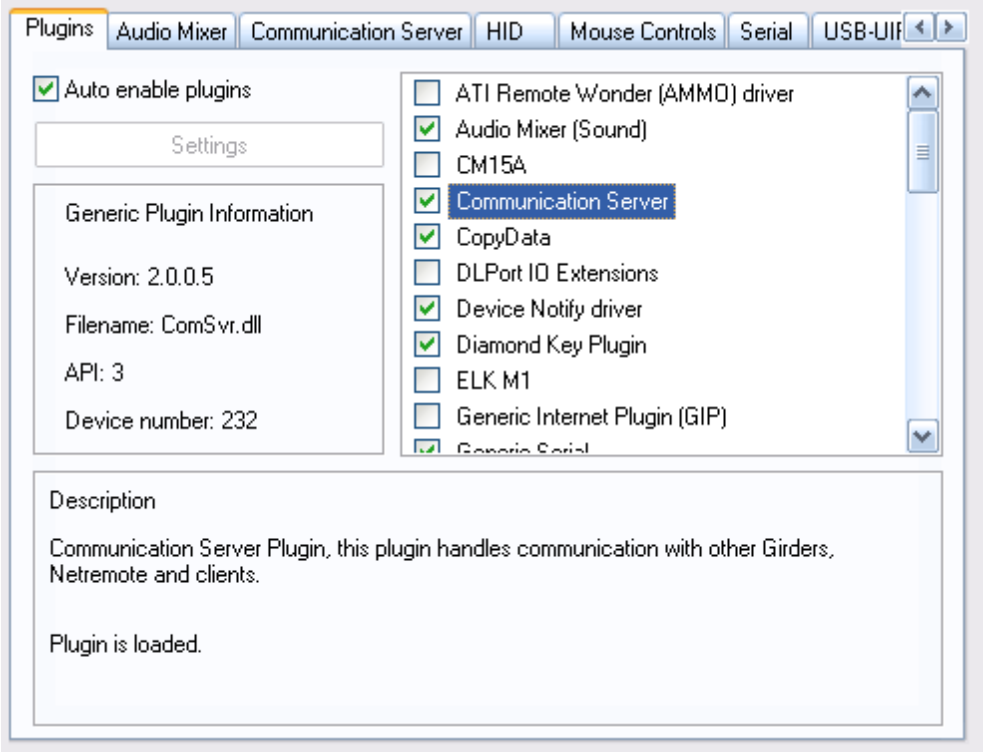
A plugin can provide any combination of the following -

- Generation of Events and provision of Event pick-lists.
- Additional Actions with Action Editors.
- Additional Conditionals with Conditional Editors.
- Configuration Pages.
- Lua Extensions.

Details of the Plugins provided with Girder are in the Plugins Reference section.

If a Plugin DLL file is located in the **%GIRDER%\plugins** directory when Girder is started, it will be listed on the Plugins configuration tab. Press the **Preferences** toolbar button or use the **Settings** item on the File menu, and then click the **Plugins** icon.

© 2008 Promixis, LLC



To enable a Plugin, check the tickbox and press **Apply**. If the Plugin provides a configuration page it will appear as a tab on this dialog. Some older Plugins publish a separate configuration dialog and if this is the case, the **Settings** button will be available when the Plugin is selected.

Plugin Procedures

- To install a downloaded Plugin, first extract the file name.dll and any files with the extension .xml from the distribution archive or zip file. Move the .dll file to the %GIRDER%\plugins\ directory and the xml files to the %GIRDER%\plugins\UI\ directory. (Some Plugins provide an installation program which does this for you.) Then exit from Girder by right-clicking the notification area icon and using Exit Girder. Lastly re-start Girder using the Start menu. The new Plugin will now appear on the list in the Preferences dialog.
- To enable or disable a Plugin, tick or un-tick it in the list of Plugins on the Preferences dialog (see above).
- To configure a Plugin, select its tab on the Settings dialog or press the Settings button on the Plugins tab with the Plugin highlighted on the list. Note that a given Plugin will present one or the other configuration method or it may require no configuration.

© 2008 Promixis, LLC

Part

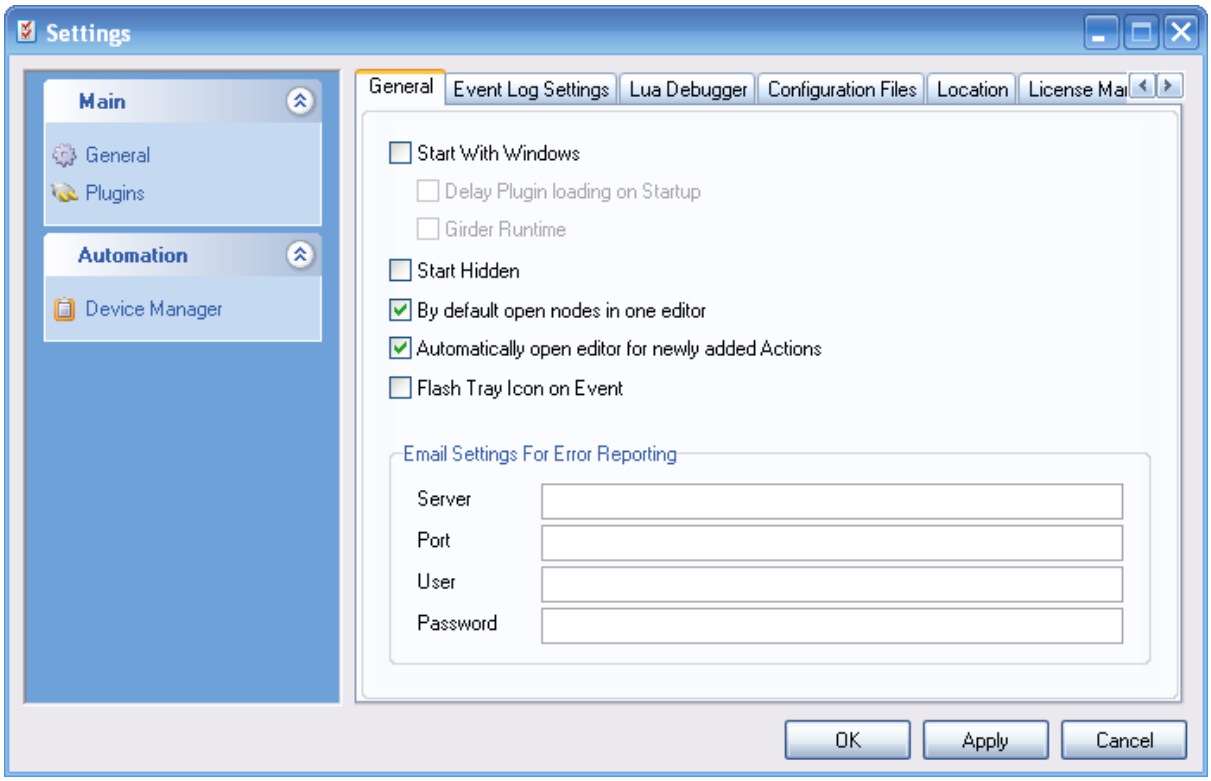
4 Settings and Applications

This section details features provided by Girder "out of the box" as well as some general settings. These features do not require scripting or Girder Tree development, although many of them can be extended or modified in this way. Not all features are available in the Standard edition. Settings pages which relate to specific device drivers are not included in this section. For help on these, see the Plugins Reference.

4.1 Settings Introduction

The Girder settings system is accessed via the **Preferences** button on the toolbar, or the **Settings** or **Components** items on the **File** menu. The settings system is extensive and the exact layout depends on product version, which Plugins are loaded and on scripting.

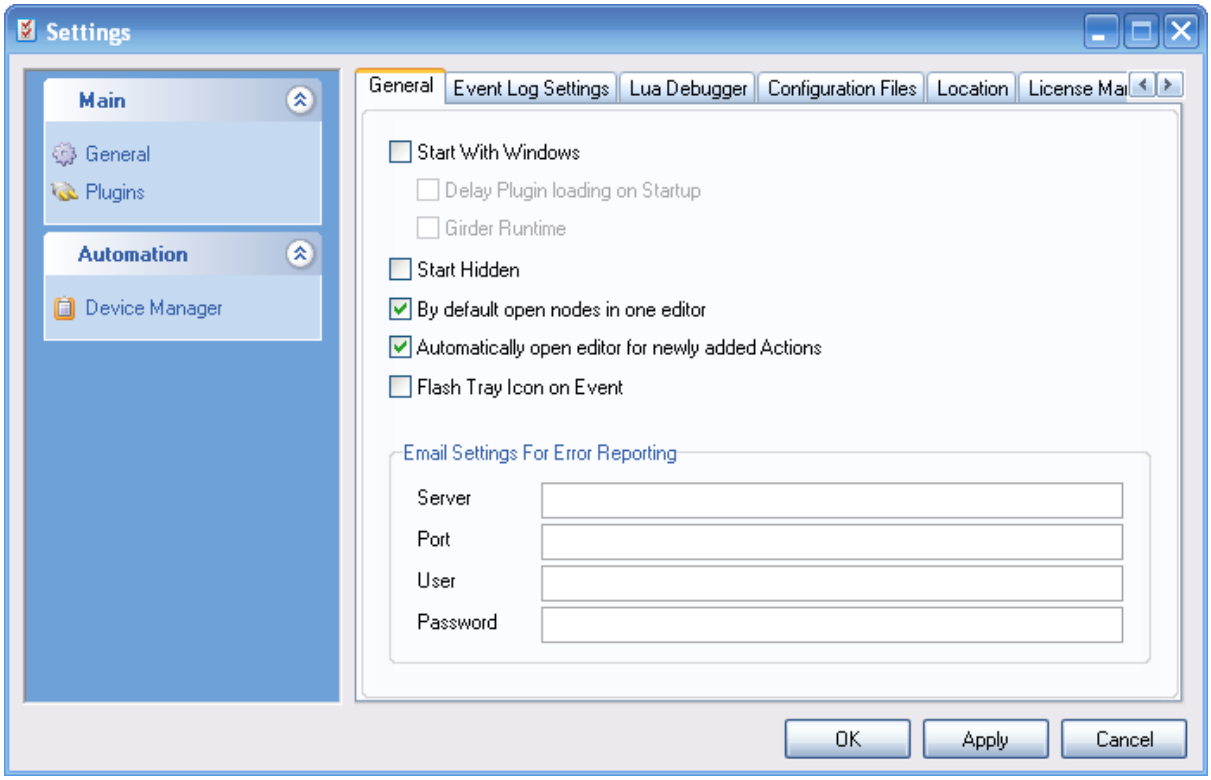
The strip on the left selects groups of settings under the **Main Settings** and **Automation Settings** headings. Each settings group contains one or more settings tabs arranged along the top of the right-hand section.



© 2008 Promixis, LLC

Settings changes entered on the tabs can be applied immediately without closing the dialog using the **Apply** button, or alternately use **OK** to apply the settings and dismiss the dialog.

4.2 General Settings



Start With Windows: Tick to have Girder start automatically when Windows boots (recommended).

Delay Plugin loading on Startup: Some Plugins may depend on other resources which are loaded at startup and so may not initialize correctly if **Start With Windows** is ticked. This option delays the startup of Plugins to allow for this. Only tick this if necessary.

Start Hidden: Normally Girder starts with its main window showing. Tick this to have Girder start minimized to a notification area icon.

By default open nodes in one editor: When ticked, double clicking a node in the tree means **Edit** (an editor window is only opened if there are none already, otherwise the node replaces the node being displayed in an existing editor window). If un-ticked double clicking a node means **Edit in New Window**, and a new editor window will be opened in addition to any already open.

Automatically open editor for newly added Actions: When ticked, an editor window opens automatically when a new node is added to the tree.

Flash Tray Icon on Event: Tick this to use an animated icon in the notification area which flashes whenever a Girder Event occurs. Un-tick if this is distracting.

Email Settings for Error Reporting

© 2008 Promixis, LLC

Specify the details of an SMTP mail server and account to which error messages will be sent. This allows for unattended monitoring of a Girder machine. Settings can be obtained from the Outgoing Server details of an email program.

Server: The URL of the mail server, typically something like "mailhost.zen.co.uk".

Port: The standard port for an SMTP server is 25.

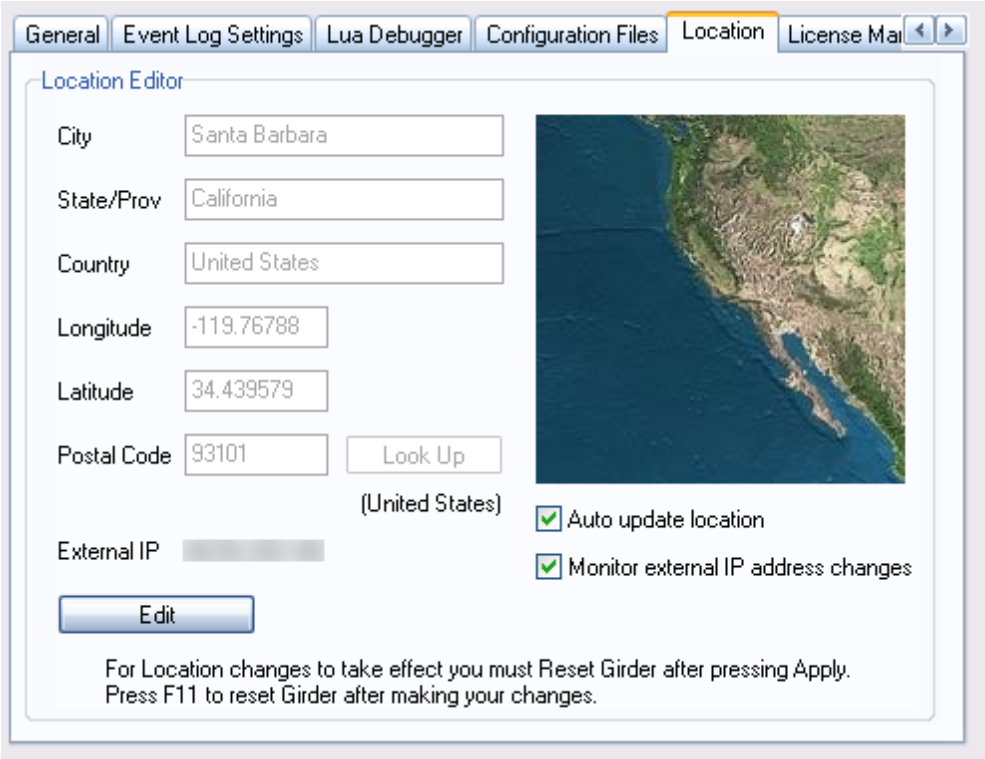
User: If the server requires a login, enter the User Name.

Password: If the server requires a login, enter the Password.

See Error Handling in the **User Interface Guide** for details of how to specify a destination mail address and message.

4.3 Location Settings

This records the location of the computer as a place name and as Latitude and Longitude. These settings are used by all built-in applications which require this information, such as the Weather Application and Sunrise and Sunset Scheduler tasks.



Within the US, simply click edit and enter your Zipcode and press **Lookup** to fill in all the fields if the automatic detection didn't get it right.

Outside the US, **Lookup your coordinates** links to a gazetteer web site on which you can look up the Latitude and Longitude of your nearest town. Alternatly, Google Earth http://earth.google.com/ is an excellent and fun way to pinpoint your location exactly.

Tip: When using Google Earth, select **Tools/Options/View/Rendering** and check the "Degrees" radio button next to "Lat/Lon:". The Pointer lat and lon values shown below the map now give values that can be typed directly into Girder.

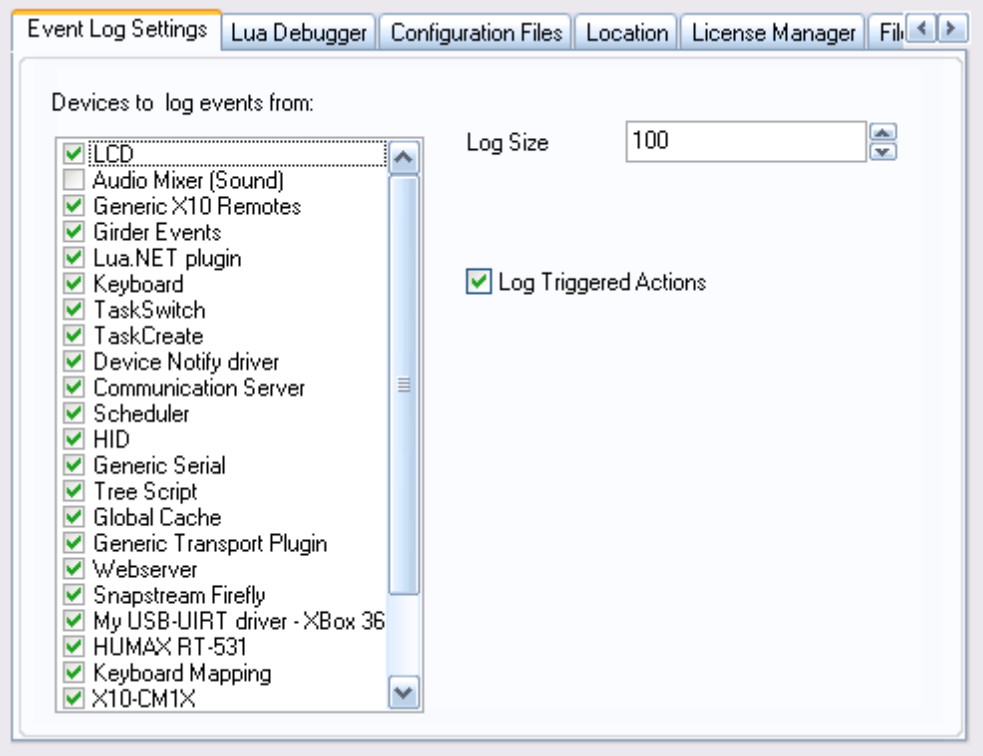
© 2008 Promixis, LLC

Latitude is entered as decimal degrees -90.0 at the south pole, 0.0 at the equator, 90.0 at the north pole. Longitude is entered as decimal degrees 0.0 at the prime meridian through Western Europe and West Africa, running negative through the western hemisphere to -180.0 in the middle of the Pacific ocean and positive through the eastern hemisphere to the same line.

Tip: The minus sign **must** be entered in front of longitudes in the Americas - it is sometimes omitted in reference sources.

Temperature Units can also be specified on this page as Celcius or Fahrenheit.

4.4 Logger Settings



These settings control which messages, and how many, appear in the Logger.

Log Size: Maximum number of events remembered for scrolling back in the log.

Log Low Level Scripting Debug Messages: Provides additional detail which may be useful when debugging Lua scripts.

Log Triggered Actions: Normally only Events are logged, but if this is ticked, messages are also logged each time an Action is triggered (run).

Devices to log events from: This section allows messages from selected Plugins to be filtered out of the log. This is useful if unwanted messages are making it difficult to see wanted ones. For example, the Keyboard Plugin generates several messages each time you press a key.

© 2008 Promixis, LLC

4.5 Lua Debugger

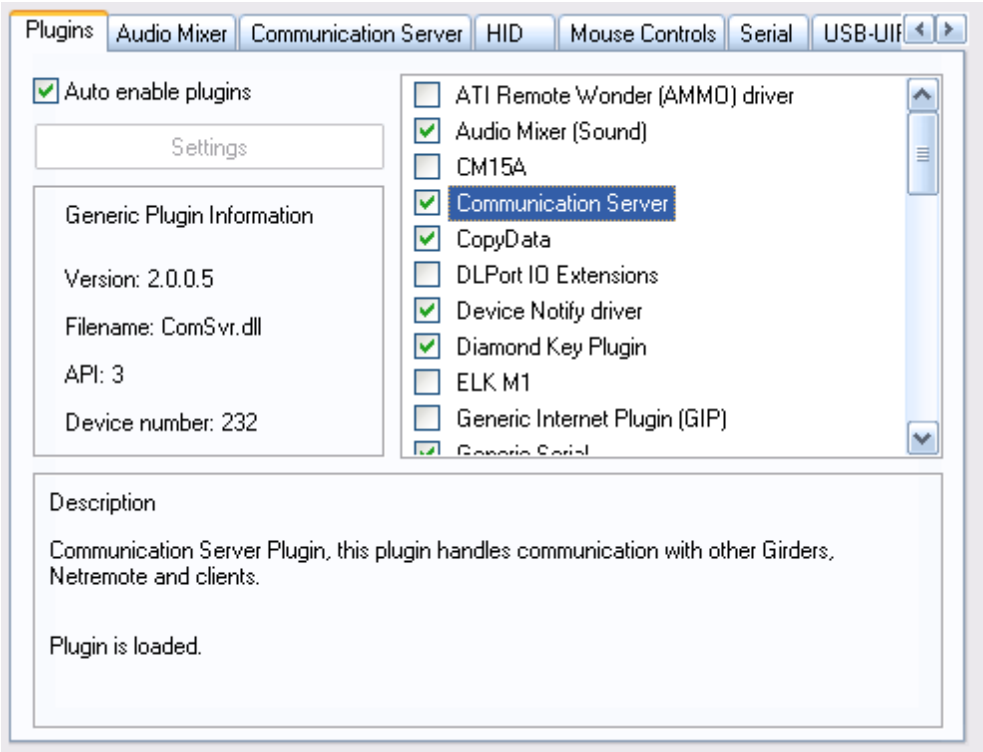
The Lua debugger are an advanced setting that should be disabled for normal operation and only be used by experienced Girder users.

The debugger allows you to step through lua code while Girder is executing it. Make sure you have saved all your files while using the debugger.

4.6 Plugins Settings

The **Plugins** tab of the **Plugins** group controls which extension features are loaded with Girder. Details of the Plugins provided with Girder are given in the Plugins Reference section of the reference manual. The **Plugins** settings tab allows individual Plugins to be loaded and unloaded and provides version and other information.

Tip: The term **loaded** for Plugins means that the user interface elements (settings tabs, Action and Conditional editors) are available. The term **enabled** means that the Plugin will generate Girder Events as appropriate. Plugins can be loaded and unloaded individually, and they can only be enabled if they are loaded. The loaded Plugins can only be enabled and disabled as a group, not individually.



The list box top right shows all the available Plugins. Click on the text to select a Plugin and display information about it. Click on the tick box to the left of the text to load or unload the Plugin.

Auto enable plugins: If this is ticked, all plugins that are selected (ticked) will start generating events as soon as Girder starts. If this is not ticked, event generation needs to be manually started using the **Enable Events** button on the toolbar or notification icon menu.

Settings button: Normally, Plugins provide additional tabs containing any configuration settings they need. However some older ones present their settings in stand-alone dialogs. If a

© 2008 Promixis, LLC

settings dialog is available for the selected Plugin, this button will be enabled and can be pressed to bring it up.

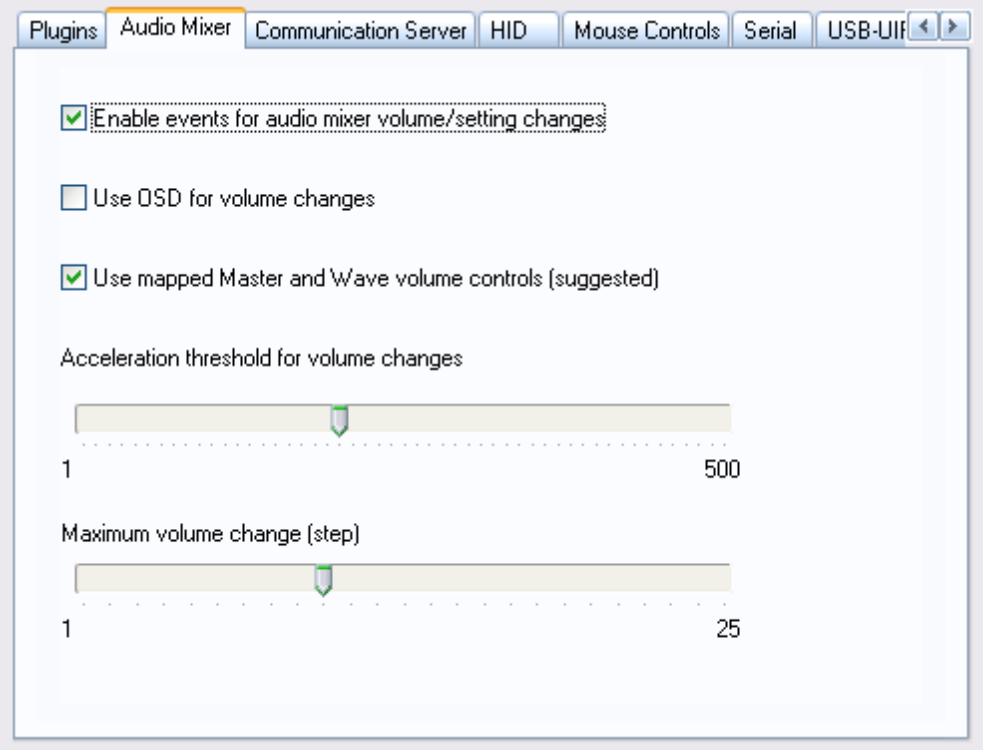
4.7 License Manager

Some plugins for Girder need a separate license if that is the case this dialog is used to keep track of the licenses. A license consist of 2 lines, the name and a whole line of letters with a dash in the middle. To enter the license copy and paste BOTH lines from the confirmation email into the license box. If you are experiencing issues here please contact sales@promixis.com for further assistance.

```
John Doe
4ECAAASBAAAAAAA-VWXI9ZHIU96SXNNKVKGEXKF3TXMDPVTUUK7SUX86QEQX8
```

4.8 Audio Mixer

The Audio Mixer settings tab is available if the Audio Mixer (Sound) Plugin is loaded. It accesses some built-in audio control features for the master volume and mute of the computer. Other audio controls are available through Actions and Lua.



Enable events: If ticked, Girder Events are generated whenever the audio settings change for whatever reason (for example, via controls in a media player). These Events can be used to trigger Actions in the Girder Tree.

Use OSD: If ticked an On Screen Display is shown briefly whenever the master volume or mute setting changes.

Use mapped Master and Wave volume controls: If this is ticked, most standard remote control Event Maps will provide control of the master volume and mute (and possibly also the Wave

© 2008 Promixis, LLC

mixer input channel used for audio from most PC media players) without further configuration.

Event Mapping

The following Output Events in the Event Mapping Editor work audio controls automatically if the **Use mapped...** setting is ticked.

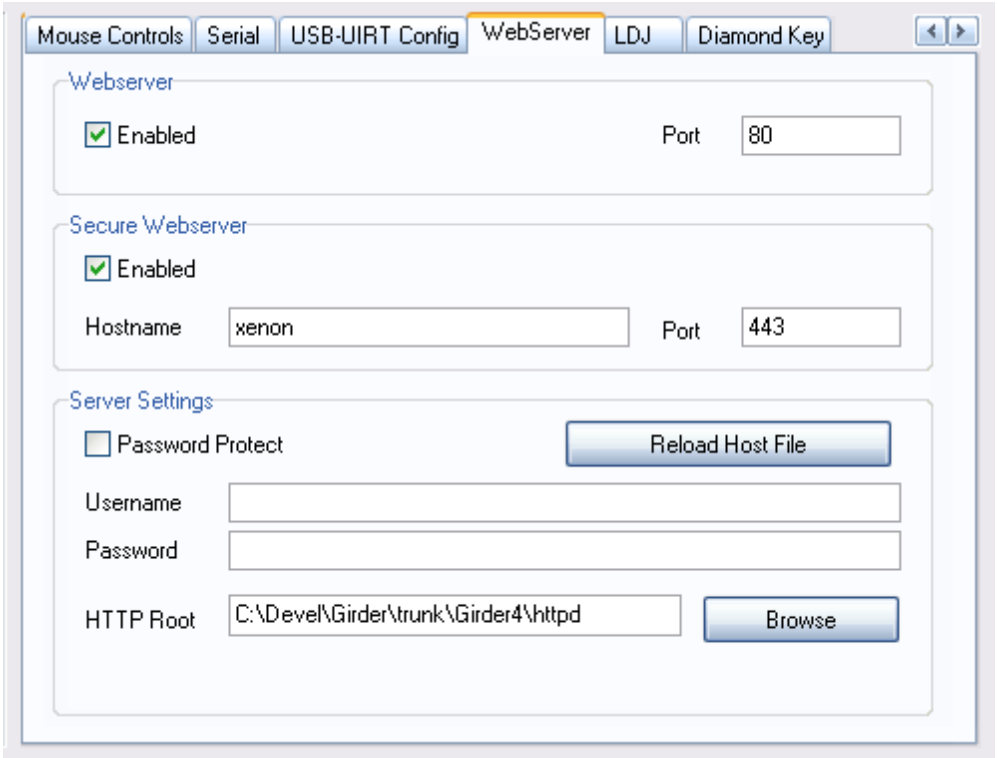
- MASTER VOLUME UP
- MASTER VOLUME DOWN
- MASTER MUTE
- WAVE VOLUME UP
- WAVE VOLUME DOWN
- WAVE MUTE

4.9 Webserver

Requires Girder Pro.

The Web Server Plugin provides access to Girder from the net using any Web Browser. You can view the example web pages shipped with Girder by pointing your browser at **http://localhost/**. From a different machine on your local network, substitute the machine name or IP address for **localhost**. Access from anywhere on the internet can be provided with a little effort. It requires a globally visible IP address and/or DNS name and the firewall, if any, will need to be configured to allow the Girder machine to act as a web server.

You can edit the sample web pages provided, or create your own from scratch. These pages can interact with Girder in complex ways and full details are provided in the Web Programming Reference section of the reference manual.



HTTP Root: This specifies the directory that will contain the public web page files. The symbol %

© 2008 Promixis, LLC

GIRDER% may be used to provide the path to the Girder installation directory. The default is % GIRDER%\httpd. The URL given above accesses the file "index.html" in this directory.

Port Number: Web browsers use port 80 for open web requests (http) or 443 for secure web requests (https). Use of different port numbers allows multiple web servers to run on the same machine or can provide greater security when used with a firewall.

Password Protect: If this box is ticked, the Web Browser will prompt for a user name and password and access will only be granted if the supplied values match those below.

Username: The user name for authentication.

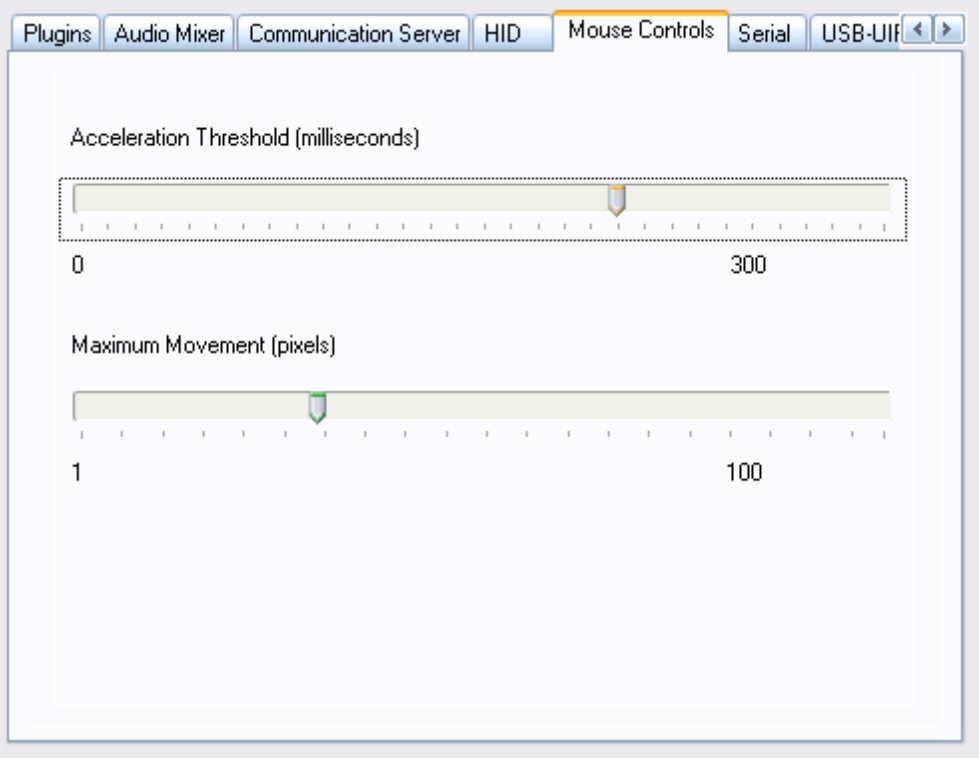
Password: The password for authentication.

Secure Webserver: Tick this box to use secure (encrypted) transmission. Use a URL of the form **https://localhost/**. See the reference section Web Server Plugin for more information on setting up secure transmission.

Hostname: This must match the hostname used to access the site in secure mode (for local network access it would be the machine name).

4.10 Mouse Controls

The Mouse Control Plugin allows the mouse pointer on the computer to be moved around using buttons on a remote (normally the arrow keys or a positional "hat"). The **Mouse Controls** tab provides settings to adjust the dynamics.



Acceleration Threshold: The mouse pointer initially moves one pixel for each **down** or **repeat** button event. After this time in milliseconds, each event moves it **Maximum Movement** pixels. Movement is reset to one pixel after an **up** event or after the same time expires with no further **down** or **repeat** event.

© 2008 Promixis, LLC

Maximum Movement: The number of pixels per event after **Acceleration Threshold**.

These settings allow fine pointer control without it taking forever to move large distances across the screen. Press and hold the button for large movements and stab the button for fine control.

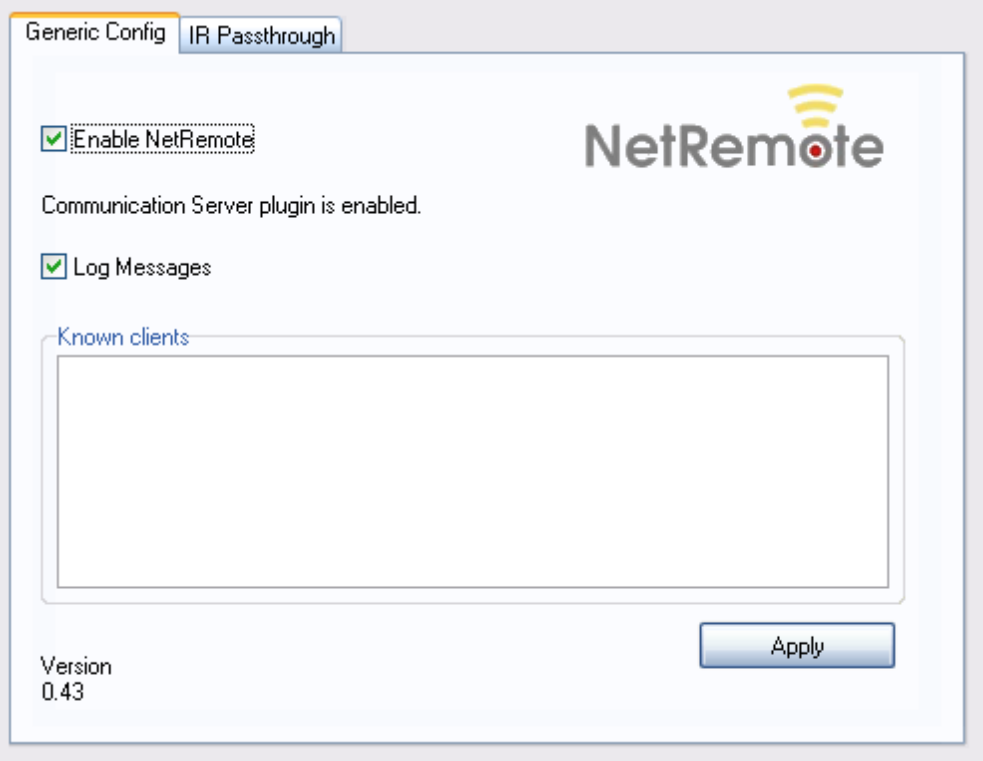
Event Mapping

The output events prefixed **MOUSE** and **MOUSEBUTTON** may be mapped using the Event Mapping Editor to remote raw events to provide mouse control. The **MOUSE MODE TOGGLE** output event allows a button to have the function of switching the directional buttons between moving the mouse pointer and generating **ARROW** events. The standard remote mappings are set up to do this automatically if there is a suitable directional controller on the remote.

4.11 NetRemote

Girder provides a number of automatic features for integrating with **Promixis NetRemote**, an advanced programmable remote control based on Pocket PC devices and WiFi. NetRemote support requires the Communication Server Plugin.

NetRemote settings are located in the Automation Settings group.



Enable NetRemote: Tick this to enable support for NetRemote clients.

Log Messages: Tick this to see diagnostic messages in the Logger.

Connect Clients: This box lists the NetRemote clients that are currently connected to Girder.

The settings in the NetRemote Girder Plugin must match those on the **Communication Server** tab. Take care changing the settings at the Girder end because Communication Server settings

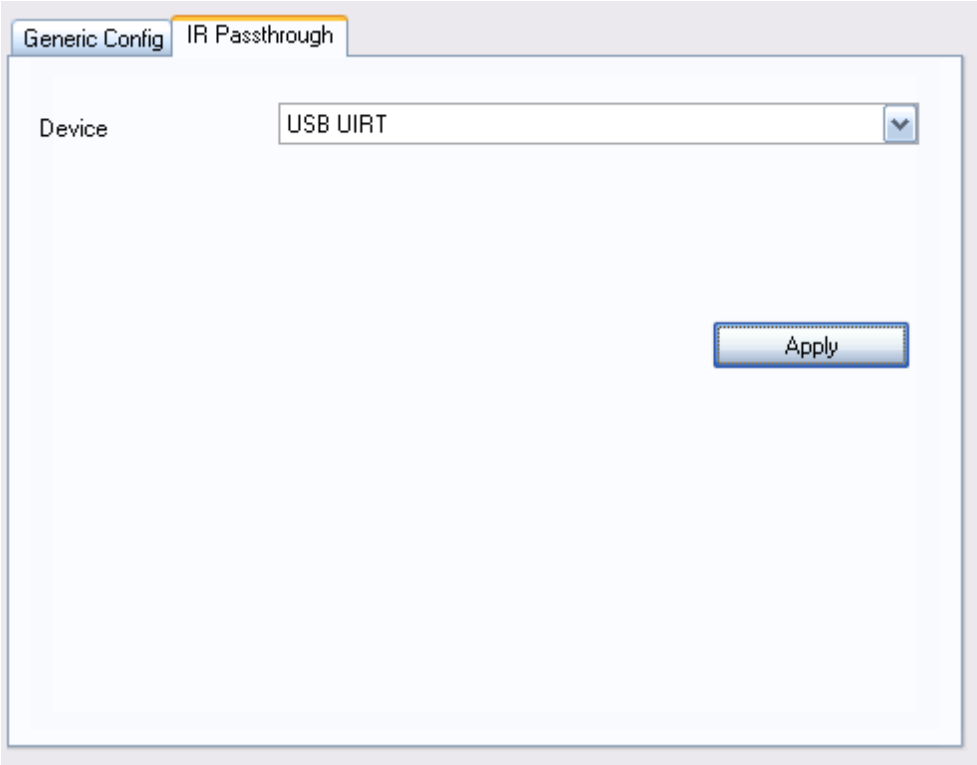
© 2008 Promixis, LLC

effect Girder to Girder networking and well as NetRemote.

IR Passthrough

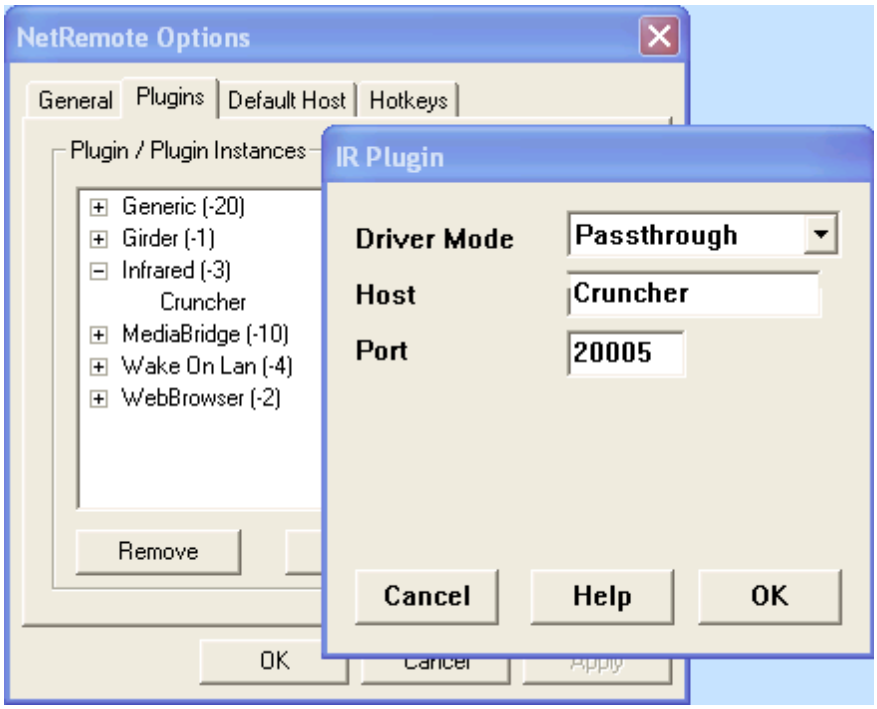
IR Passthrough allows Girder to act as a WiFi to IR bridge for NetRemote. Girder must have a supported IR transmitter device attached. GlobalCache, Nirvis SlinkE, IRTrans and USB-UIRT are supported, each via a dedicated Plugin which must be enabled on the **Plugins** tab.

Next visit **Automation Settings**, **NetRemote**, **IR Passthrough** tab. Select the appropriate **Device** from the drop-down list. Extra configuration options may appear depending on the selected device.



In the NetRemote client, select the default Infrared Plugin instance on the **Options**, **Plugins** tab and press **Properties**. Set the Driver Mode to **Passthrough**, the **Host** to the name of the Girder computer and the **Port** to the **Server Port** on the **Communication Server** tab in Girder Settings.

© 2008 Promixis, LLC



Applications

Out-of-the-box, Girder and NetRemote cooperate to provide weather information on NetRemote and to allow NetRemote to control the mouse pointer and act as a keyboard for the Girder computer.

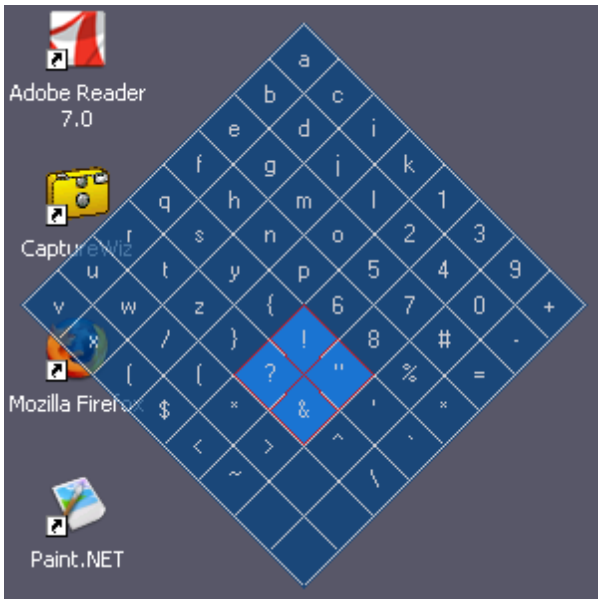
Girder can set images and labels in NetRemote and jump to a new page using Actions in the NetRemote Actions group. NetRemote Designer allows NetRemote buttons to be programmed to fire events in Girder.

4.12 Diamond Key

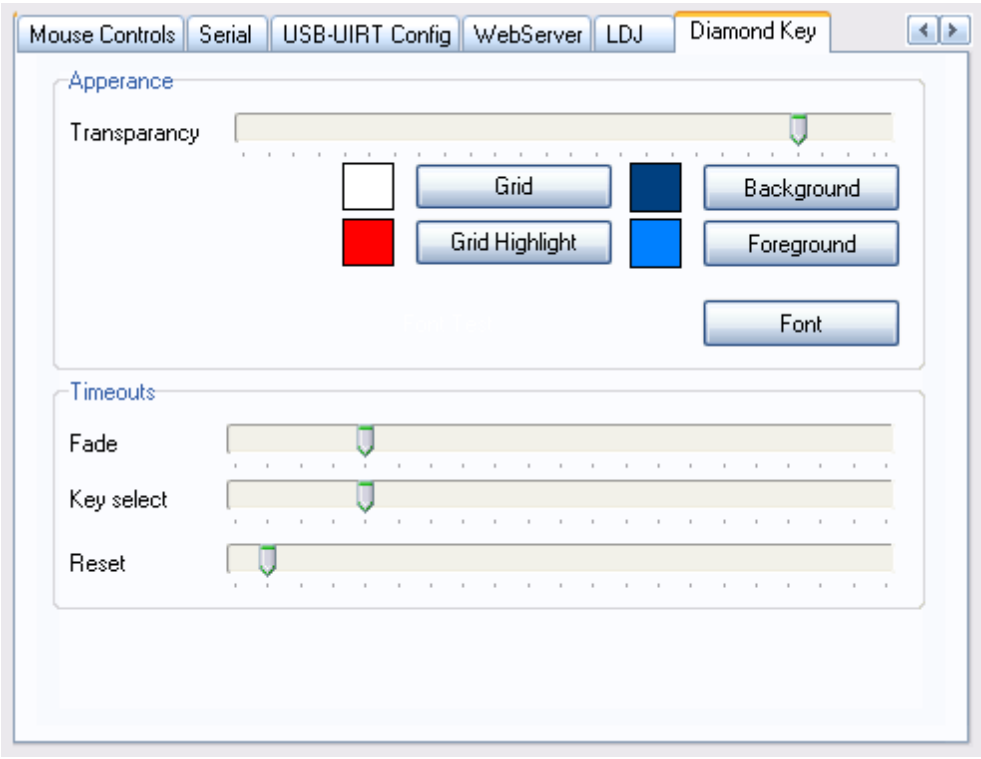
The Diamond Key Plugin implements an innovative on-screen keyboard which works using four arrow keys, typically on a remote. It is suited to short text entry on a large-screen or projector based media computer setup.

The Girder Demonstration GML file has a Diamond Key group which illustrates how to map controller buttons to bring up the diamond key OSD and control it.

© 2008 Promixis, LLC



To enter text, use the arrow keys or buttons to "narrow down" the highlighted area to a single character cell. The keyboard action in the GML file then "simulates" a keystroke and the diamond key OSD highlight expands again to the full diamond allowing the process to be repeated for the next key.



Transparency: The diamond key OSD can be translucent so that underlying windows and the desktop show through. Slide to the left to make the OSD more transparent, slide all the way to the right to make it a normal opaque window.

© 2008 Promixis, LLC

Colors: Use these buttons to re-style the OSD by changing the colors of the various elements.

Font: Use this button to alter the font and the size of characters in the OSD grid.

Fade: Adjust the time with no action before the OSD fades out.

Key Select: Adjust the time allowed between keystrokes.

Reset: Adjust the time between the final key selection and the selection resetting.

© 2008 Promixis, LLC

Part V

5 Home Automation Applications

Requires Girder Pro.

In addition to providing an excellent platform for Media PC integration, Girder also supports using a PC as a home automation controller integrating lighting, telephony, environmental control and security monitoring. Control the environment of your media room from the media PC (requires a Pro licence) or control your whole home by running Girder on all your PCs (requires Whole Home Pro). The Device manager provides a uniform interface to deal with all your automation devices, no longer does it matter if you have X10, Insteon or ZWave (TBA) they all work the same from Girder's point of view.

For the ultimate in integration, replace all your switches, control panels and remote controls with Promixis NetRemote.

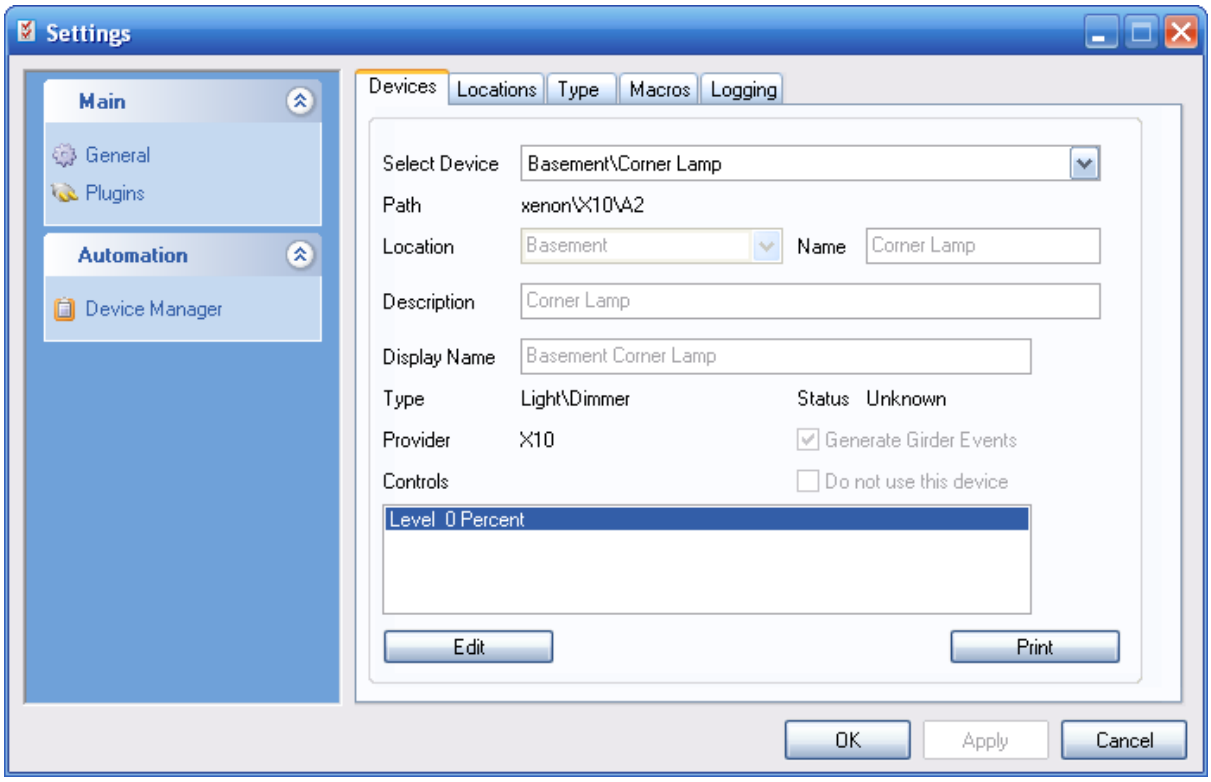
5.1 Device Manager

The Device Manager is a unified way of controlling all the various pieces of equipment in your environment. It does not matter if your device is an amplifier, Insteon or X10. As long as the driver is written to support the Device Manager it all works the same.

5.1.1 Device Manager Main Dialog

The Device Manager's main dialog can be found under file->settings.

© 2008 Promixis, LLC



The Devices tab lists all the devices in the system. You can also rename and assign a location to a device here.

The Locations tab lists all the locations in the system. You can also rename and add or delete locations here.

The Type tab allows you to retype a device, this is an advanced operation and should not be attempted normally.

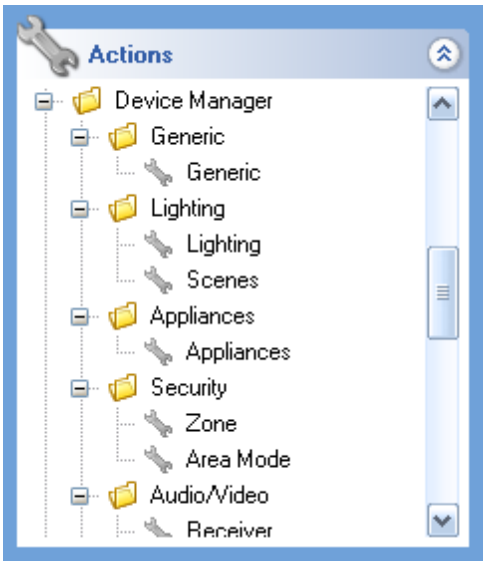
Logging sets up the logging options for all device manager related operations.

You might be wondering how to add devices to the device manager, that is done by the device source or so called "Provider" and example of a provider is the Insteon plugin.

5.1.2 Actions

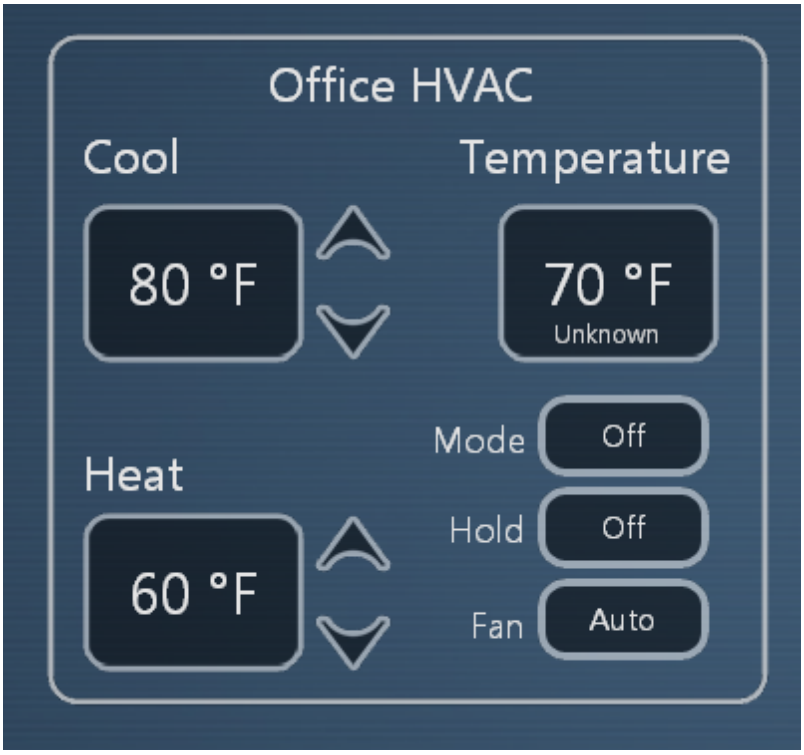
The Device Manager has several action, you can find them here.

© 2008 Promixis, LLC



5.1.3 NetRemote Device Manager Interface

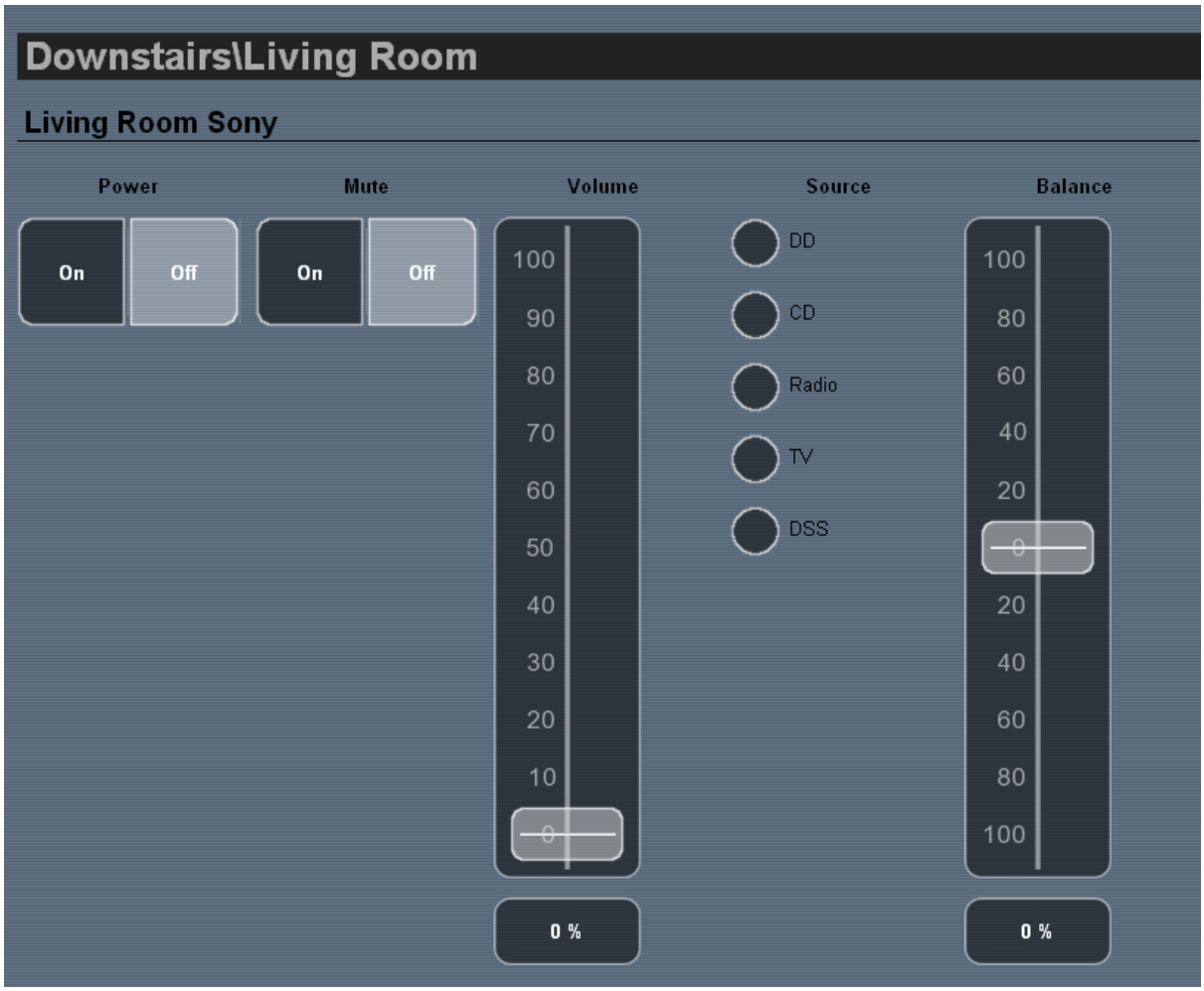
NetRemote comes with a CCF called "Flatstyle Music and DM.ccf" This CCF includes the Device Manager Functionality.



5.1.4 Webbrowser Device Manager Interface

Any modern web browser will be able to access the Device Manager interface from anywhere in the world as long as the webserver plugin are enabled and the ip address that the Girder computer is running on is reachable from the outside.

© 2008 Promixis, LLC



5.1.5 iPod Device Manager Interface

You can also control any device available in the device manager from an Apple iPhone or iPod.

© 2008 Promixis, LLC

Sorry for the bad quality it's an actual photo of the iPod. See the instruction video on how to do this here.

5.2 Scheduler

Requires Girder Pro.

Introduction

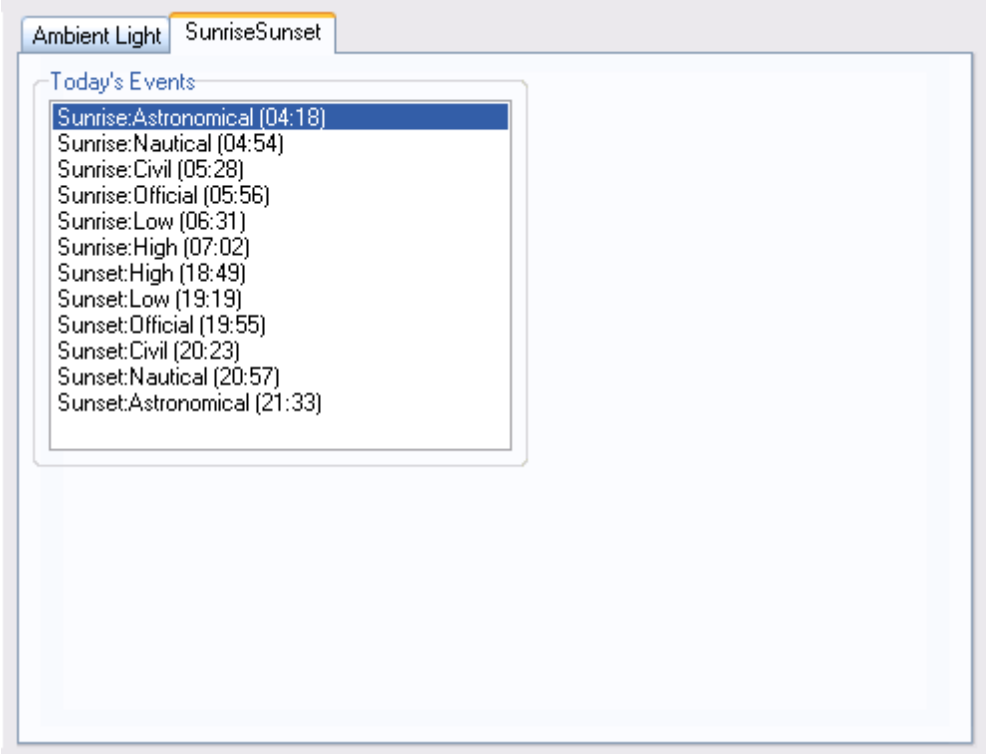
The **scheduler** allows you to create Girder Events which will be generated according to schedules based on clock time or sun time. **Light Zones** enable lighting to be controlled based on predicted or measured natural light levels.

IMPORTANT: Sun time calculations use the Latitude and Longitude from the Geographical Location tab.

Sunlight Events

Check **Enable extended Sunrise/Sunset events** on the **Sunlight Events** tab to cause Girder to generate a set of events at sunrise and sunset according to several alternate definitions published by the United States Naval Observatory http://aa.usno.navy.mil/faq/docs/RST_defs.html. The box shows the names and times of these events on the current day.

© 2008 Promixis, LLC

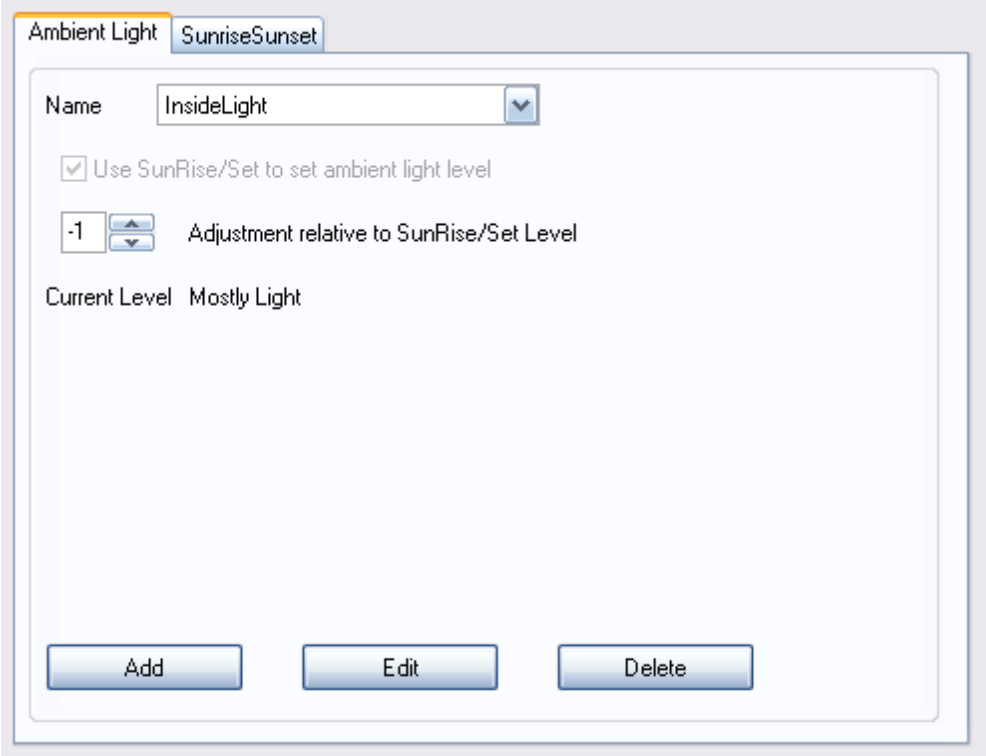


To respond to these events in the Girder Tree, select them from the **Lua: SunRiseSet** Event Device on the Event Editor.

Light Zones

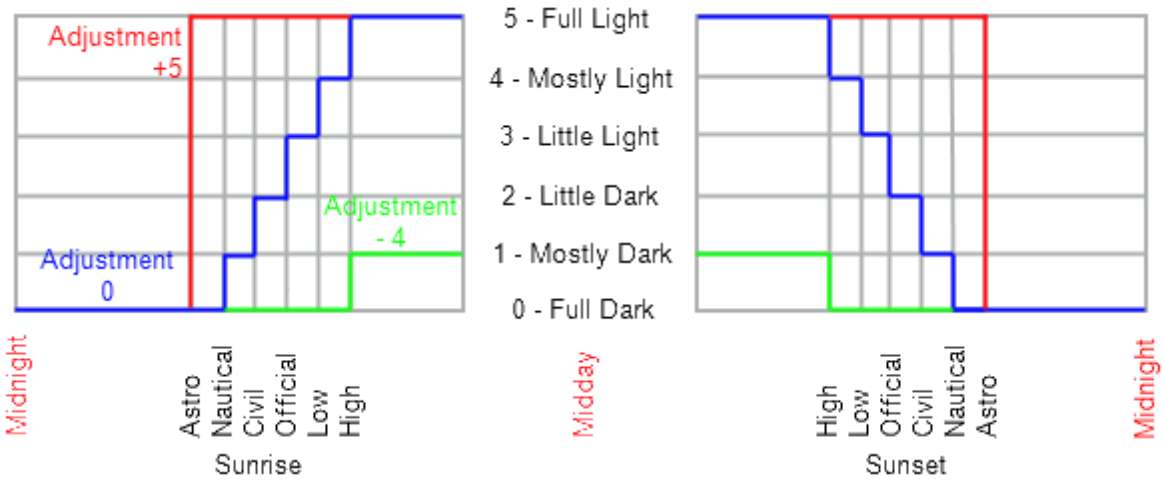
On this tab, you can create multiple, named light zones. Each zone records the actual or predicted light level on a scale of 0 (Full Dark) to 5 (Full Light). The Ambient Light Conditional makes Girder Tree Actions conditional on the level, so for example, you could have a movement detector turn on the porch light, but only if the light level in the "outside light" zone is equal or less than "Mostly Dark".

© 2008 Promixis, LLC



To create a zone, enter a name in the Area box, select **Use SunRise/set** and press OK. You can also edit the settings for an existing zone, or delete a zone by typing the zone name and selecting the corresponding button.

The following graph shows how the light zone level depends on the Sunlight Events and the **Adjustment** setting.



If **Use SunRise/set** is not ticked for a zone, the level can be controlled using the Ambient Light Level Actions, perhaps connected to events generated by a light sensor device.

Event Scheduler

© 2008 Promixis, LLC

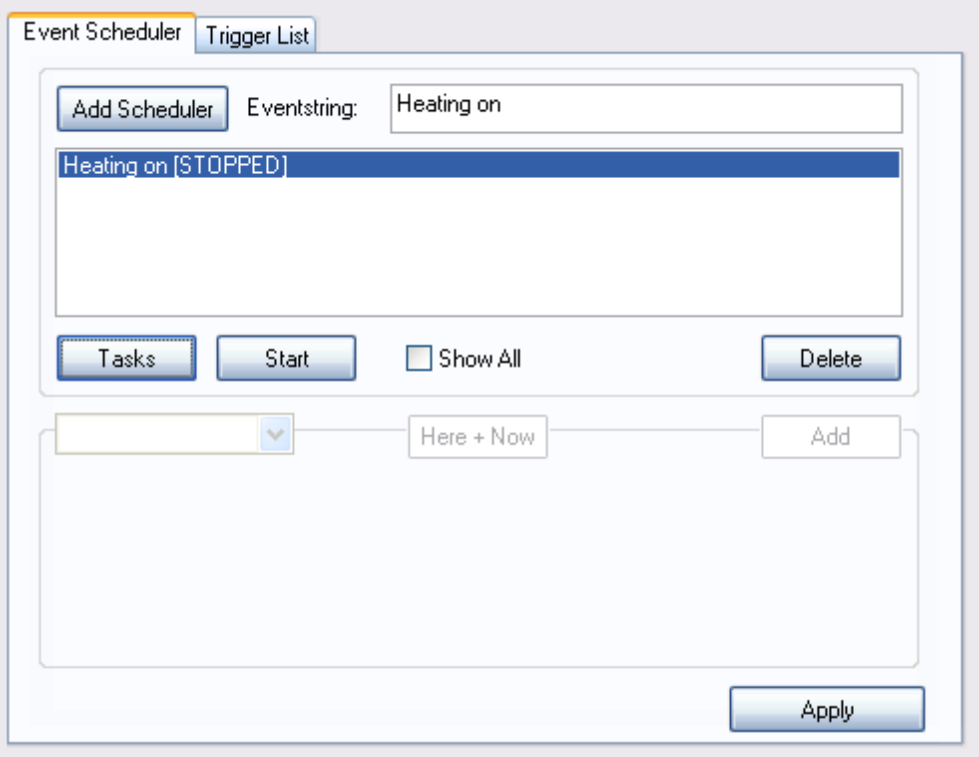
The **Event Scheduler** tab configures schedules of periodic and one-off events which can trigger Actions in the Girder Tree.

First, some definitions.

Scheduler: You can define any number of schedulers. Schedulers generated using this tab produce events which may be selected on the Event Editor using the **Lua: Scheduler** Event Device with the specified Event String.

Task: Each scheduler can have one or more tasks which define when the event gets generated. A full set of task types is available to generate events once at a specified time and date, or periodically every minute, hour, day (at a specified time or at sunset or sunrise), week or month.

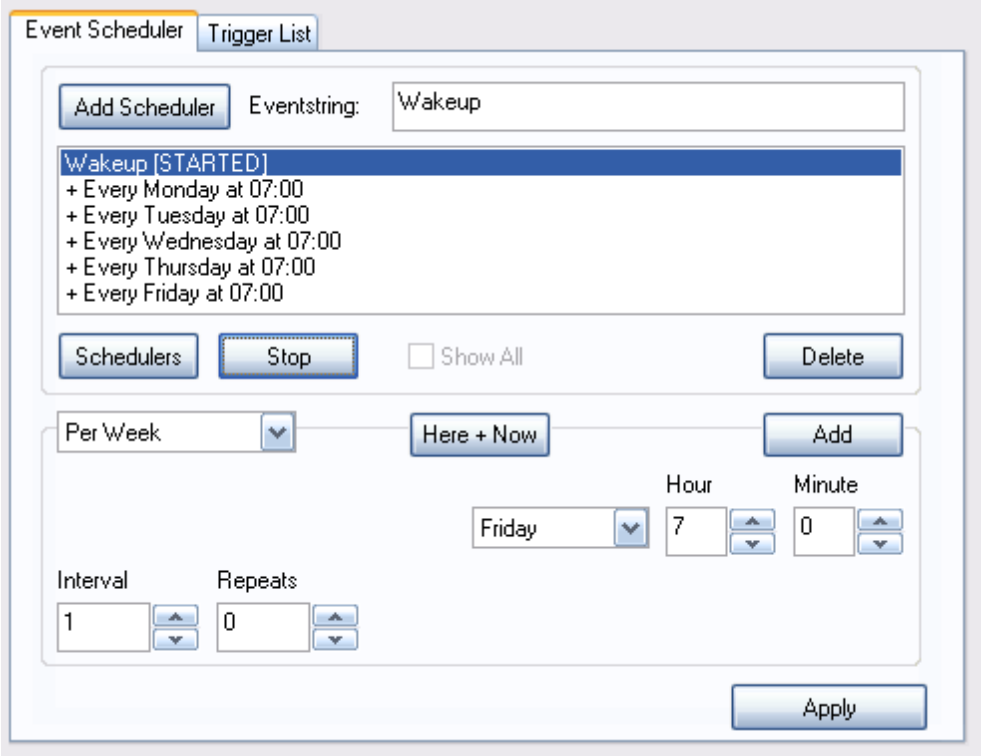
Constraint: Constraints are added to tasks to limit the period of operation of a periodic task. You can set a **not before** date, a **not after** date, both or neither.



The top pane shows a list of all the defined schedulers or a list of the tasks and constraints of one scheduler. Use the button bottom left of the pane to switch between the two and to refresh the list. Normally, only schedulers defined using this tab are shown in the list, but checking **Show all** will list all the schedulers on the system including, for instance the Sunlight Events schedulers.

To create a new scheduler, enter the Eventstring and press **Add Scheduler**. If necessary switch to the Tasks view.

© 2008 Promixis, LLC

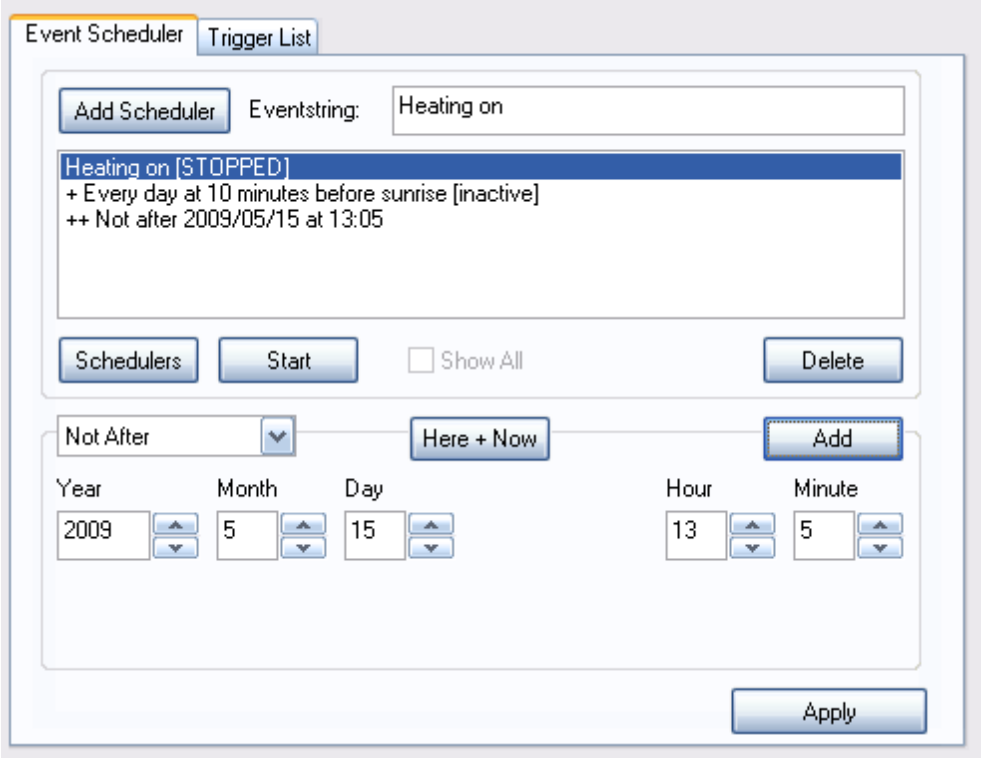


The first line of the Tasks View represents the **scheduler**. Singly indented lines represent **tasks** and doubly indented lines (see screenshot below) represent **constraints**.

To add a Task, highlight the first row in the task list and pick the required **Task Type** from the drop-down top left of the lower pane. Set the parameters for the Task using the controls in the lower pane. The first row of controls determine when the task will generate the event. The second row shows miscellaneous parameters depending on the task type.

For the repeating task types, **Interval** specifies the gap between events, for example **per day** with an interval of two means every other day. **Repeats** sets a limit on the number of times the event can happen before it is disabled (0 means no limit).

© 2008 Promixis, LLC



For the **sunrise** and **sunset** daily task types, **Latitude** and **Longitude** specify the location for which the time is calculated (by default, or on pressing **Here + Now**, the location specified on the **Geographical Location** settings tab is used). Note that the values are specified in decimal degrees, using negative values for south of the equator or west of the meridian. **Offset** allows the event to be generated a specified number of minutes before or after the calculated time. Note that the calculation used corresponds to the "Official" sunrise or sunset definition used in Sunlight Events.

The editing action depends on the selection made in the top pane. If the **heading** is selected, the task is added to the scheduler. If a **task** or **constraint** is selected, another task replaces the selected one, or a constraint is added to the selected task replacing any existing constraint of the same type. The selected object (task, constraint or scheduler) can be deleted using the button bottom right of the upper pane.

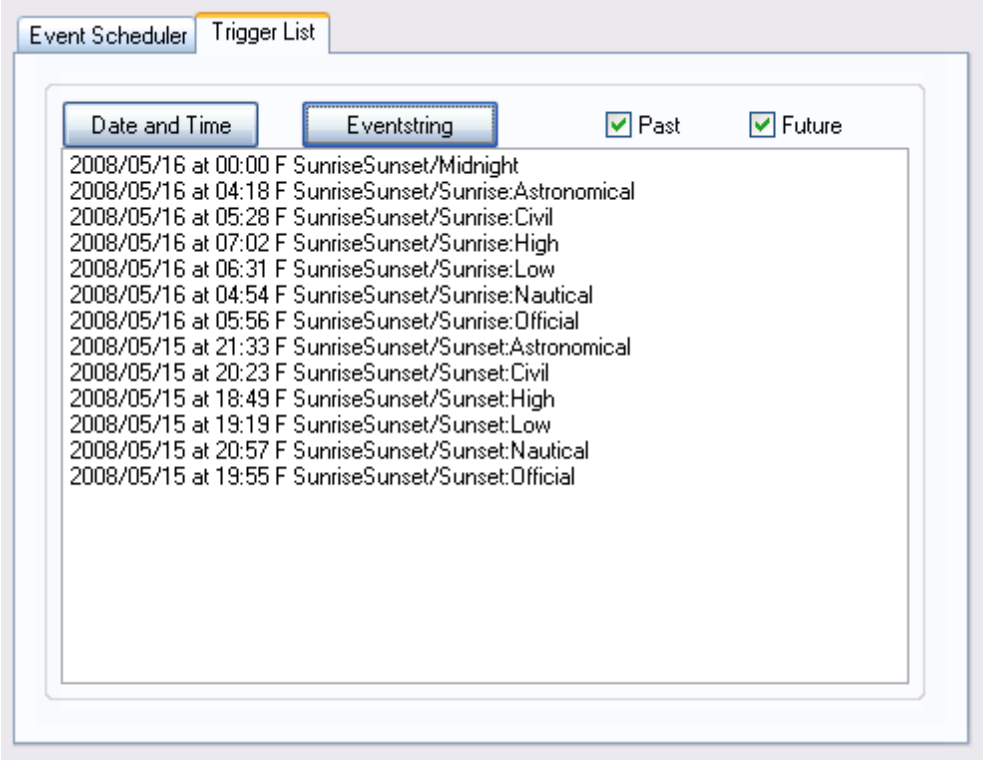
After creation, a scheduler must be **started** before it will generate any events. You can also **stop** a scheduler which is already running to temporarily suppress event generation.

To use the scheduler to trigger actions in the Girder Tree, add an Event to the Action or Macro to be triggered and, in the Event Editor, select the **Lua: Scheduler** Event Device. The Event Strings for the schedulers you have defined will then be available in the Event String drop-down.

Trigger List

The **Trigger List** tab shows all the schedulers on the system with the time and date of the next and/or previous event. You can sort the list by time and date or by event string.

© 2008 Promixis, LLC



Note that the event is shown in **EventDevice**/**EventString** format except for those created on the Event Scheduler tab, for which only the EventString is shown. Where possible, the EventDevice will be shown in text form, but an unknown device type will be shown as a number.

5.3 X10 Home Automation

Requires Girder Pro.

X10 is a widely supported standard for lighting control, electrical appliance switching, environmental control and security monitoring using signalling carried by mains wiring or radio (RF). For more information see http://www.x10pro.com/.

Girder supports Cm11a, Cm17a (x10 send only), Powerlinc (USB or RS232) and CM15a powerline interfaces. Powerlinc (http://www.smarthome.com/) is recommended for maximum functionality. For RF interfacing Girder supports the W800RF32 interface (http://www.wgldesigns.com/).

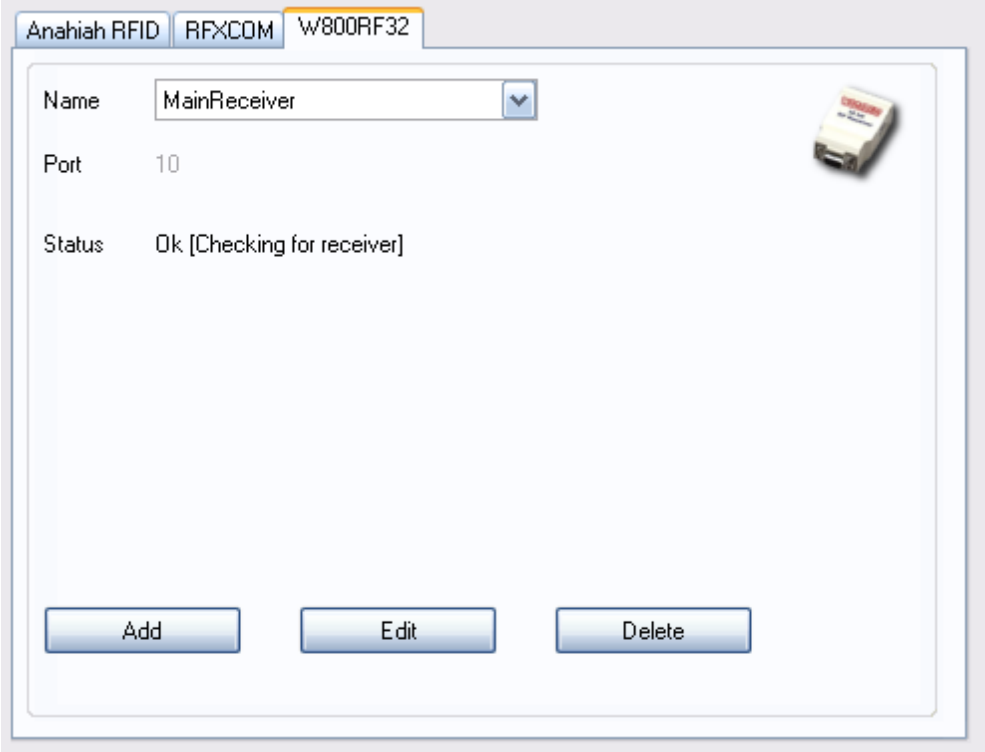
Interface Configuration

X10-CM1X and PowerLinc USB devices each have their own plugin. Ensure this is enabled on the **Plugins** tab. For CM1x, switch to the **X10-CM1X** tab and select the Com Port and the Device type. For PowerLinc, switch to the **PowerLincUSB** tab and select the send delay (if required). Press **OK** and then restart Girder.

PowerLinc (serial) and W800RF32 devices are supported via the Generic Serial Plugin. Ensure this plugin is enabled on the **Plugins** tab, switch to the **Serial** tab, click on the com port to which the device is connected and press **Change Device**. Select **PowerLinc Serial** or **W800RF32** in the **Select Device** dropdown and press **OK Change**. Press **OK** and then restart Girder.

© 2008 Promixis, LLC

For the W800RF32, switch to the tab in the X10 section of Automation Settings.



Ensure the correct serial port is selected (if not, it can be changed here). The **Antirepeat time** setting is used to 'slow down' devices that repeat too quickly.

The w800rf32 receives standard X10 commands and X10 security device commands and generates events for them. More advanced event handling for standard X10 devices can be specified. This is done by selecting the house code and then checking **Enable X10 events for this house code**. You can also check **Send events to the powerline** to have Girder echo this house code to the powerline (requires a X10 powerline controller as listed above).

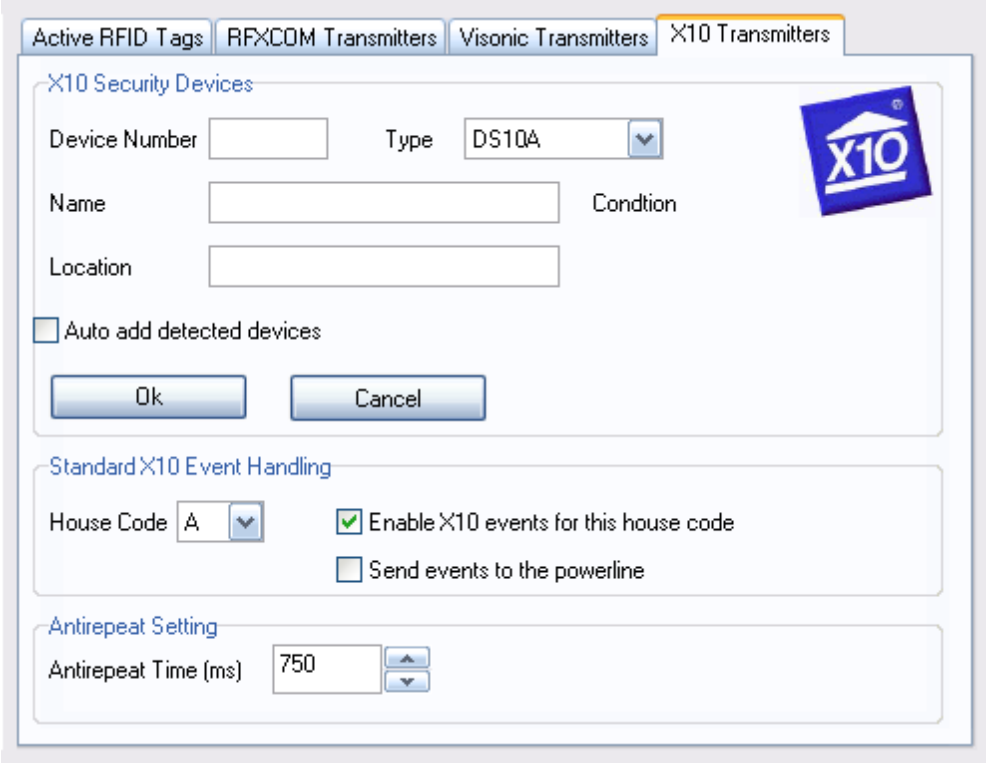
When working with a mix of RF and powerline devices, there are two ways to go. You can install a RF to powerline bridging device such as the V572 (http://www.wgldesigns.com/) and bring all the signals into Girder using one of the supported powerline interfaces. Alternately, you can install both a W800RF32 and a powerline interface and have Girder act as the bridge. The first approach may be more robust, because RF signals reach the powerline even when the PC is not working, however it adds delay between RF devices and Girder due to the limitations of powerline signalling. Bridging in Girder allows both powerline and RF to work at full speed.

Device Configuration

Girder provides an X10 device manager to allow a convenient method to name all X10 devices. Each device is given a location and name that must be unique.

Naming for powerline devices is provided on the **Devices** tab whilst RF security device naming is on the **W800RF32** tab.

© 2008 Promixis, LLC



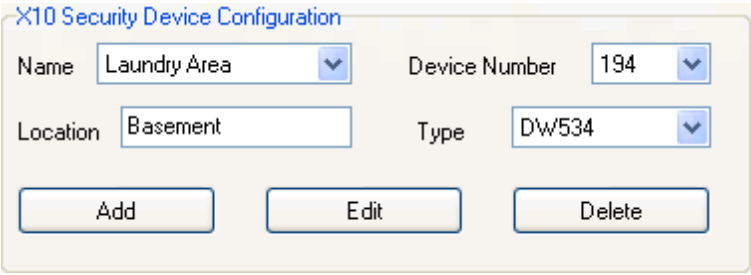
A device **Type** should be assigned to each device. If the device type is not listed, select the closest matching type and adjust the **Status report only**, **Status request**, **2 Way** and **Dim** settings to match the features of the device.

Device polling can be used for devices that support the X10 Status Request command. Girder will check the status of these devices using the **Polling interval** set on the **Settings** tab. Girder always waits for at least 10 seconds of powerline inactivity before polling for a device's status.

Girder supports advance scene management as implemented by SmartHome and PCS type devices. These devices use X10 addresses to define lighting scenes. When you add a device, specify the scenes that it belongs to by either entering the X10 address of the scene or the name of the scene. Add scenes to girder by selecting the device type **X10 Scene** and add the names or addresses that this scene controls to the **Members** field.

Note: Girder does not send commands to scene devices, it uses this information to track device status.

To configure security devices, switch to the W800RF32 tab.



© 2008 Promixis, LLC

Set the **Location** and **Name** of each security device against the **Device Number**. This number changes when the device loses power, so you will have to modify these settings when you change the batteries. Also select the device **Type** from the drop down box.

Applications

The X10 Controls Actions group provides Girder Tree actions for generic X10 control based on house codes and unit codes. The X10 Actions group provides easier to use and more sophisticated actions which use the names, locations and device type information set above.

Use the Device Command Action to send a command to an X10 device. Use the SmartHome Commands Action to send the commands specific to the SmartHome devices (see the manual that came with the device for details). The House/Unit Code Commands Action is a more convenient alternative to the X10 Controls Actions for sending raw X10 commands.

There is also named Girder Event support. In the Event Editor, select **Event Device** = **Lua: X10** . The **Event String** dropdown now lists all the events from named devices. Similarly, for security devices, use the **Lua: W800RF32** Event Device. In both cases, the events offered depend on the device type.

5.4 Girder to Girder Networking

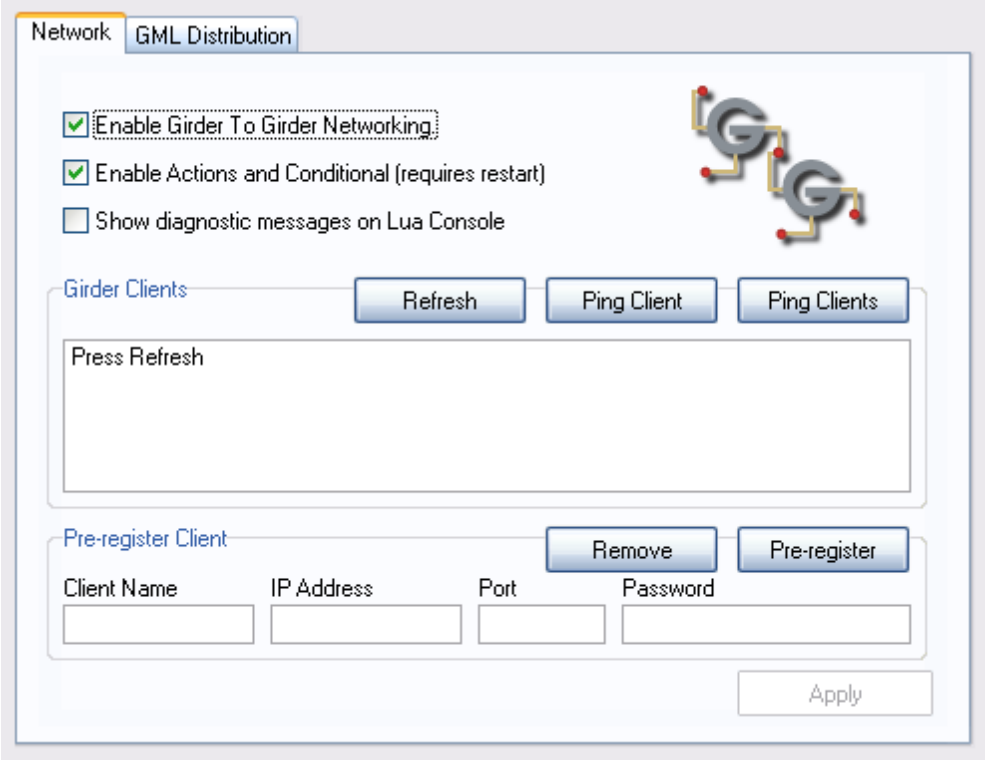
Requires Girder Pro.

Girder nodes on multiple PCs on the same local area network (LAN) can work together using the Girder to Girder Networking feature. This depends on the Communication Server Plugin and is only available in the Pro and Whole Home Pro versions of Girder.

The **Run Server** tick-box on Communication Server Configuration must be ticked on all machines participating in the network. Some features such as remote scripting and Lua table copying also require **Allow Remote Code** to be ticked. If you change the **Server Password** from the default "girder", you will need to use Pre-registration (see below) on all the other machines.

The **Girder to Girder** group in the Automation section allows configuration and trouble-shooting of this feature on the **Network** tab and distribution of GML files to remote Girder machines on the **GML Distribution** tab.

© 2008 Promixis, LLC



Enable Girder To Girder Networking: Tick to enable the networking feature.

Enable Actions and Conditional: Tick this to enable the Girder Tree Actions and Conditional. Girder must be restarted for changes to this setting to take effect, or for the Actions and Conditional to appear after networking has been enabled.

Show diagnostic messages: This is an advanced feature that can be enabled for trouble shooting. Detailed transaction reports are shown on the Lua Console.

Girder Clients

The list shows the clients (other Girder machines) recognized on the network. The name of the client is shown first, followed by the status and then the IP Address and Port. Press the Refresh button to ensure the latest information is showing.

The status may be **no connection**, **online** or **offline**. Connect on demand is used, so **no connection** just means that no messages have been sent yet. Status will change to **online** or **offline** once a message send has been attempted. A test message can be sent using the **Ping Client** or **Ping Clients** buttons. Highlight a client and press **Ping Client** or press **Ping Clients** to try them all.

Pre-registration

Normally, Girder detects client machines automatically. However on some networks it is not able to do this. Also, Girder assumes that the password for the Communications Server on all the clients is the default, "girder". If you change the password or if you want to work with clients that are not discovered automatically, use pre-registration.

To pre-register a client, enter its details in the Pre-register box and press the **Pre-register** button. To register a password for a client, click it in the Girder Clients list, enter the password in the pre-register box and press the **Pre-register** button. Press **Apply** to store the pre-

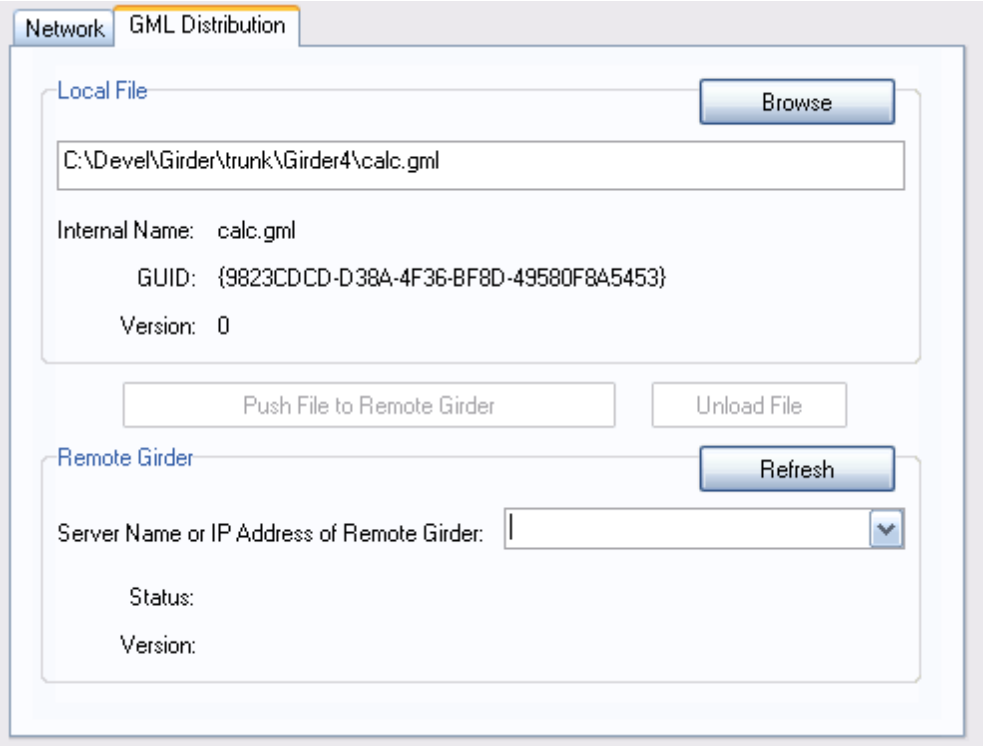
© 2008 Promixis, LLC

registrations permanently.

To remove a pre-registration, click a client with a **(PreReg)** flag and press the **Remove** button. Then press **Apply**. This removes the pre-registration flag, but the client remains in the list until Girder is restarted.

GML Distribution

The GML Distribution tab supports a scenario where GML files are developed and stored on one machine (the "server") and distributed out to other machines (the "clients"). Other supporting files can also be transferred.



To select a local file for transfer, either use the Browse button or type the path and filename in the edit box. If a GML file is selected, the Internal Name, GUID and Version Number will be shown.

To select the remote Girder to work with, pick it by name from the drop-down list or type the name or IP address in the box. The Refresh button may need to be pressed to add all the clients to the list. If a GML file and a valid client are specified, the status of that file in the remote Girder and the version number of the file are reported.

The two buttons in the middle of the dialog change the file status on the remote Girder. The Push button will transfer the file to the Uploads directory in the remote Girder. If the file is a GML file, it will also be loaded into Girder. The Unload button operates on GML files only, unloading them from Girder but leaving them in the Uploads directory.

Applications

Girder to Girder networking is used by the Caller ID Application to transfer information to all the networked machines. Actions from the Girder to Girder Actions group make it easy to generate events in another Girder, to "pump" whole categories of events from one Girder to another (for example all the events from a particular remote) and even to run Lua scripts on another Girder.

© 2008 Promixis, LLC

5.5 Caller ID Application

Requires Girder Pro.

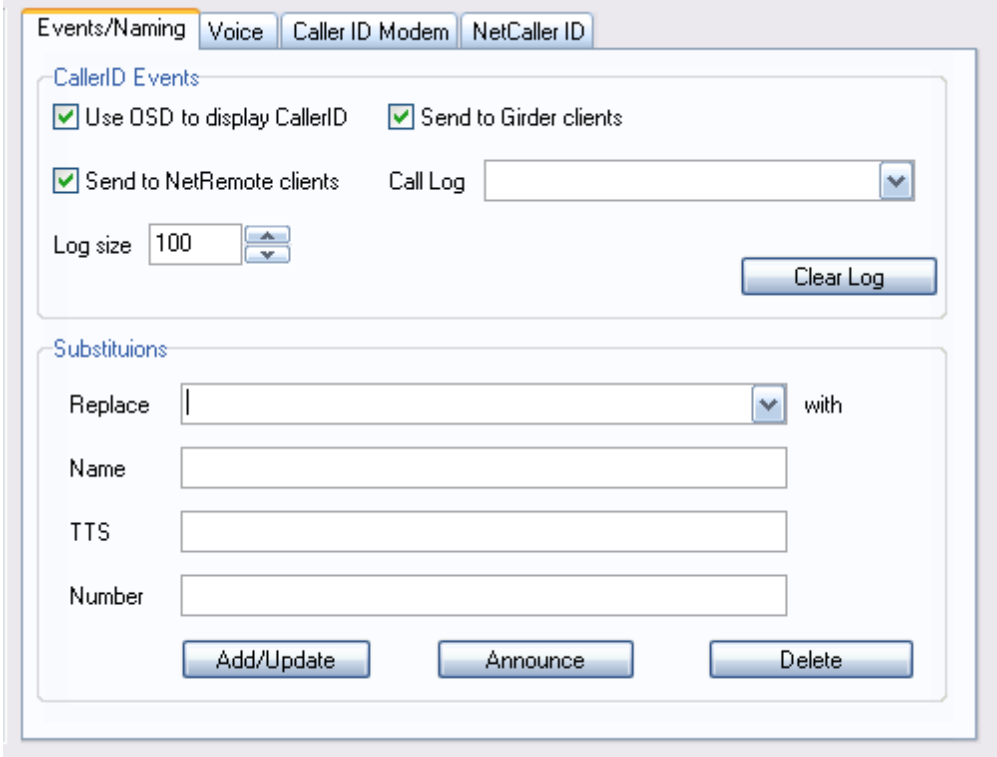
The Caller ID application is accessed from the **Automation Settings** group. Caller ID messages must be generated on at least one Girder machine, and this requires the Generic Serial Plugin which is only available in the Pro versions.

Configuring the Caller ID Provider

Ensure that **Generic Serial** is enabled on the **Plugins** tab, change to the **Serial** tab, select the com port that your modem or NetCallerID device is attached to, press **Change Device** and select either **CallerID Modem** or **NetCallerID** in the select device drop-down. Press apply and restart Girder. Incoming calls on the line connected to the modem should now cause Caller ID Events in the Logger.

Configuring the Caller ID Application

Click on **Automation Settings** and then **Caller ID**.



Use OSD to display CallerID: Tick this to have an On Screen Display show whenever a new call is received.

Send to NetRemote clients: Tick this if you have a Promixis NetRemote device. Caller ID information will be sent by Girder to all connected NetRemotes and can be displayed.

Send to Girder clients: Tick this to copy Caller ID Events from devices connected to this machine to all Girder machines on the local network. OSD and Voice announcements can then be generated on any or all Girder machines.

Log Size: Set the number of previous calls that will be remembered. The drop-down to the right

© 2008 Promixis, LLC

shows this call history.

Announce CallerID using Voice: If ticked, Girder will announce incomming calls verbally through the audio output using the Voice Application.

Start announcement with: This text will be read out prior to the number or name of the caller.

End announcement with: This text will be read out after the number or name of the caller.

Substitutions: The information supplied by the Caller ID may be replaced by alternate information for display or announcement. Pick a call from the call log and enter a replacement name and/or number and press **Add/Update**. Pick a call and press **Delete** to remove the substitution and revert to using the received Caller ID information.

Custom Applications

The Caller ID application generates Girder Events which can be used to trigger actions in the Girder Tree. Select **Lua: CallerID** as the Event Device and then either **NewCall** or **NewCallBroadcast** as the Event String. The former is raised on the machine with the caller ID device and the latter on any other machines if **Broadcast to Girder Clients** is checked. In many cases, it does not matter which machine raised the event, so you will need to respond to either.

Pld1 is the caller name, **Pld2** the calling number, **Pld3** the date and **Pld4** the time of the call.

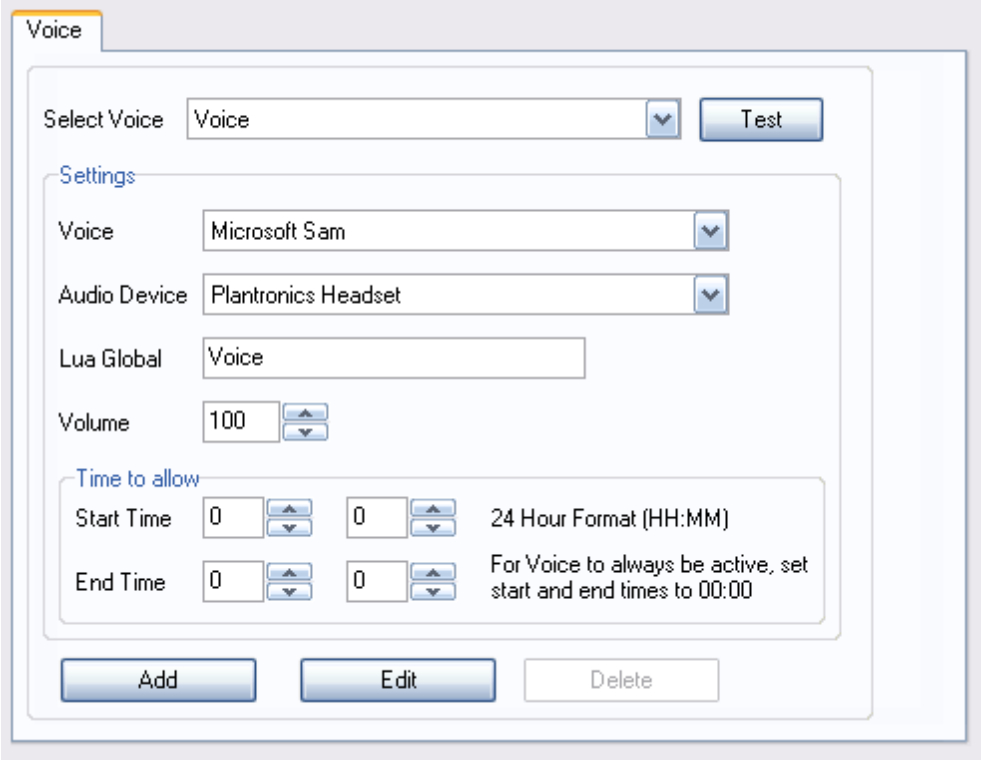
5.6 Voice Application

Requires Girder Pro.

Girder can convert text messages into speech and play this through the computer's audio output. This is used by the Caller ID application and by the Voice Actions.

Voice is accessed from the **Automation Settings**, **Voice** group.

© 2008 Promixis, LLC



Enable Voice/Speech: Tick this to enable spoken announcements.

Time to allow: Spoken announcements can be restricted to particular times of day (for example, to suppress them at night). Enter the start and end times in 24-hour format. To allow at all times, enter 00:00 and 23:59.

Select Voice: Windows provides one voice as standard (Microsoft Sam), but other voices may be installed by third party speech applications or obtained from other sources. Select the voice to use here.

Audio Device: Some computers have more than one audio output device. If so, this can be selected here.

© 2008 Promixis, LLC

Part

6 User Interface Guide

This section provides information about the Girder User Interface.

6.1 Novice or Expert?

Girder has two alternative interfaces, *Novice* and *Expert*. Switch between them using the **View** menu.

Using the Novice interface you can

- Add a remote control device.
- Add a Program Definition.
- Teach Girder which remote function controls which program function.
- Configure Girder options.

See Main Window (Novice).

Using the Expert interface you can additionally

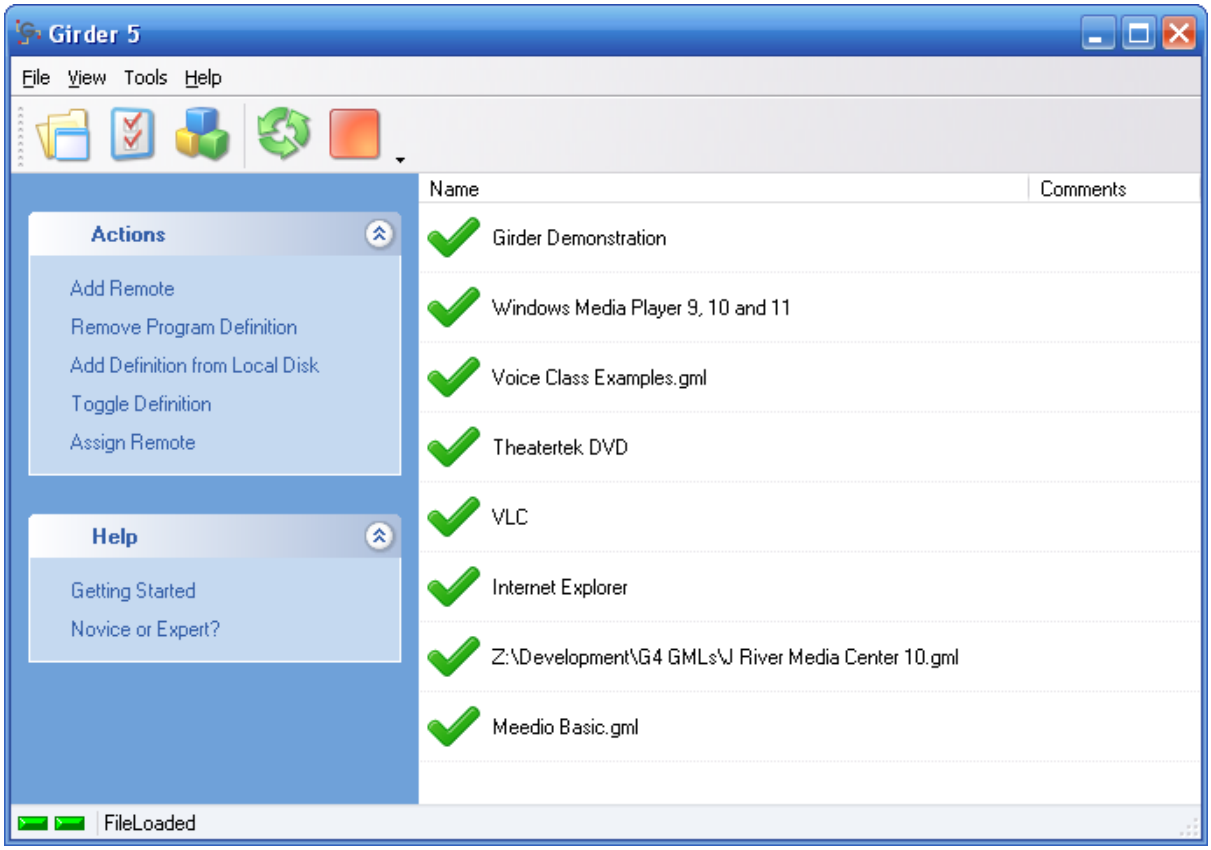
- Create Program Definitions from scratch or modify existing ones.
- Configure your remote control device to control computer and operating system functions.
- Schedule actions to happen once or repeatedly in the future.
- Write complex and powerful automation functions using LUA scripting.

See Main Window (Expert).

© 2008 Promixis, LLC

6.2 Main Window (Novice)

This is the main application window in Novice mode. Right click on Girder icon in notification area and select **Show Girder**. Switch modes if necessary from the **View** menu.



Toolbar

 Add Definition from Local Disk.

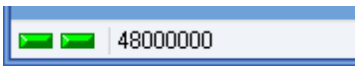
 Show the Settings Dialog.

 Show the Component Manager Dialog.

 Start or Stop Event Generation.

 Reset the scripting Engine.

Statusbar



The left indicator shows green if event generation is enabled, grey if disabled and flashes amber when an event is detected. The Event String of the latest event appears to the right of the indicators.

© 2008 Promixis, LLC

The right indicator flashes amber when Girder is busy processing and lights green when it is ready.

Definition List

The definition list on the right of the window shows all currently loaded application definition (GML) files. A green tick indicates that the actions in the definition are enabled, a red cross that they are disabled. Highlight a file for commands that act on a specific file.

Actions Task Box

The Actions Task Box on the left of the window contains a menu of Novice Commands.

Add Remote: Brings up the Add Remote Wizard to enable a specific remote control and its event map.

Add Definition from Local Disk: Allows selection of an application definition (GML) file stored in your local file system and installs it.

Remove Program Definition: Removes the definition highlighted in the Definition List from Girder, but leaves the GML file in your file system.

Check for Definition Updates: Brings up the Definition Updates dialog to check for and install any updates available for definition files that are already installed.

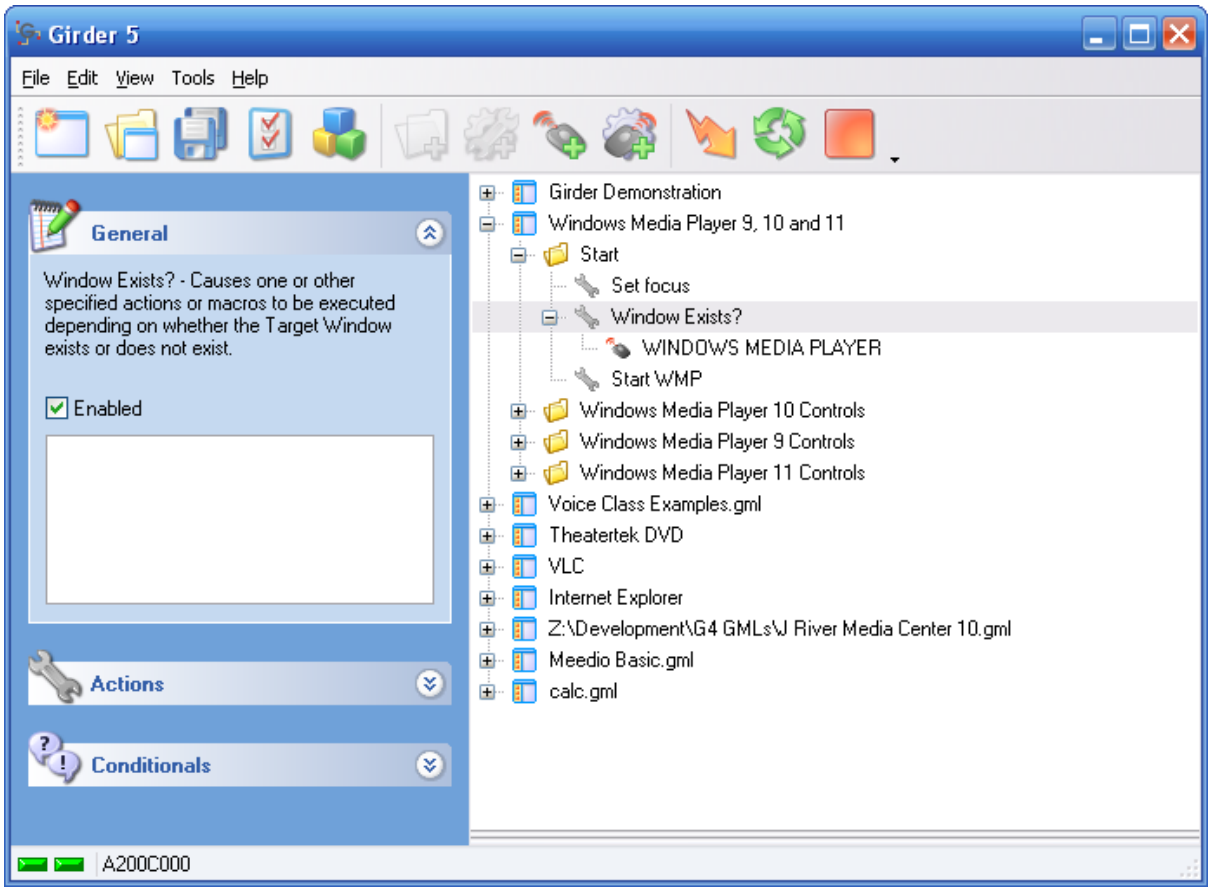
Toggle Definition: Enables or Disables the actions in the definition highlighted in the Definition List.

Assign Remote: Brings up the Remote Assignment Editor to change the mapped remote which triggers the actions in the definition highlighted in the definition list.

6.3 Main Window (Expert)

This is the main application window in Expert mode. Right click on Girder icon in notification area and select **Show Girder**. Switch modes if necessary from the **View** menu.

© 2008 Promixis, LLC

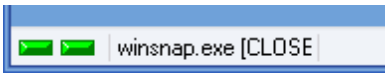


Toolbar

Create new GML file.
Add GML file from Local Disk.
Save all changes to Disk.
Show the Settings Dialog.
Show the Component Manager Dialog.
Add Group Node to tree.
Add Macro Node to tree.
Add Event Node to tree.
Add Macro Event to tree.
Test the highlighted Action or Macro.
Start or Stop Event Generation.
Reset the scripting Engine.

Statusbar

© 2008 Promixis, LLC



The right indicator shows green if event generation is enabled and flashes amber when an event is processed.

The left indicator shows green when Girder is ready, amber when it is busy. Either indicator showing red indicates a fault.

The left hand number references the GML file containing the node highlighted in the Girder Tree. If a GML is selected 2 numbers are displayed, the right hand number references the node within the file. These references are used in some Lua scripting functions.

Girder Tree

The Girder Tree, to the right of the main window, shows all currently loaded GML files and their contents. Each node has a right-click context menu and can be double-clicked to bring up an editor dialog.

General Task Box

Provides information about the selected node in the Girder Tree and provides controls for enabling and disabling it and for editing attached documentation.

Actions Task Box

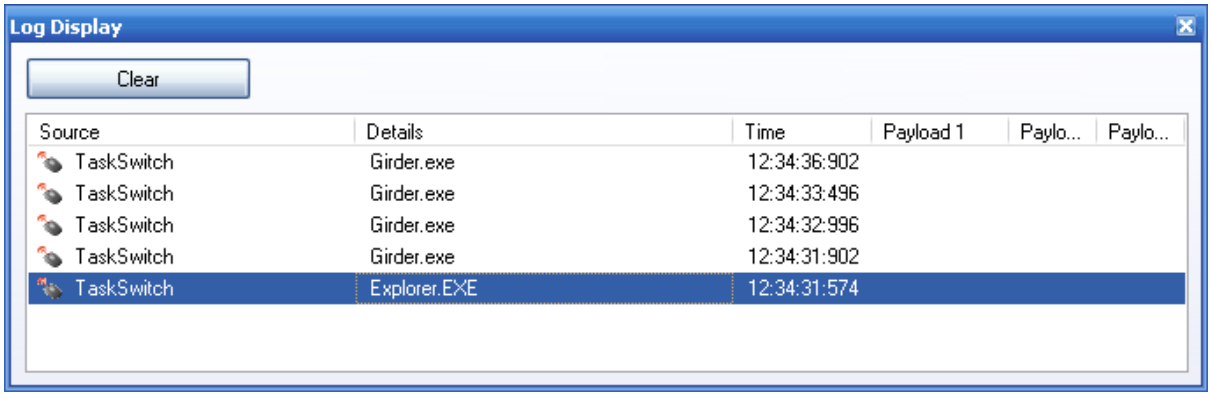
Contains a tree of available actions. These can be dragged onto nodes in the Girder Tree or you can double-click to add an action to the selected node.

Conditionals Task Box

Contains a tree of available conditionals. These can be dragged onto nodes in the Girder Tree or you can double-click to add a conditional to the selected node.

6.4 Logger

The Log Display (**view**/**logger** or **F4**) logs Girder Events as well as informational and error messages.



The **Source** column gives the Girder Plugin or the subsystem that generated the event or message. **Details** is the Event String or a message. Time shows hour of day, minutes, seconds and milliseconds. Hover the mouse pointer over an event entry to see the Event Payload if any. Additional columns are available for Date and for the first three payload strings. To enable or

© 2008 Promixis, LLC

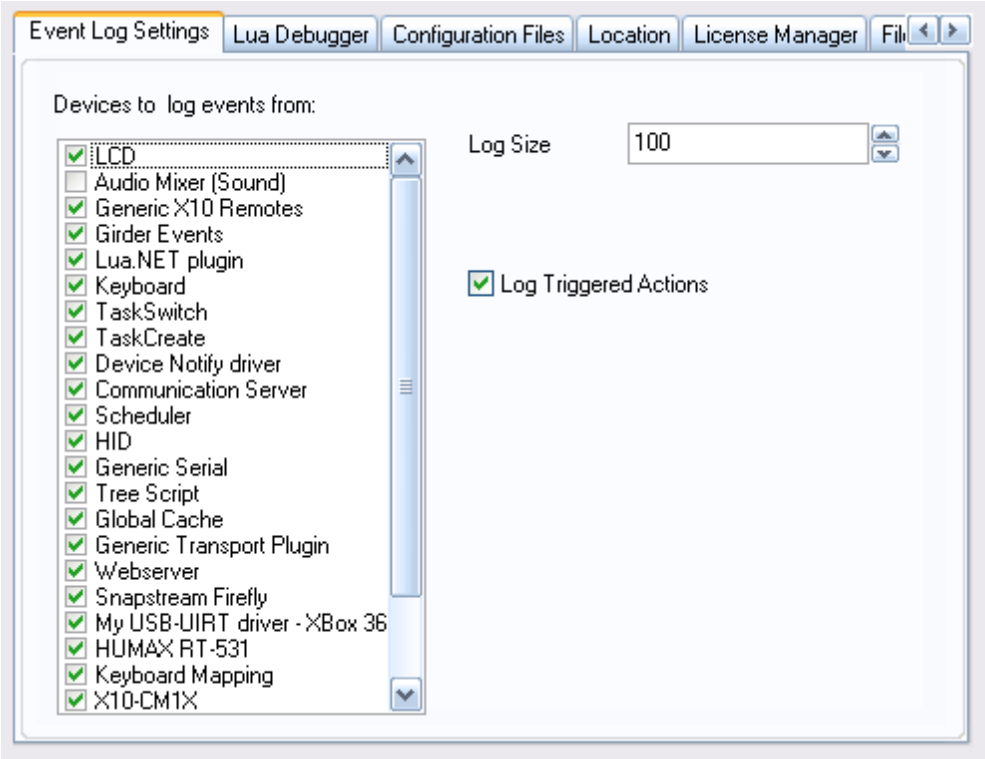
disable columns, right-click in the column headings area. You can re-order columns by dragging the column header or resize by dragging the boundary markers in the column header strip.

Clear: Press to clear all messages from the window.

Drag and Drop: You can drag an event icon from the logger onto an Action, Macro or Macro Event in the Girder Tree. This produces an Event Node which recognizes future occurrences of the same Event and triggers the associated Action or Macro.

Logger Configuration

The logger has a dedicated configuration page in the Generic section of the Settings dialog.



Devices to log events from: You can disable logging for specific devices (plugins) if they are getting in the way.

Log Size: The number of past log entries to keep.

Log Low Level Scripting Debug Messages: Tick to get extra information when debugging script problems.

Log Triggered Actions: Tick to log when actions in the Girder Tree get executed.

6.5 Add Remote Wizard

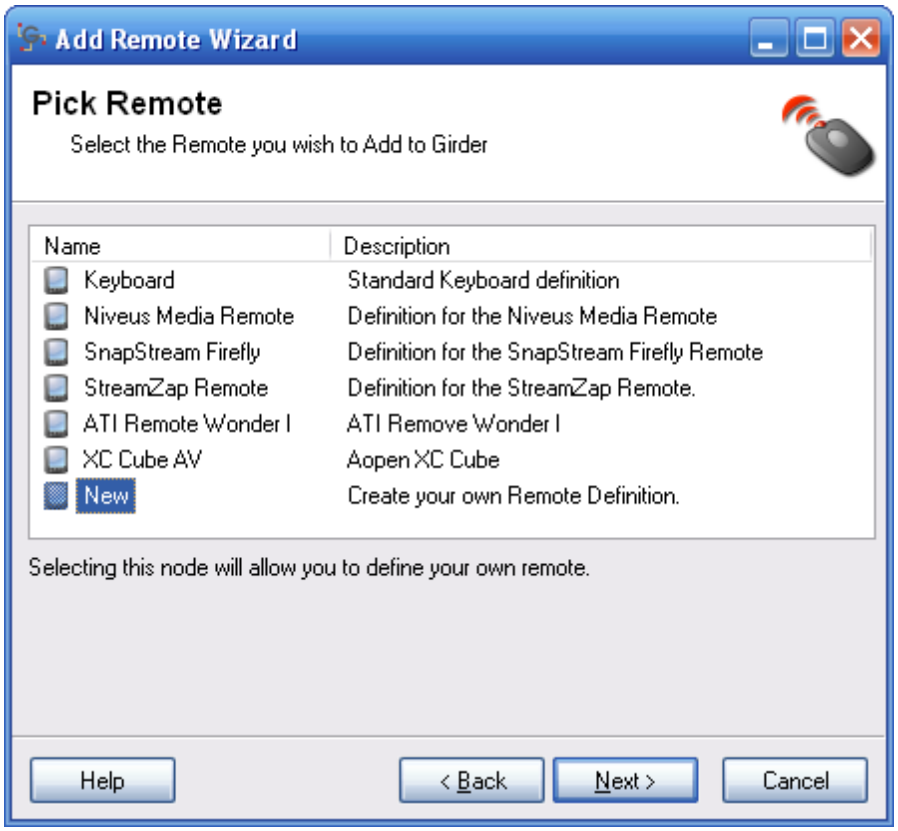
Note the add remote wizzard adds a remote as an INPUT device for Girder, this is not the place to add actions that send out IR signals. Depending on your IR hardware this is done through Actions.

The Add Remote Wizard is accessed using the **Add Remote** item in the Actions Task Box on the Novice Interface, or from the **Tools** menu on either interface. It is used to install a supported remote control device or to teach Girder about the buttons on a device which is not directly

© 2008 Promixis, LLC

supported.

After clicking Next on the introductory page, the list of supported remotes appears.



The **Online Update** button checks the internet for any remotes that have been added or modified since Girder was released.

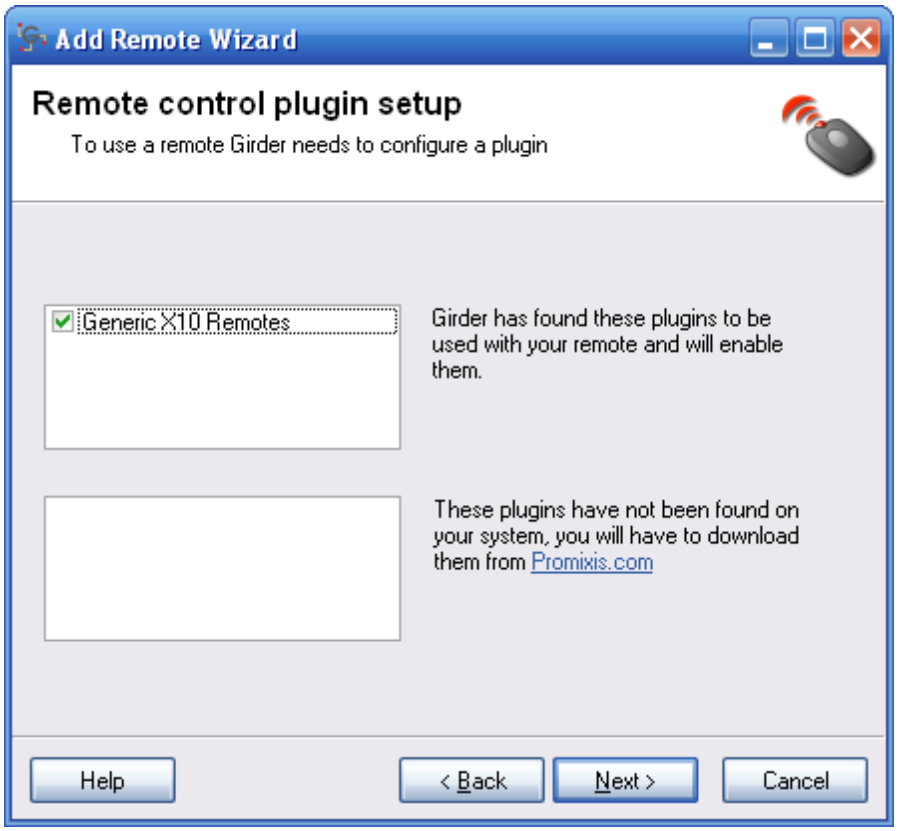
You can select one of the supported remotes, or select "New" if your remote is not directly supported.

The area below the list of remotes identifies any third-party software requirements for the remote you have selected. This may include an internet link to the necessary download. After following this link, you should cancel the Wizard and exit completely from Girder before installing the software according to instructions supplied with it. Then restart Girder and the Wizard.

Supported Remotes

When a supported remote is selected, **Next** brings up, if necessary, a page identifying any Girder Plugins required to use that remote.

© 2008 Promixis, LLC



If Plugins are listed in the lower window, follow the Promixis link and then close down the Wizard and Girder. After downloading and installing the required Plugins, restart Girder and the Wizard.

Press Next and then Finish on the closing page. Note that some Plugins and systems will require additional configuration in the Settings dialog.

Unsupported Remotes

Selecting the "New" remote in the second step of the Wizard follows a different path which enables "Unsupported" remotes to be added. "Unsupported" means that the device communicates via a method Girder knows about, but button mapping for the specific device is unknown.

Pressing **Next** with "New" selected leads to the "Detecting Remote" step. When a button on the remote is pressed, if Girder can "see" it, the Wizard steps to the next page. If Girder cannot see the remote, nothing will happen. In this case press the button to show some diagnostic steps which may enable the device.

The next step allows confirmation that Girder has detected the right source device (if not, reselect from the drop down list) and naming of the device. This name is actually the name of the **Mapping Device**.

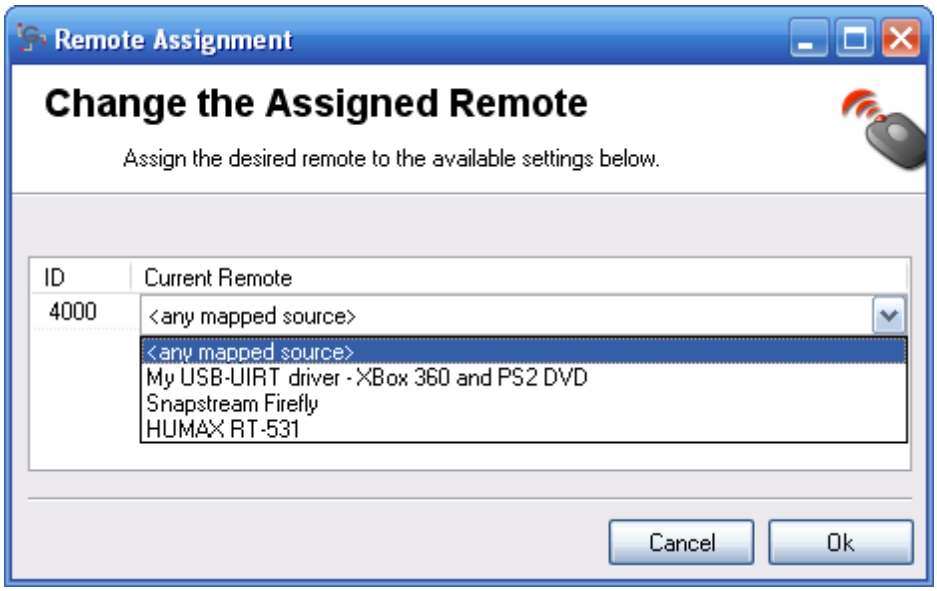
The next step allows training of Girder to recognize the buttons on the remote. Select the button you wish to train on the left list and then press the corresponding button on the remote. Repeat until all the buttons are trained.

Press **Next** to reach the final page and then **Finish** to save the new Mapping Device.

© 2008 Promixis, LLC

6.6 Remote Assignment Editor

The Remote Assignment Editor is a means of bulk changing the Event Device in all Event Nodes within a GML file or a Group. It is accessed in the Novice Interface from the Actions task box (GML file only) or in the Expert Interface from the right-click menu on Files and Groups.



The editor shows a row for each distinct Event Device in the Event Nodes which are children of the selected nodes. To change an Event Device, select a row by clicking on it, wait a short moment and click again to access the drop-down. The Dropdown offers the Mapping Devices defined on the system (this will usually be a device for each defined remote), and the pre-defined device "<any mapped source>".

Select a named Mapping Device to associate a particular remote with the node, or <any mapped source> to allow the node to respond to all remotes.

6.7 IR Profile Editor

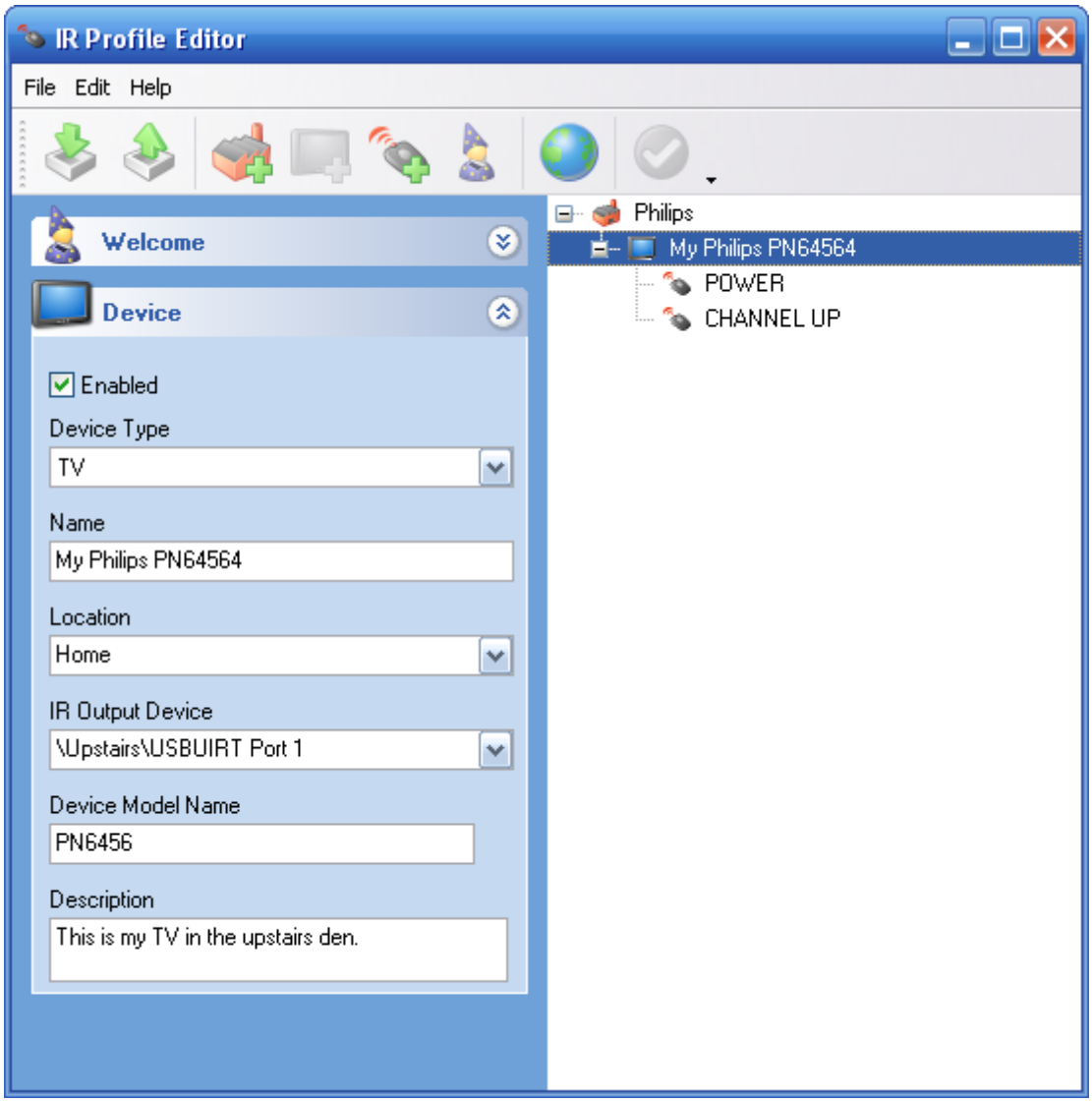
Most consumer electronics like DVD Players, TV and Projects can be controlled by a remote. Typically these remotes use Infra Red as their communication medium. (Infra Red is a type of light that we cannot see with our eyes). By telling Girder about your consumer electronics (called devices in this context) we can control these devices easily, so this functionally is about controlling external devices, this in contrast to the IR Remote Wizard which sets up remotes to control Girder.

To setup a new device to be controlled open the IR Profile editor, you can find this in the View menu of the main Girder interface.

The easiest way to setup a new device is to go through the IR Profile Wizard. This will walk you through the steps of making or downloading a new device. We have an online IR database available of many thousands of IR devices. If yours is not listed consider contributing it to the community! If a profile you downloaded does not work, do not hesitate to upload a fixed version!

Currently the IR learning and testing only works with the USB-UIRT. And as output device GlobalCache and USB-UIRT are supported more devices are underway...

© 2008 Promixis, LLC



There are three different icons to be found in the tree view in the profile editor.


The Manufacturer. This node on the tree represents the manufacturer of all devices below it. Only the name can be set and must be unique.


The Device. This node represents the device, for example your TV or your Amplifier. A device has a few properties

- **Device Type:** The device type is used by the GUI renderer to decide how to display this device.
- **Enabled**: This toggles the device on and off.
- **Name**: The name of the device, in combination with the location this must be unique.
- **Location**: The location of this hardware.
- **IR Output Device**: The device used to transmit the IR codes.

© 2008 Promixis, LLC


An IR code. Each device will a few remote codes assigned to it, these correspond to the buttons on your remote. You should try to pick from the list of predefined names for this so that device that renders the GUI for this device knows how to group the buttons together.

- **Function**: This is the function that is performed by the device when the button is pressed, try to use the predefined function names to assist the GUI in rendering a sensible layout.
- **IR Code:** This is the actual IR code that is sent when this button is pressed. The format that is expected here is the so called raw-CCF format (meaning the first 4 numbers of the code must be zeros). You can either learn this from your USB-UIRT (or other devices in the future) or download them from external websites, for example remotecentral.com is a great resource.
- **IR Code (odd)**: This is the same as the IR Code field but when this is present the system will send IR Code and IR Code odd alternatively. This to facilitate toggle codes in RAW CCF format. A lot of Philips devices use the toggle code.
- **Repeat Count:** This is the number of times this code should be repeated. Some hardware requires a repeat larger than one. If you are having trouble try settings this to 3 or higher.

Troubleshooting

My device does not appear in the Wizard!
Since there are hundres of thousands of devices out there it is unlikely we'll have an IR profile for all device available, you might have to teach Girder about your device. Simply follow the wizard's instructions.

My device is not responding

- Make sure that the IR emitter is close enough to the device
- Make sure that the IR output device (e.g. the USB-UIRT or GlobalCache) are enabled. (The USB-UIRT needs it's plugin and component enabled!)
- Make sure that you have the correct IR output device selected in the IR profile device.

My device only responds half the time!

This means that it is expecting a toggle code. To setup a toggle code go to the IR code and use the remote to teach it both the 'even' and 'odd' IR code. The remote will alternate between sending the odd and even code after you release the button. So follow this recipe:
- Select IR Code node in the tree that you wish to change
- Make sure the "*Use Odd IR code*" box is NOT checked.
- Press learn
- Hold the button on the remote until learn finished, DO NOT LET GO OR TOUCH THE REMOTE AFTER THAT.
- Now check the "*Use Odd IR Code*" box.
- Hold the button on the remote until learn finished, DO NOT LET GO.

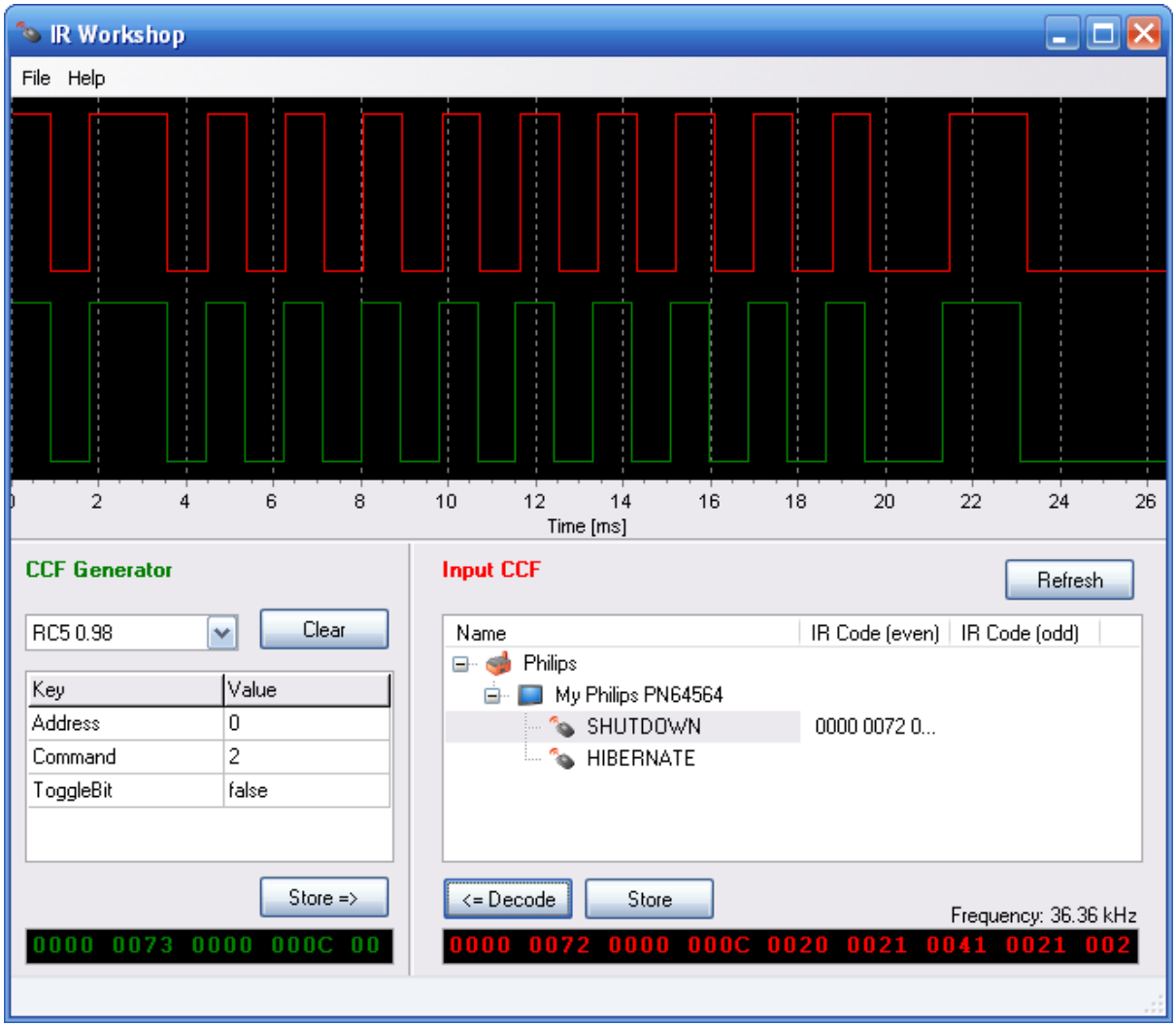
If everything went well you should now have two different codes in the two IR code boxes if not repeat the steps above. If you can't get two different codes to appear this button does not toggle.

I've made some changes but the device manager is still using the old data
The device manager checks every 30 seconds to see if there was an update to your devices. Just wait half a minute and try again.

© 2008 Promixis, LLC

6.8 IR Workshop

The IR workshop is an advanced feature that is useful if you need to clean up captured IR codes or if you need to generate IR codes that you are unable to capture.



As you can see the workshop consist of three parts.

The Signal Scope

The Signal scope or chart is located at the top of the window. It displays the IR signals that you are working on. The red signal is the CCF code from the IR library and the green signal is the generated CCF. The scope allows you to zoom. To zoom in click at the top-left hand point of the area you want to zoom in on and drag the mouse to the lower right hand point. To zoom out reverse this movement. click and hold anywhere on the chart and move the mouse to the top left.

CCF Generator

This panel allows you to generate CCF codes from manufacturer specification. The advantage

© 2008 Promixis, LLC

to this is that the code is very clean and can thus be detected by your hardware more easily. Depending on the mode this panel is in you either see all of the available generators or you see a collection (if any!) of generators that recognized the CCF code on the left panel after pressing "<=*Decode*". The accuracy of detection is listed after the name. In the screenshot you see an example. The RC5 generator recognized an RC5 code with 98% accuracy. To get the full list of generators back simply click *clear*.

Each generator will have different parameters you can configure, this is very protocol and device specific. There is also little documentation available from manufacturers what these values should be for your device. One approach is to learn one of the buttons on your device and then tweak the numbers to see if you can expose any functionality that is not available from your remote but your device does support!

Input CCF

The input CCF panel controls what CCF code you want to display, decode or store back into the IR database. Note that you can select to work on either IRCode (even) or IR Code (odd).

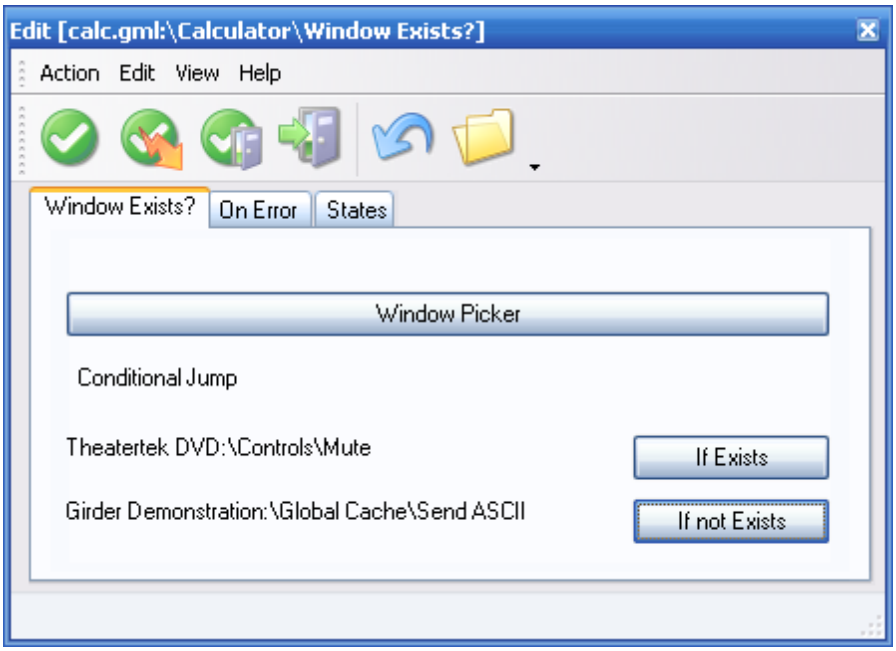
Example Workflow

1. Find the Manufacturer, Device and IR code that you wish to examine.
2. Click on either the Even or Odd IR code column.
3. The red graph should appear in the signal scope.
4. Click on "<=*Decode*".
5. If the signal is recognized by any of the decoders they will be listed in the CCF generator drop down box. Visually inspect the generated (green) signals to see which one matches the best. You can now store this value back into the IR code field by pressing "*Store =>".*

6.9 Action/Conditional Editor

The Action or Conditional Editor is accessed by double clicking on an Action or Conditional node in the Girder Tree. Multiple editors may be opened by using the **Edit in New Window** option on the context menu for these nodes.

© 2008 Promixis, LLC



This is a typical Action or Conditional editor which pops up when a node of that type is added to or double clicked on the Girder Tree. The first tab is specific to the type of action or conditional and is described in the Actions Reference or Conditionals Reference. The On Error tab and the States Tab are described in the next two sections.

Toolbar

 Save changes to settings.

 Save changes and close editor.

 Save changes and test action.

 Close editor without saving changes.

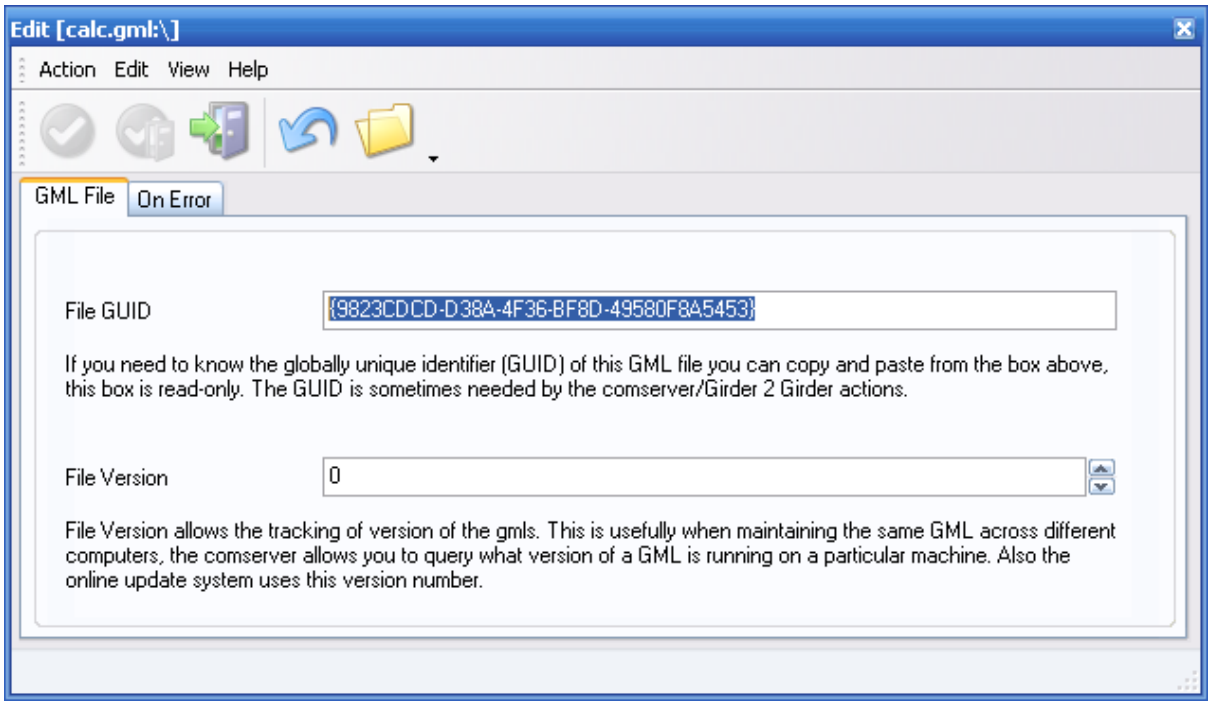
 Undo changes.

 Change action/conditional type.

6.10 GML File Editor

The GML File Editor dialog is accessed by right-clicking a GML File node (top level node) in the Girder Tree and selecting **Edit**. It provides facilities for file configuration control.

© 2008 Promixis, LLC



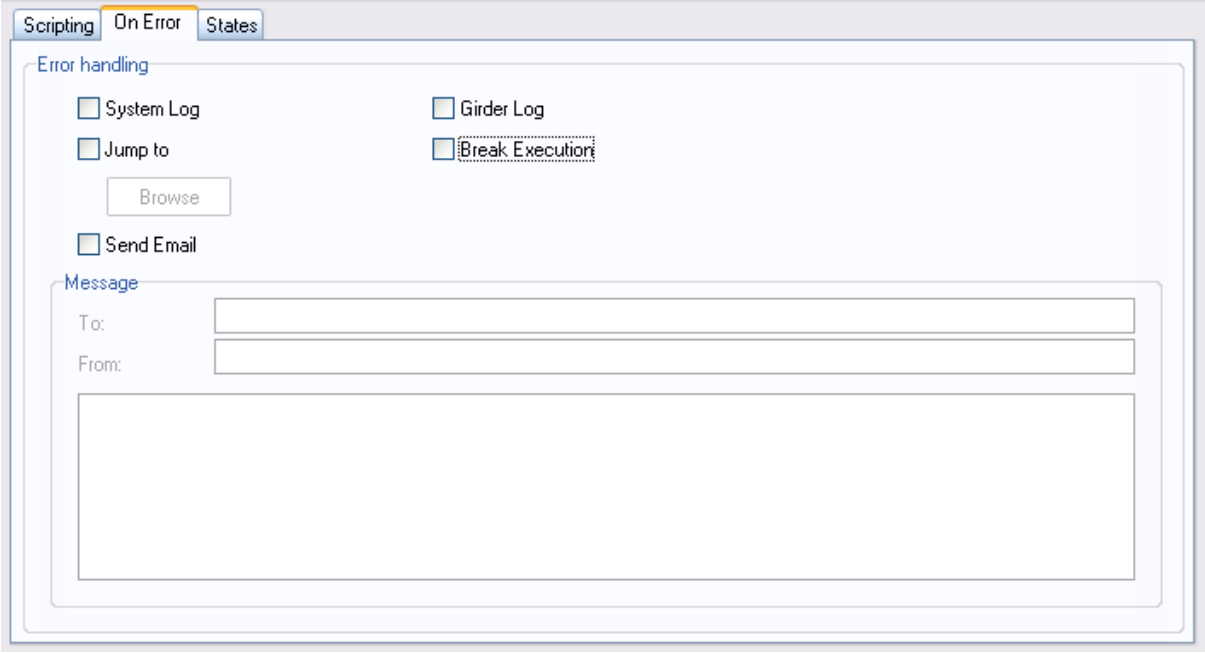
The **File GUID** field is for reference only. The file GUID is assigned once when a new file is created in Girder. It does not change when the file is modified in Girder or copied or renamed in Windows. This string is guaranteed to be unique among all GML files created by all Girder users. Some Lua functions require this string and it can be copied and pasted from here into code editors.

The **File Version** is manually edited using this dialog. It can be interrogated on a remote Girder instance using Girder to Girder Actions or the Lua Girder2Girder Library. It is also used by the GML file download system for Promixis deliverable GML files.

6.11 Error Handling

Error handling in the Girder Tree is hierarchical. If you do not specify error handling for an Action, the error will ripple up to the containing Macro (if present), then the Group and finally the File. The editor for each of these nodes has an On Error tab.

© 2008 Promixis, LLC



System Log: Tick this to write error messages to the Windows System Log.

Girder Log: Tick this to write error messages to the Girder Log.

Jump To: Tick this to jump to another action or macro when an error occurs. The **Browse** button brings up the Tree Picker to select the target node.

Break Execution: Do not execute any further actions for the current Event.

Send Email: Tick this to send an email message when an error occurs. This requires a server to be defined on the General Settings tab.

To: Email address to send error email to.

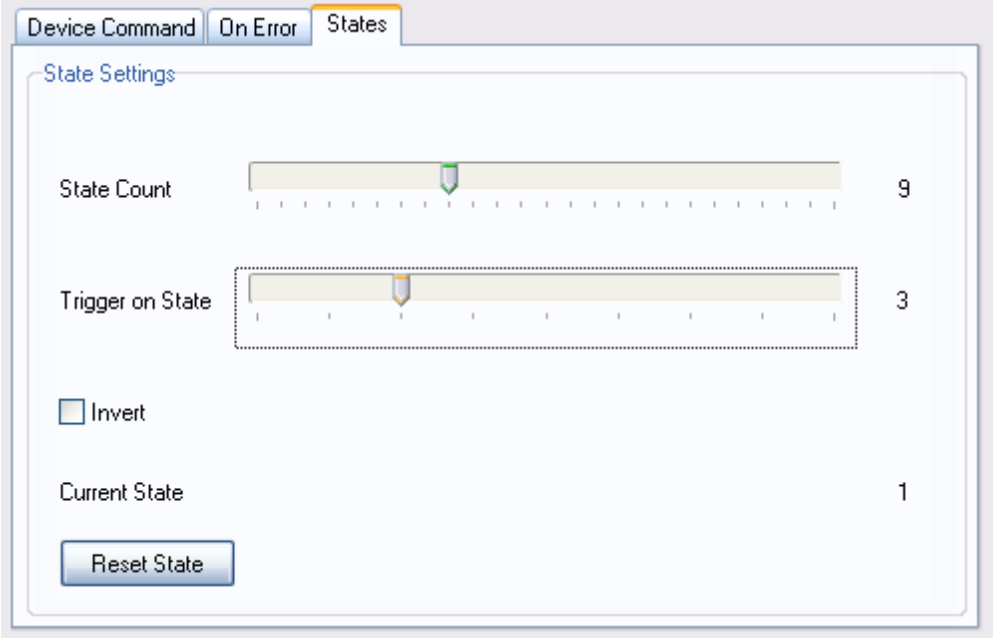
From: Email address for reply to the error email.

Message: The message to be sent by email.

6.12 State Settings

States provide a counter which executes an action or macro every *n*th occurrence of the triggering event. The editors for Actions and Macros have a States tab.

© 2008 Promixis, LLC



State Count: The maximum value of the state counter.

Trigger on State: The state counter value at which the action is executed.

Invert: Tick to execute the action on every state *except* the trigger state.

Current State: Shows the current value of the state counter.

Reset State: Press to reset the state counter.

Usage

The State feature has numerous uses. A typical example is to make a remote button toggle between two, or cycle between several Actions.

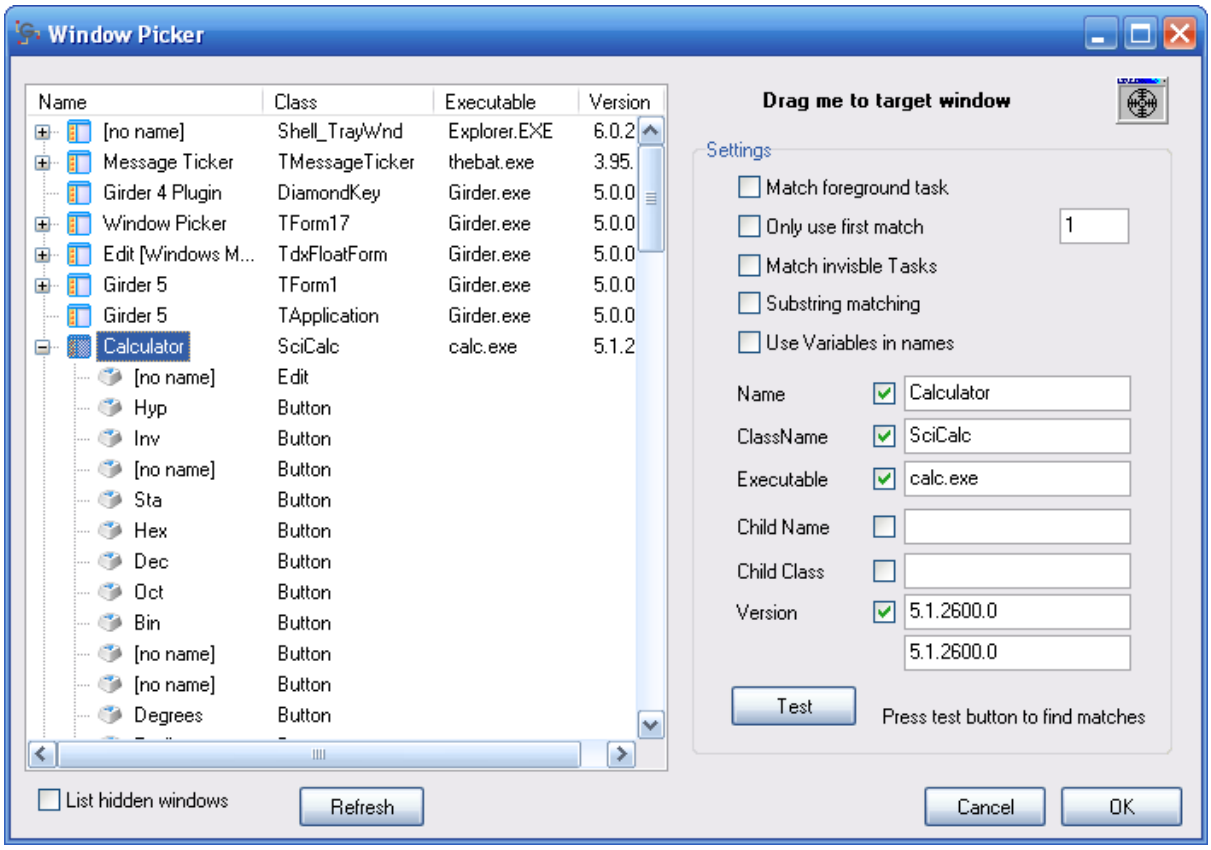
For a toggle, create the two Actions and attach the same Event Node to each (say the **Button Down** event from a remote). For both Actions, set the **State Count** to 2. For the first Action, set **Trigger on State** to 1 and for the second, set it to 2. Press **Reset State** on both Actions.

First time the event is triggered, the state of both Actions is advanced to 2 and the second Action is executed. Next event, the state of both Actions resets to 1 and the first Action is executed. This continues, alternating between executing the first and second Actions.

6.13 Window Picker

Some actions apply to specific windows. Window Picker enables selection of the windows which the action should apply to. Actions which require Window Picker have a button to specify the target window which brings up this dialog.

© 2008 Promixis, LLC



There are two ways to discover the settings applicable to an active window.

- You can locate it in the window hierarchy tree on the left of the dialog and click on it.
- Alternately, you can drag the locator icon from the top right of the dialog over the desired window. Make sure the black outline encloses the client area of the whole desired window rather than a child window.

Care is required when selecting which settings to use as match criteria. These may not uniquely select one window in all circumstances or conversely a change in conditions may mean that the criteria no longer match the desired window. You can use the **Test** button to experiment with different settings. This will report how many current windows match the selected criteria.

Match foreground task: Supersedes all other criteria and matches the current foreground window.

Only use first match: If this is selected, and the other criteria match more than one window, the *n*th window is used as specified in the text box.

Match invisible Tasks: Check this if you wish to match windows that may not be visible.

Substring matching: Check this if the remaining match texts may not be the entire text. For example, the Name of a window may include a file name as well as a program name and you may wish the match to ignore the file name.

Use Variables in names: Check this if the remaining match texts may contain Lua variables (enclosed in square brackets). Girder will attempt to resolve these to substitution text before performing the match.

Name: Matches the window title of the main window.

© 2008 Promixis, LLC

ClassName: Matches the class name of the main Window, which is often unique.

Executable: Matches the name of the program file (exe) which created the main window.

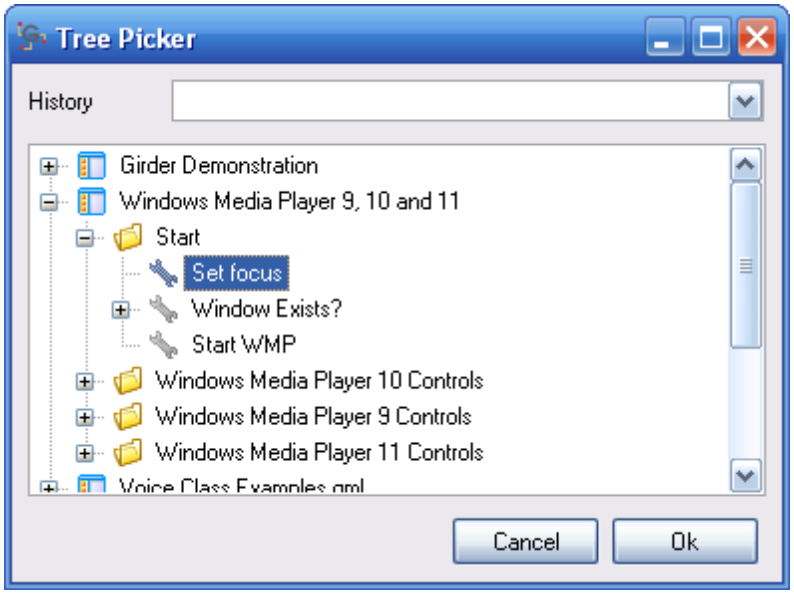
Child Name: Matches the name of a child window to be selected within the main window.

Child Class: Matches the class name of a child window to be selected within the main window.

Version: Matches the file version of the program file or DLL. Take care using this with programs that are frequently updated.

6.14 Tree Picker

The Tree Picker is used to select a node in the Girder tree to be the target of an action. Actions which require node targeting have one or more buttons which bring up this dialog which comprises a copy of the tree.

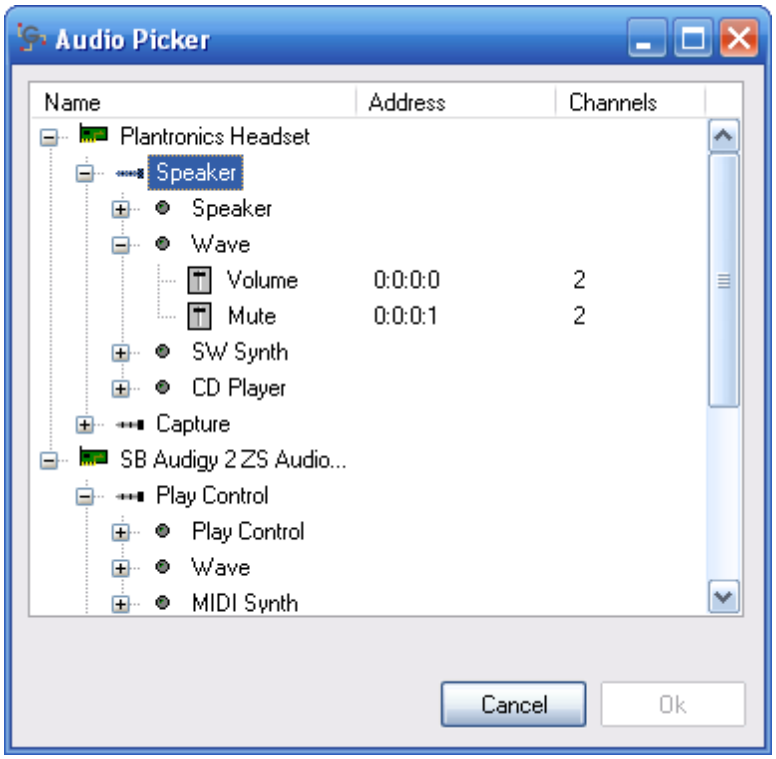


Select a node and press OK. Press **Clear Link** to remove an existing target.

6.15 Audio Picker

The Audio Picker selects an audio control to be the target of an action. Actions which target an audio control have a button which brings up this dialog.

© 2008 Promixis, LLC



TIP: The available controls will depend on the hardware on your computer.

- The root nodes of the control tree represent different hardware audio devices on your computer. Normally there is just the one, but there may be more if, for example, you have a USB headset in addition to the built-in audio codec. This represents the first component of the Address, from 0 for the first device.
- The second level nodes represent the audio destinations. Normally there are two, the Volume Control (sometimes called Line Out or speakers/headphones) and the recording control or recording destination (for recording to your hard disk). This represents the second component of the Address, from 0 for the first destination.
- The third level nodes represent the audio sources that can mix into each destination. The first entry (address -1) comprises the master controls which change all the sources. The remaining entries change the available sources individually, starting at 0 for the first source.
- The fourth level nodes represent the audio controls for each source. Each control changes a different parameter of the audio like the volume or the mute switch. The fourth component of the address starts at 0 for the first control.

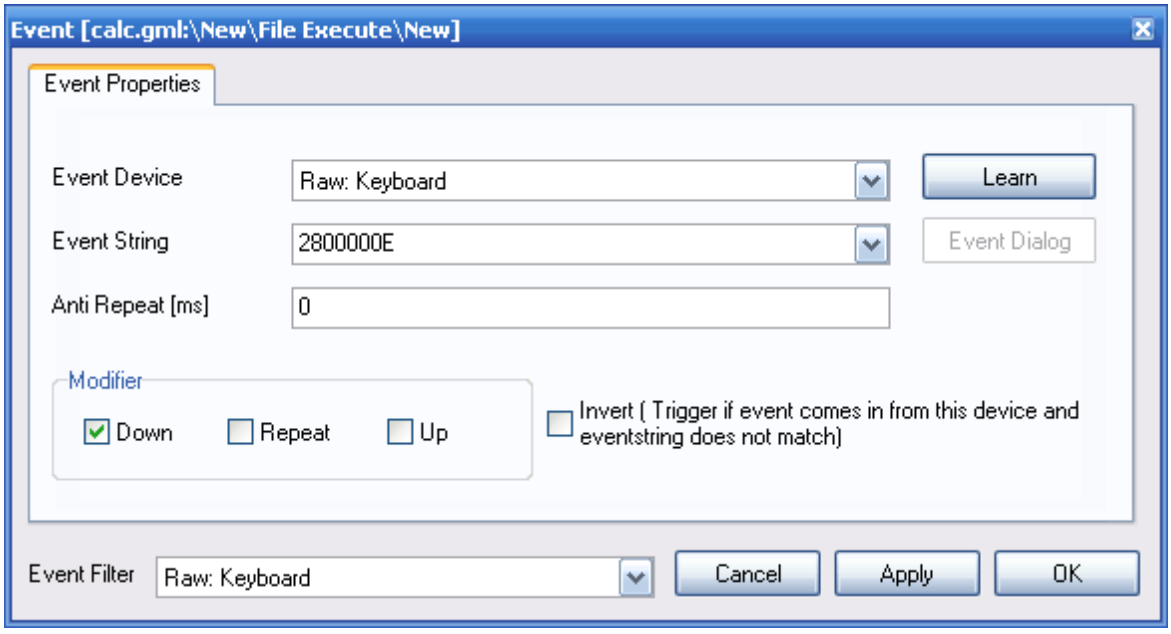
Note that each control also has a **Channel** parameter. A two channel control allows individual settings for the left and right channel of a stereo system, for example. Special Actions and Lua functions are provided to manipulate the channels of two channel controls independently. Furthermore, some controls are **on/off** (like Mute) whereas some are **percentages** (like Volume). Girder provides different Actions and Lua functions for each type.

To select a particular audio control as the target for an Action, highlight it in the tree window and press **OK**. Press **Cancel** to leave the target unchanged.

© 2008 Promixis, LLC

6.16 Event Editor

The Event Editor is accessed by double clicking on an Event node in the Girder Tree or via **Edit** or **Edit in New Window** on the context menu.



Learning an Event

TIP: It is often easier to use the Logger to capture an event and then drag it onto a node in the Girder Tree.

TIP: When recognizing events from remotes or from the keyboard, it is best to use Mapped Events rather than Raw Events as this gives more flexibility should you later change your remote.

Learn: To learn an Event using the Event Editor, press this button and then take whatever action is necessary to generate the Event. It may help to set the **Event Filter** first to avoid seeing random events that happen coincidentally. When the correct Event Device and Event String is showing, press **OK**.

TIP: When learning shifted keyboard events like **Alt-F1**, a problem arises in that you have to press **Alt** first and then **F1**. This means that the event string for **Alt** on its own will appear before and after the event string for **Alt-F1**. The three distinct event strings are registered in the drop-down list and in this case the **middle** of the three is the one you should select.

Selecting an Event Manually

Event Device: Select the device (Plugin, Mapping Device or Girder) which is expected to generate the event.

Event String: Enter the Event String you wish to recognize. The Event Device may provide a drop-down pick-list, or it may provide an Event Dialog in which case the button of that name will be enabled. In either case, an Event String is picked and entered into this field.

Modifier: This applies to keyboard Events and some remote controller Events. When you press the key, an Event is generated with the **Down** modifier. If you then hold the key down beyond a set time, a stream of Events with the same event string but the **Repeat** modifier are

© 2008 Promixis, LLC

generated at a set rate. When you release the key, another Event is generated, still with the same event string, but now with the **Up** modifier. You can set the Event Node parameters to respond to Events with none, any or all of the three modifiers.

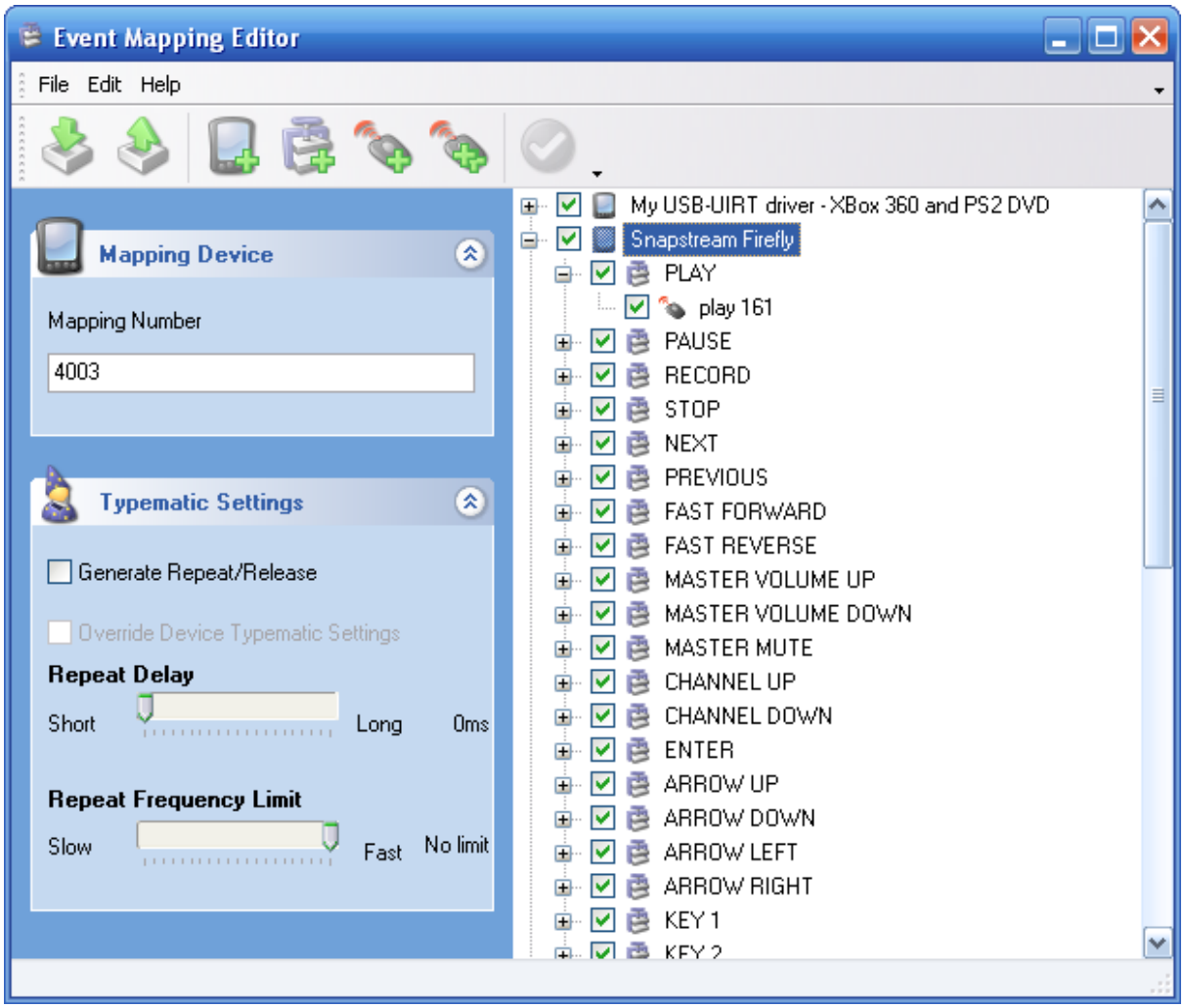
Invert: It is possible to have the Event respond to all events from a given device *except* the stated one.

Anti Repeat: This sets a guard interval in milliseconds after recognizing an event during which further recognized events will be ignored. This may be useful in eliminating overactive repeat events from some controllers.

6.17 Event Mapping Editor

An **Event Mapping Device** translates **Input** or **Raw** Events from Plugins (generated by the keyboard or by a remote controller) into **Output** or **Predefined** Events used by Program Definition GML files. Multiple Input Events can generate the same Output Event and you have complete flexibility over which key controls which function.

The Event Mapping Editor (**View** menu, **Event Mapping** or **F6**) is used to create or or modify Event Mapping Devices.



Toolbar

© 2008 Promixis, LLC

 Import a map from a MAP file.

 Export the selected map to a MAP file.

 Add a new Mapping Device.

 Add a new Output Event to a Mapping Device.

 Add a new Input Event to an Output Event.

 Add multiple Input Events to an Output Event.

 Apply changes.

Mapping Device Node

Any number of Mapping Device nodes may be defined, usually one per controller device, but this is not necessary. A device node can accept Input Events from any device, or several Mapping Device nodes may be used to map different functional groups on the same device.

Mapping Number: It is not normally necessary to change this. It sets the pseudo Device Number for the Mapping Device and may be useful recreating a Mapping Device.

Typematic Settings

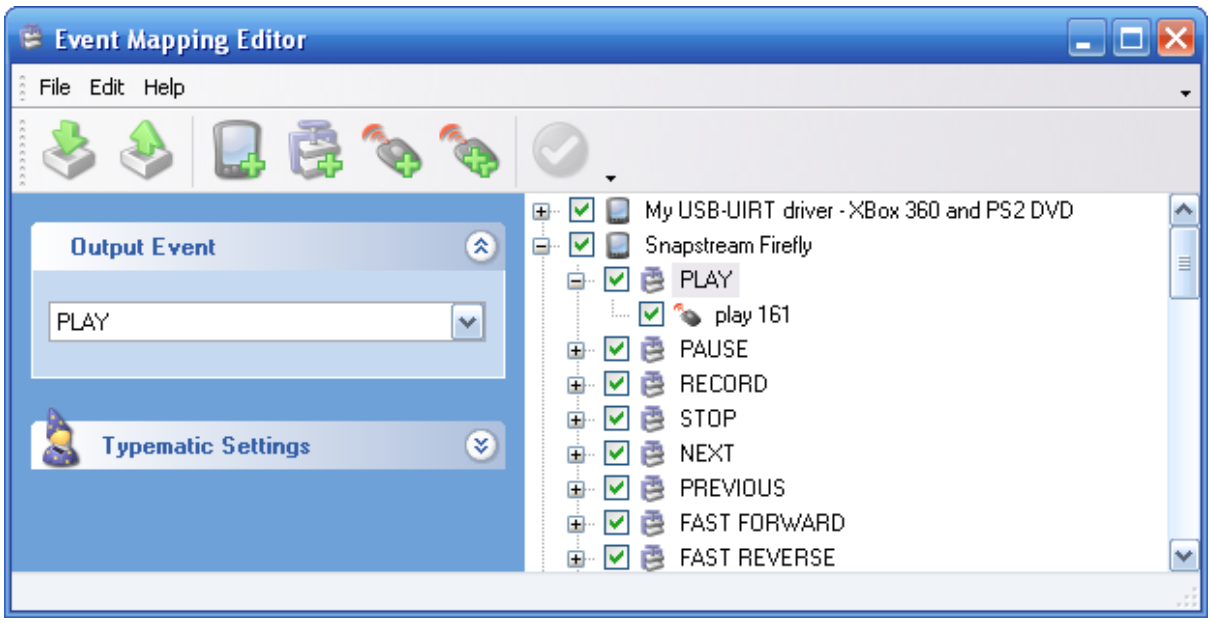
Generate Repeat/Release: Tick to generate Mapped Repeat and Release events even if the Raw event does not generate these.

Override Device Typematic Settings: Tick to suppress Raw Repeat messages and regenerate them according to the settings below.

Repeat Delay: Time in milliseconds from the Down event to the first Repeat event.

Repeat Frequency Limit: Maximum frequency at which Repeat events are generated.

Output Event Settings

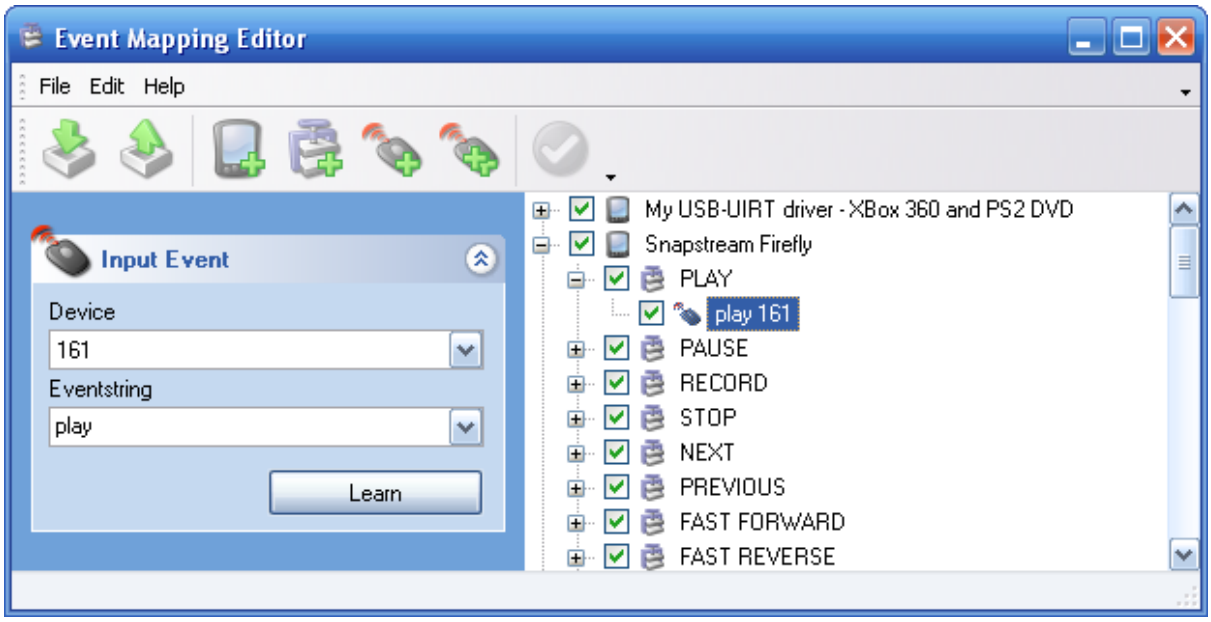


© 2008 Promixis, LLC

To add an Output Event to a Mapping Device, highlight the Mapping Device and press the **Add Output Event** button or **Add Output Event** on the Edit menu.

Output Event: Select the Standard Event to be generated.

Input Event Settings



To add an Input Event to an Output Event, highlight the Output Event and press the **Add Input Event** button or **Add Input Event** on the Edit menu. Alternatively, press the **Add Input Events** button to learn several Raw Events at once.

Device: Select the device which will generate the Raw Event.

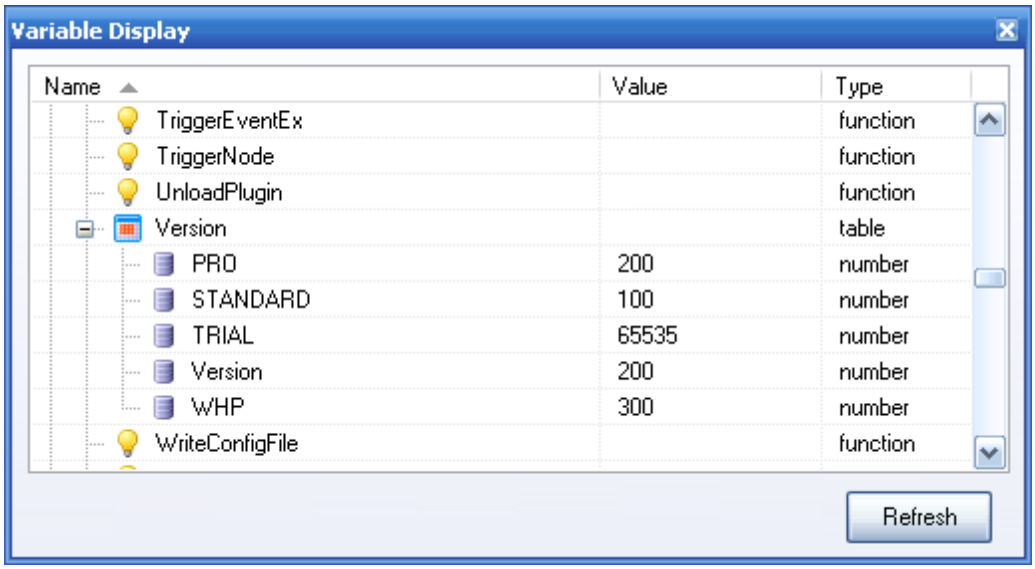
EventString: Select the event string of the Raw Event.

Learn: Press the learn button and then generate the Input Event (for example, press the remote button) to automatically fill in the above settings.

6.18 Variable Inspector

The variable inspector (**View/Variable inspector** or **F12**) shows all Lua global variables (see Scripting Girder).

© 2008 Promixis, LLC



Refresh: Press to update the list (if a new global variable has been added or values have changed).

Sort: Sort on any column by clicking on the column header, once for ascending order, twice for descending. The small triangle indicates the sort column and order.

Entries in the Variable Inspector use the following icons.

A Numeric, String or Boolean variable.

A Function.

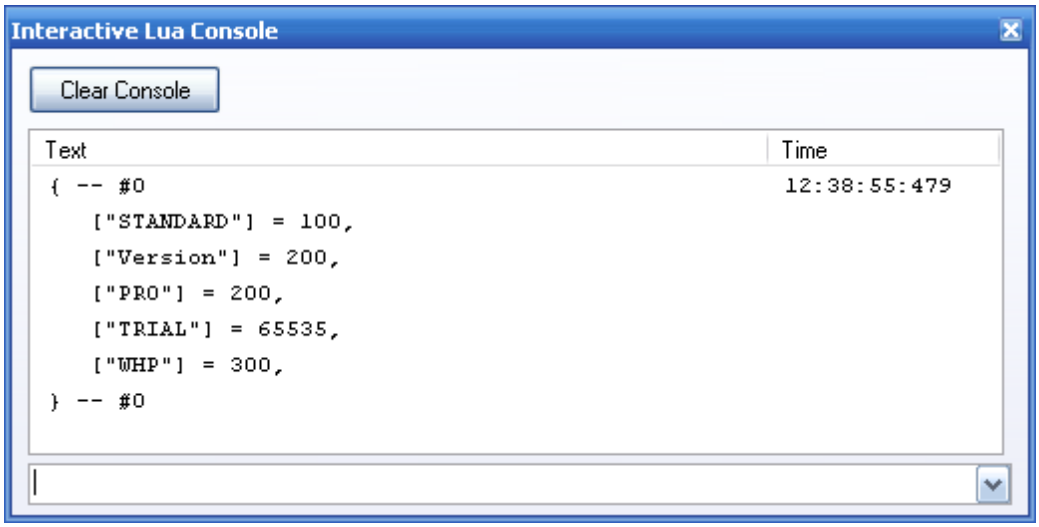
A Table. Press the + or - button to show or hide the contents.

Tip: You can explore Libraries of functions (they appear as tables) as well as variables in the traditional sense.

6.19 Interactive Lua Scripting Console

The Interactive Lua Console (**View menu** or **F7**) shows error messages and the result of Lua **assert** and **print** commands. You can also type Lua statements in the box at the bottom and execute them immediately. (See Lua Language Reference.)

© 2008 Promixis, LLC



Clear Console: Clear the console window.

6.20 Command Line and Shortcuts

Command Line

The main Girder application, girder.exe can take a GML path and filename as a parameter. This file will then be loaded in addition to the normal set of GML files. There is also one non-positional switch **-startup** which is the command line equivalent of the "Delay Plugin loading on Startup" option on the Settings General tab.

```
"C:\Program Files\Promixis\Girder\Girder.exe" "C:\Start.gml" -startup
```

The command line for the the Event Generation applications, **event.exe** and **csevent.exe** are described in the Command Line and COM Event Generation topic of the Events Reference section.

Shortcut Keys

Key	Function	Key	Function
F1	Show Help	F4	Show Logger
F5	Test Current Action	F6	Show Event Mapping Editor
F7	Show Lua Interactive Console	F8	Show DUI Form Designer
F9	Disable/Enable events	F10	Software Updates
F11	Reset Lua Scripting	F12	Lua Variable Inspector
Ctrl+N	Create new GML file	Ctrl+O	Open GML File
Ctrl+S	Save current GML file	Ctrl+Alt+S	Save all GML files
Shft+F1	Add Group	Shft+F3	Add Macro
Shft+F4	Add Event	Shft+F6	Add Macro Event
Ctrl+F4	Close Window	Ctrl+Del	Delete current node
Ctrl+Alt+X	Cut selected node	Ctrl+Alt+C	Copy selected node
Ctrl+Alt+V	Paste node	Ctrl+Alt+D	Duplicate selected node
Ctrl+Alt+F	Find node		

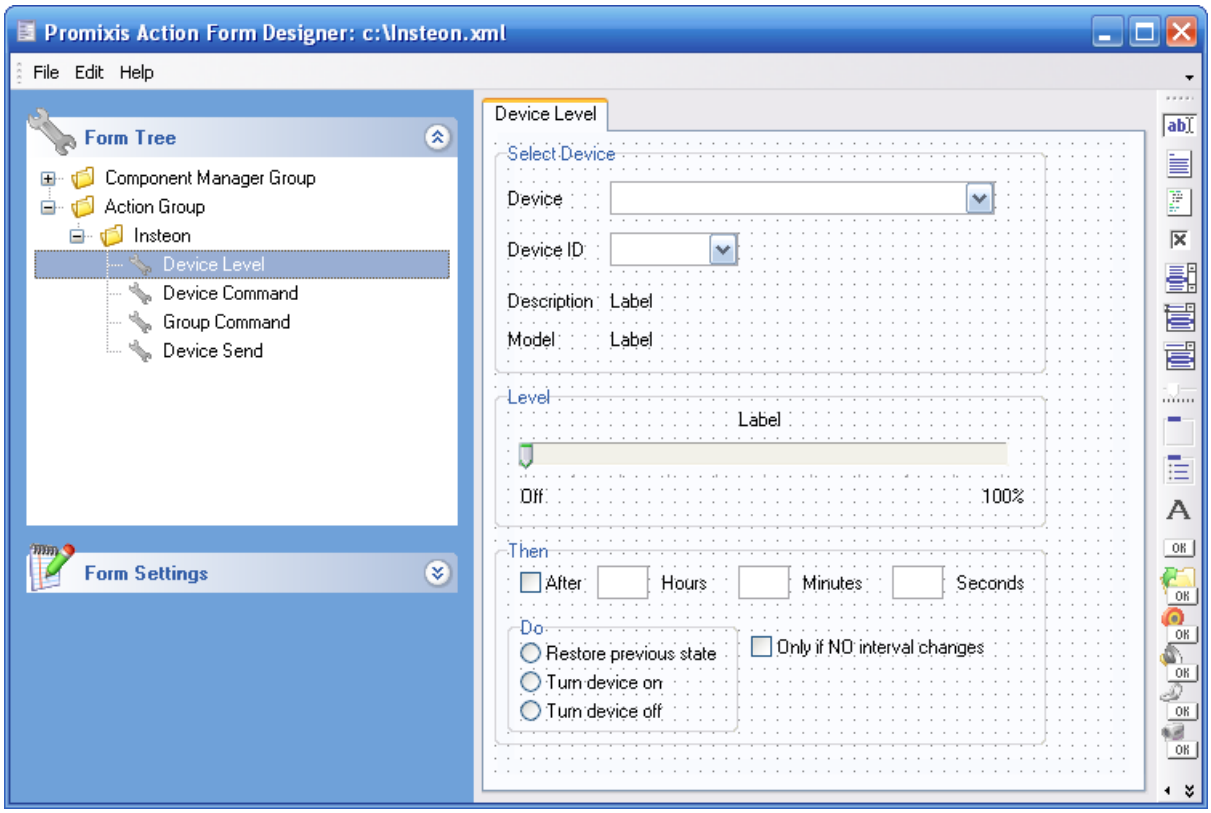
© 2008 Promixis, LLC

6.21 Dynamic User Interface Designer

The DUI Designer is a separate application used to edit Dynamic User Interface (DUI) XML files, this is not needed for normal use of Girder, only if you are interested in creating new actions for Girder would you use this.

The Action Editor, Conditional Editor and Configuration Pages are defined by these files. For the most part, each Plugin has a file with the dll extension in the **%GIRDER%\plugins** directory and an xml file in the **%GIRDER%\plugins\UI** directory. The dll file is executable code which installs the xml files in the main Girder application to define the interface. With one exception the DUI Designer is of use only to Plugin developers. The exception is the Tree Script Plugin which enables Actions, Conditionals and Configuration Pages to be defined in Lua script and DUI xml.

The DUI Designer is only available if the **Install Developer Files** option is taken during Girder installation. To start the application, use the operating system **Start** menu **Programs[B]**, **Promixis**, **Girder** by default.



The **Form Tree** task box has root nodes representing the UI category. These nodes can be added from the **Edit** menu as follows.

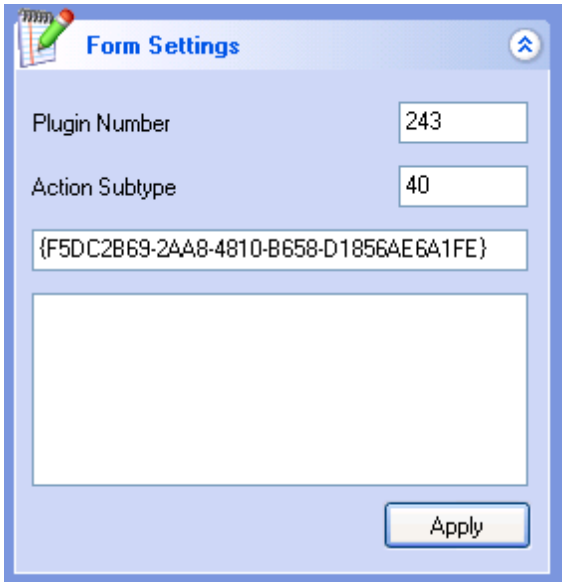
-
- Add Action Group. The tree below this node gets added to the Actions task box in Girder.
- Add Conditional Group. Gets added to the Conditionals task box in Girder.
- Add Device Config Group. Gets added to the Settings Dialog Main Settings.
- Add Automation Group. Gets added to the Settings Dialog Automation Settings.

The tree below the Action Group or Conditional Group root nodes echoes the structure of the Girder **Actions** or **Conditionals** task box. Group nodes are added in the same way as on the

© 2008 Promixis, LLC

main Girder Tree. Individual DUI forms can then be added to a selected Group or at the top level.

The **Form Settings** task box sets the **Plugin Number** (243 for the Tree Script Plugin) and the **Action Subtype** which is a unique identifier for the Action or Conditional within the Plugin.



The system also assigns a Globally Unique Identifier (GUID) to groups and forms which is the long hexadecimal string below the Action Subtype. This is what correlates the same group in different DUI files - if the Groups have different GUIDs multiple groups will show with the same name. There are menu options for adding groups with the same GUID as the standard Action groups shipped with Girder (**Edit** menu, **Default Groups**).

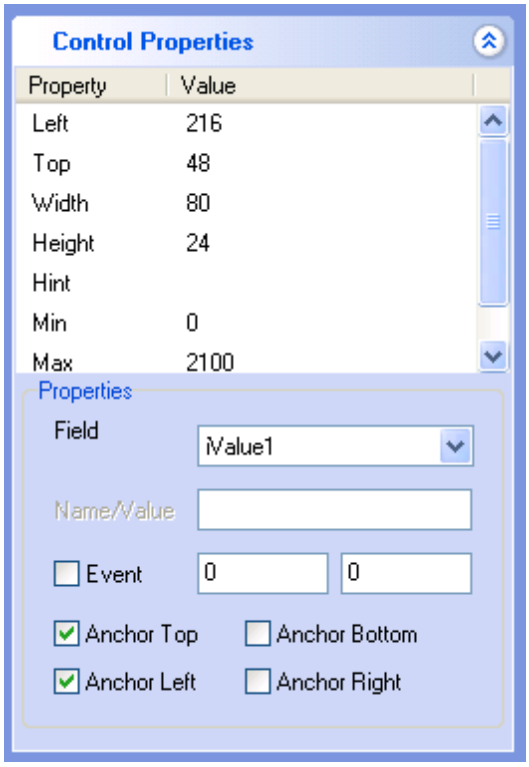
The vertical toolbar on the right contains the set of controls that may be added to the form designer.

- Edit Box - Edits a string containing a single line of text.
- Memo - Edits a string containing multiple lines of text.
- Code Editor - Edits a string containing a Lua Chunk.
- Check Box - Edits a boolean.
- List Box - Picks a string or number from a list of labels.
- Edit List Box - Picks a string or number from a drop-down list of labels.
- Combo Box - Edits a string or picks examples from a drop-down list.
- Slider - Edits a number using a slider control.
- Group Box - Labeled visual container for other controls.
- Radio Group - Picks a number using a set of labeled buttons.
- Label - Text label.
- Button - Button to fire an event.
- Browse Button - Button to select a file and path.
- Target Button - Button to select a window using the Window Picker.
- Volume Button - Button to select an audio control using the Audio Picker.
- Link Button - Button to select a tree node using the Tree Picker.
- Capture Button - Button to show the Command Capture dialog.
- Spin Button - Edits a number using up-down buttons.

© 2008 Promixis, LLC

- Directory Browser - Button to select a directory path.
- HTML Label - Label with HTML markup.
- Image - Label showing an Image File.

When a control is highlighted, the **Control Properties** task box allows its properties to be set.



The **Field** property controls the binding of the control to data storage or transfer variables. The options are as follows.

1. An action parameter. These are sValue1..3, bValue1..3, iValue1..3 (strings) and lValue1..3 (numbers - integer). Edits made in the editor are automatically stored in the GML file. If used for Configuration forms, the values are not automatically stored.
2. A Windows Registry Key (either number or string). This is used primarily for configuration pages and will automatically update the identified registry key. The registry key is identified in full in the box below, for example "HKEY_LOCAL_MACHINE\SOFTWARE\Promixis\Girder\4 \Plugins\Example\TextBox1". To automatically get the proper registry key use "HKEY_AUTO \PATH_AUTO\Plugins\Example\TextBox1" and use gir.GetRegistryHive() and gir. GetRegistryPath() from lua.
3. A Lua Variable. The control takes its default from, and writes its response to, a Lua variable. The Lua name of the control is set in the designer. When a form is showing, the Lua variable can be accessed at dui.Forms[index].Controls.name.field. The field names are given in the Tree Script Plugin section.

The **Event** tickbox specifies that the Plugin should be notified when any of the fields of the control change. The two numbers to the right of the checkbox are passed through to identify the event.

© 2008 Promixis, LLC

Part

7 Examples

This section provides worked examples of various Girder techniques.

- Starting and Stopping an Application shows how to write Actions to start and stop an application using a single button on a remote.
- Using Keyboard Spoofing shows how to send keystrokes to an application from a remote as if they had been typed on the keyboard. It also shows how to use Window Conditionals to control multiple applications from the same controller.
- Using Command Capture is an example of using the Command Capture Action to control a program feature that does not have a short-cut key.
- Running a Macro when a Program Starts shows how to run a series of Actions in a Macro whenever a program starts.
- Creating an Action using Tree Script is an advanced example which creates an Action using scripting and the Action Form Designer. The Action shows a balloon hint on the Girder notification icon.
- Transport Tutorial shows you in detail how to create lua scripts for your own devices.

7.1 Starting and Stopping an Application

Introduction

In this example we will create a GML file from scratch. The file we create will enable the Windows Calculator to be started and stopped using a remote control device.

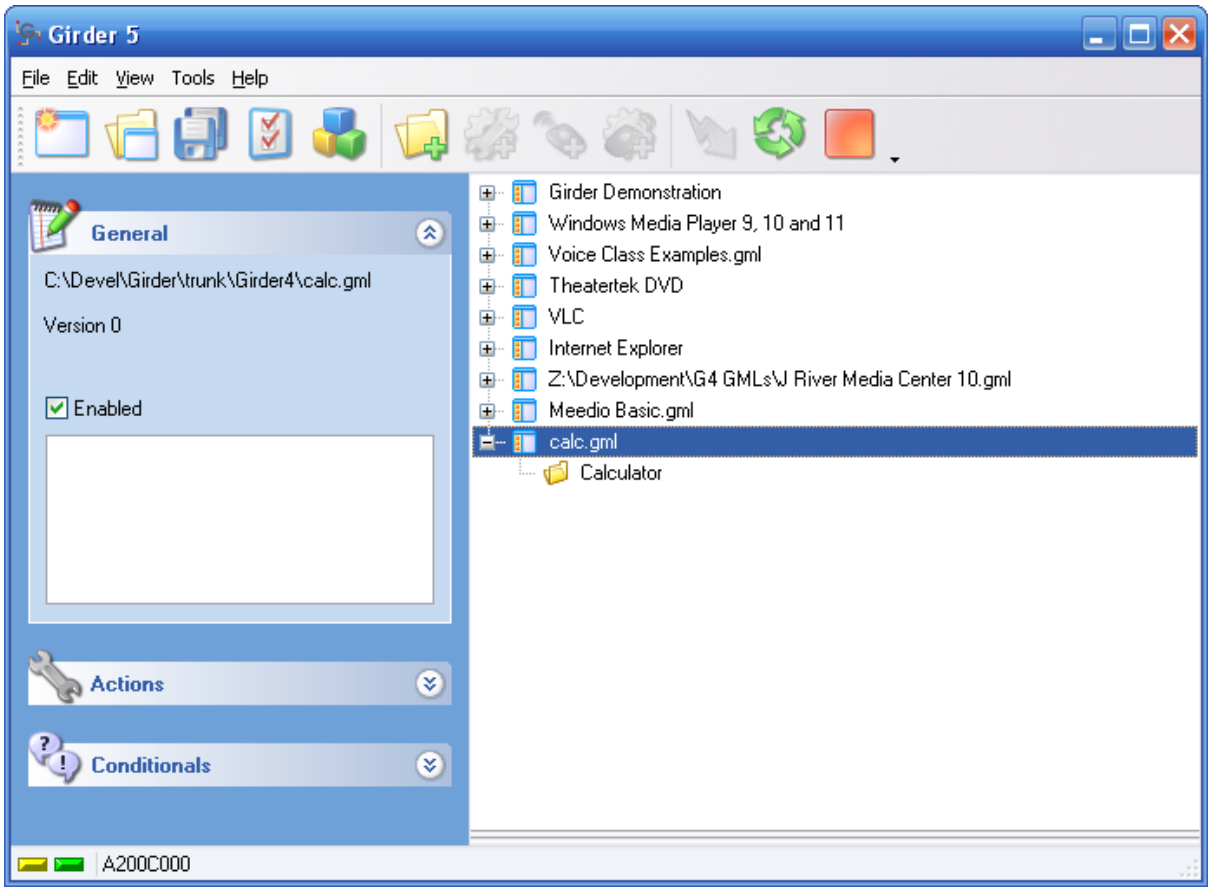
Creating the File and Group

If necessary, use the **View** menu to switch to **Expert** mode. This interface gives access to the full power of Girder.

Note the tree display top right of Girder. This is where actions are defined and associated with events. First collapse any existing trees to the top level nodes, which represent individual GML files. Now select each top level node in turn and then un-tick the **Enabled** tick box on the **General** task box in the left pane. This will simplify things by disabling all actions except the ones we are about to define.

Create a new GML file using the **Create New File** toolbar button and call it "Calc.gml". Click on the "New" folder name and change it to "Calculator".

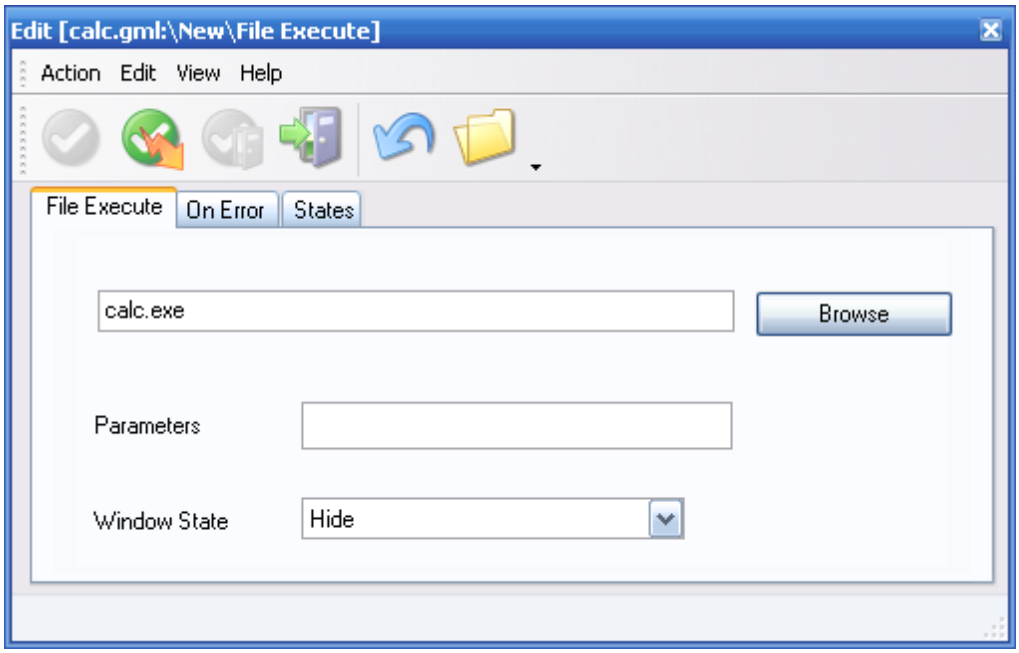
© 2008 Promixis, LLC



Starting the Program

We will add an action to start the Calculator program. Find the **File Execute** action in the **Actions** pane (it is in the **OS** folder) and drag it onto the "Calculator" folder. In the action editor, enter "calc.exe" in the text box beside the **File To Execute** button.

© 2008 Promixis, LLC

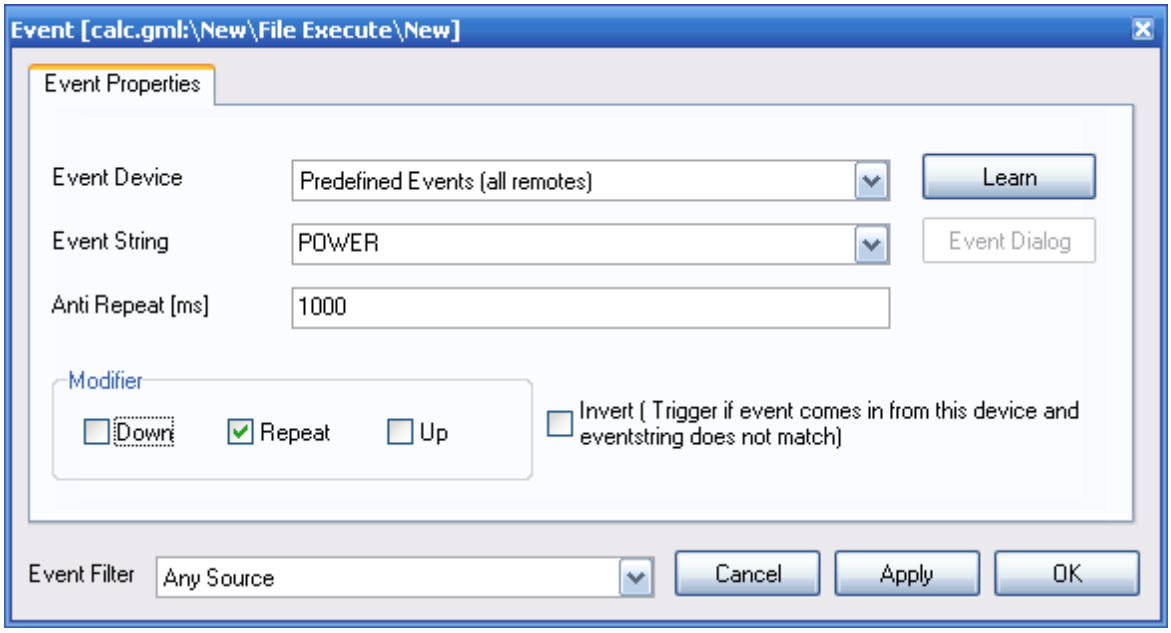


You can test this action by highlighting it and pressing the **Test Action** toolbar button.

TIP: A common mistake in Girder is to forget to Apply changes in the Action or Event editors. Always press the apply button when you have changed the Event or Action parameters.

We will now make this action happen in response to a remote event. With the action selected, click the **Add Event** button on the toolbar. In the Event editor, set the Event Device to "Predefined Events (all remotes)" and the Event String to POWER.

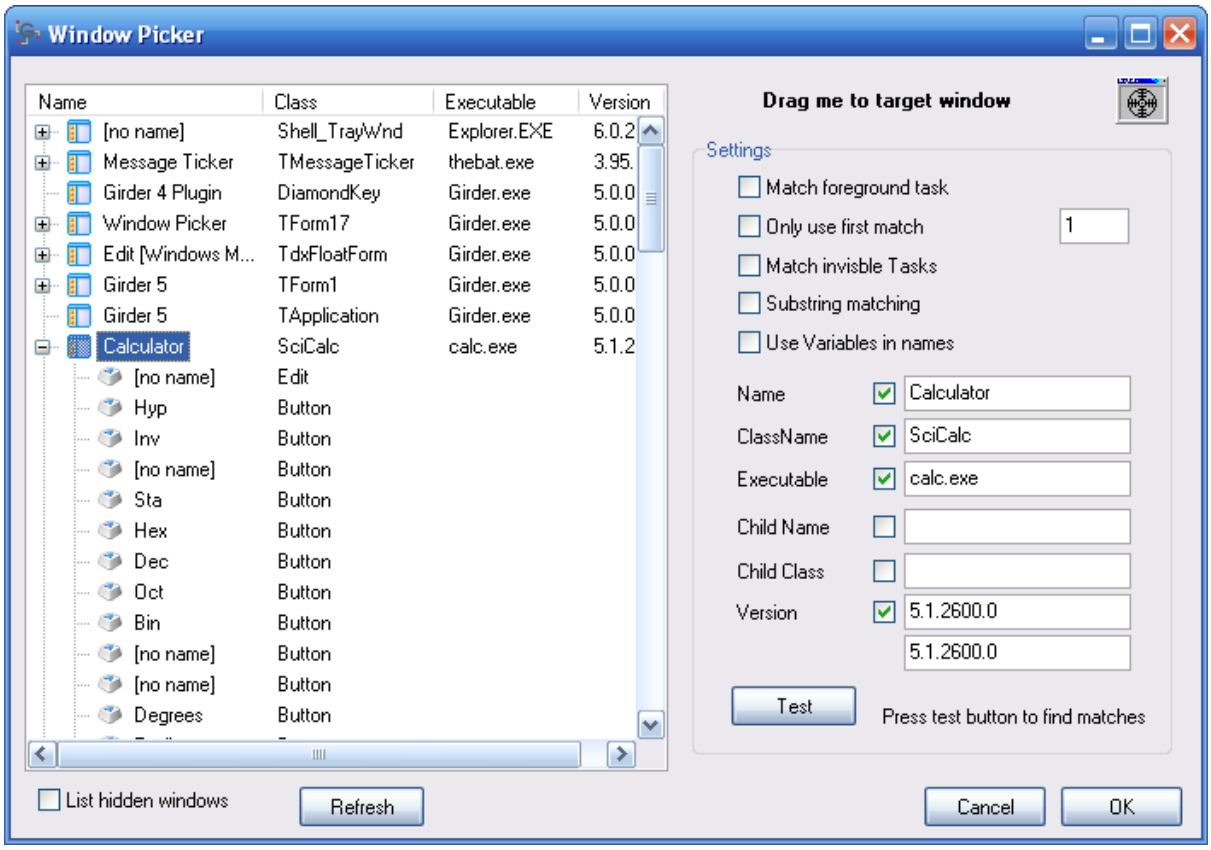
Press the **Apply** button.



© 2008 Promixis, LLC

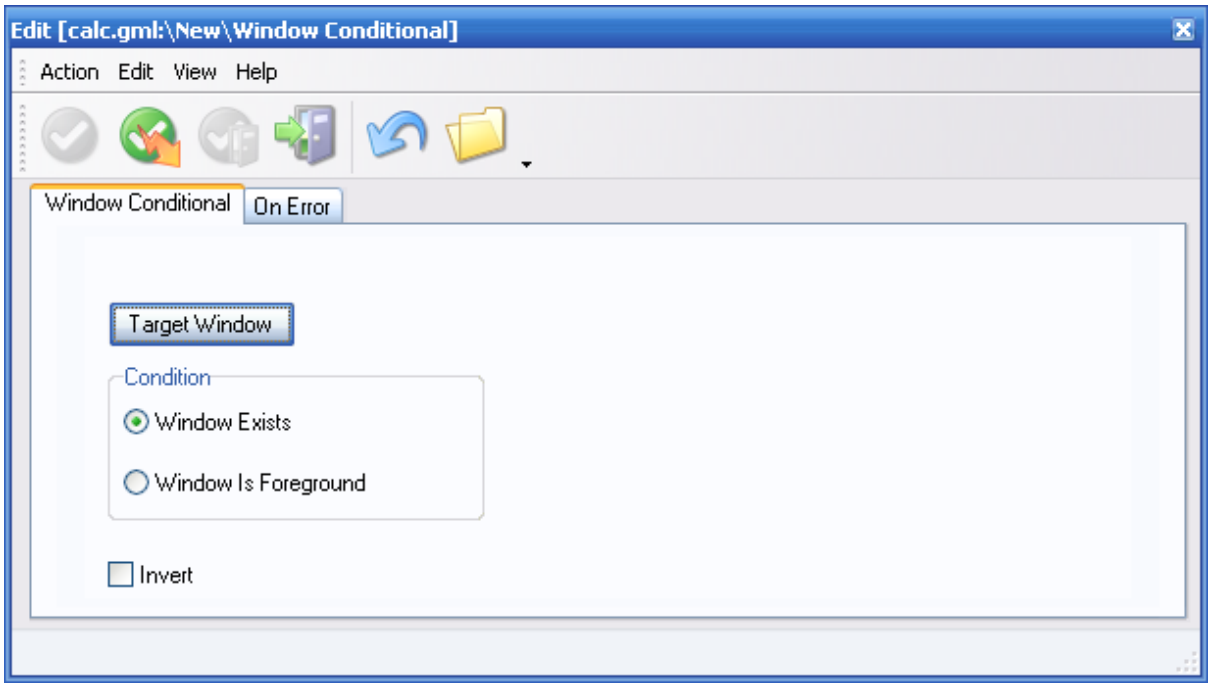
Now you should be able to start the calculator by pressing the power button on your remote. However, there is a problem: press the key repeatedly and multiple copies of the calculator appear, which is probably not desirable. We can fix this using conditionals.

Make sure exactly one copy of Calculator is running and then drag a **Window Conditional** from the **Conditionals** task box onto the File Execute action. On the Conditional editor, press the **Window Picker** button. This brings up the Window Picker dialog. Click the Calculator entry in the list of windows to the left. Alternatively, drag the little window icon top right of the dialog over the calculator itself and drop it (safest place to drop is over the title bar as this selects the top-level window and not one of the control windows). In either case, next press the **Test** button and ensure it reports "One Match".



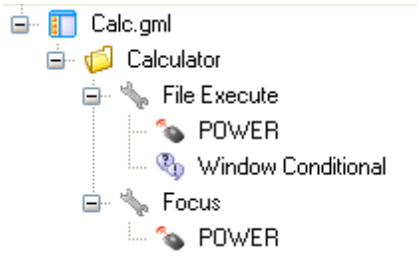
Now close the Calculator window and then press **Test** again – it should report "No Matches". Press **OK** to dismiss the Targeting dialog. On the Conditional editor, ensure the condition is set to **Window Exists** and tick the **Invert** box.

© 2008 Promixis, LLC



Remember to press the **Accept** button! This conditional will be true when the Calculator window does **not** exist, and the action will only be done when this is the case. Test this by pressing the power button several times and noting that only one Calculator is started.

This fixes the immediate problem, but it would be nice if pressing the Power button when the calculator was already running ensured it was focused and ready for use. To achieve this, first drag a **Focus** action from the Windows group onto the "Calculator" folder. On the Action Editor, press the **Target Button** and, as before, set the target window to the calculator. Right-click the Event node on the File Execute action and copy it. Right click on the Focus action and paste a copy of the same event onto this. A Window Conditional is not needed on this action, because Focus will have no effect unless the target window is present.



Now the Power button will make the calculator ready for use if it is not running, is minimized to the taskbar or is hidden under other windows.

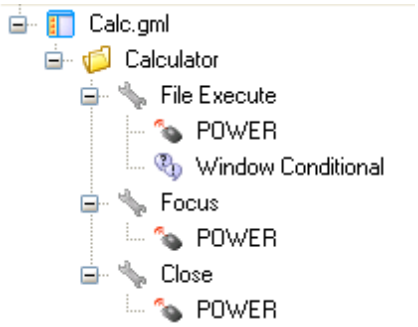
Stopping the Program

Usually, the Power button on a remote is a toggle, but we will make it so the user holds the power key down to exit the calculator program. We have already used the action of pressing the button down to focus the application. If the button is held down for any length of time, Repeat events will be generated and it is these we will use to exit the program.

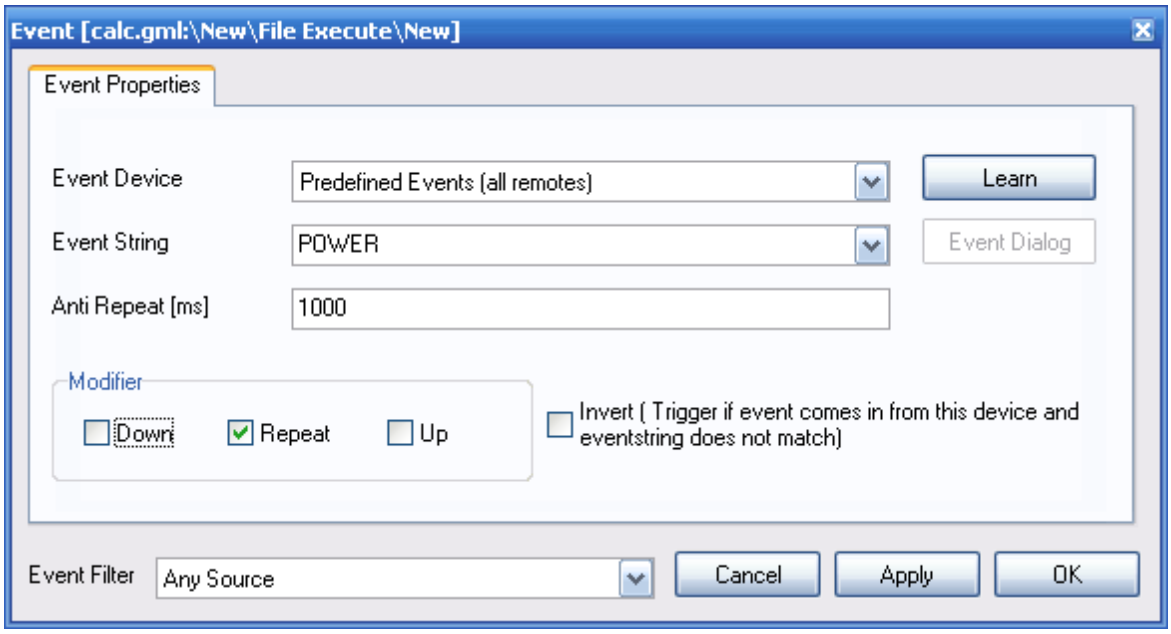
Drag a **Close** action from the Windows actions group onto the "Calculator" group. In its editor,

© 2008 Promixis, LLC

set the window target to the calculator window as before. Select this and add an event. Set the event to the POWER event as before, but this time un-check the **Down** modifier and check **Repeat** instead.



Should the action prove too sensitive (it would be difficult to press the key long enough to start the calculator but not so long as to immediately stop it again) you can set the Anti Repeat on the event. For example, if you set it to 1000 the button will need to be held down for at least a second to trigger the Stop action.



7.2 Using Keyboard Spoofing

Introduction

Most programs are designed to be controlled by keyboard and mouse. Girder is able to fake or *spoof* keyboard input using the Keyboard Action. In this example, we will control the Windows Calculator from a remote using this technique. You can continue with the file created in the last example or start with an empty one.

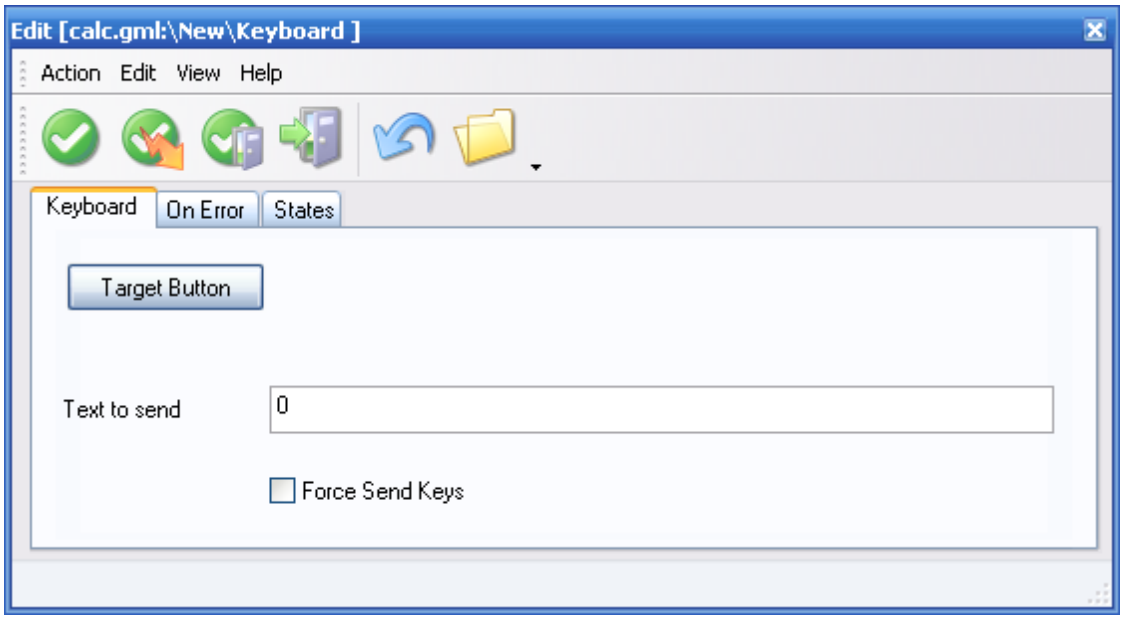
Connecting the Buttons

We want the number buttons on the remote to work the number keys on the calculator. The calculator also responds to keyboard number keys, so we will "spoof" these.

© 2008 Promixis, LLC

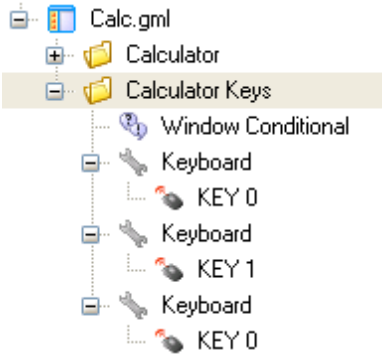
First, add a new group and call it "Calculator Keys". Now drag a **Window Conditional** onto this group and set its target to the calculator window. Set it to Condition **Window is Foreground**. This ensures that the commands will only work when the calculator window is the foreground window, enabling the number buttons to work in other programs too.

Next, drag a Keyboard action onto the group ensure its Window Target is **Match foreground task** (this is the default for new Actions). Set **Text to Send** to "0" (without the quote marks).



Right click on this event and Copy it. Highlight the "Calculator Keys" group and use **Ctrl-Alt-v** repeatedly to paste nine copies. Edit each in turn so one sends each of the numerals 0..9. See Keyboard Action in the reference manual for the full details of available key strings.

Select each action in turn and click the toolbar button to add an event. Set the **Event Device** to **Predefined Events (all remotes)** and the **Event String** to **KEY 0** up to **KEY 9**.



Similarly, you can train additional keys, perhaps CHANNEL UP for plus, CHANNEL DOWN for minus and OK for equals.

Tip: Keyboard Spoofing works fine in the Calculator application, but it can be tricky. If it does not work immediately, there are some things you can try. First try with Force Keys checked and unchecked. If that does not work, try narrowing the Window Target - often targeting the child window into which text is typed will work while targeting the application window will not. If, as

© 2008 Promixis, LLC

in this case, you only need to send keystrokes to the foreground window, try checking **Match foreground task** in the Targeting dialog. (This depends on having the Window Conditional set as described above, otherwise the keyboard spoofing would affect whatever window was in the foreground at the time.)

7.3 Using Command Capture

Introduction

Keyboard Spoofing as a means of controlling an application was introduced in the previous example. This will only work if the function you want to control has a shortcut key. In this example we illustrate an alternative technique, **Message Spoofing**.

This example also uses the Windows Calculator and you can either build on the GML file created in the previous example, or begin a new file.

Getting Started

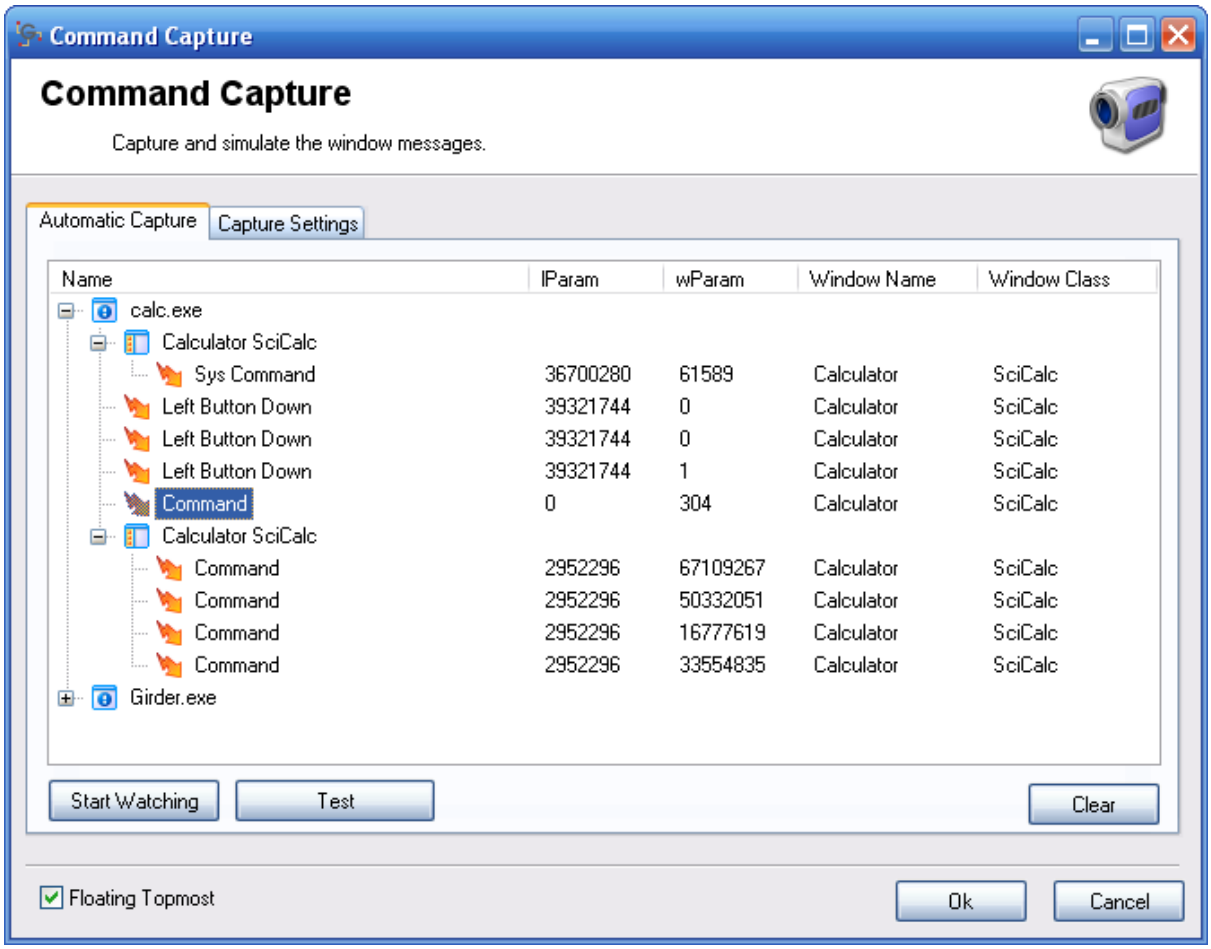
Add a folder and rename it "Calculator Messages" (either to a new GML file or to the one created in the previous example).

Capturing a Command Message

Drag a **Command Capture** action onto the "Calculator Messages" group. Make sure the calculator is in **Standard** mode (view menu). Now press the **Capture Settings** button and then the **Start Capture** button on the Command Capture dialog.

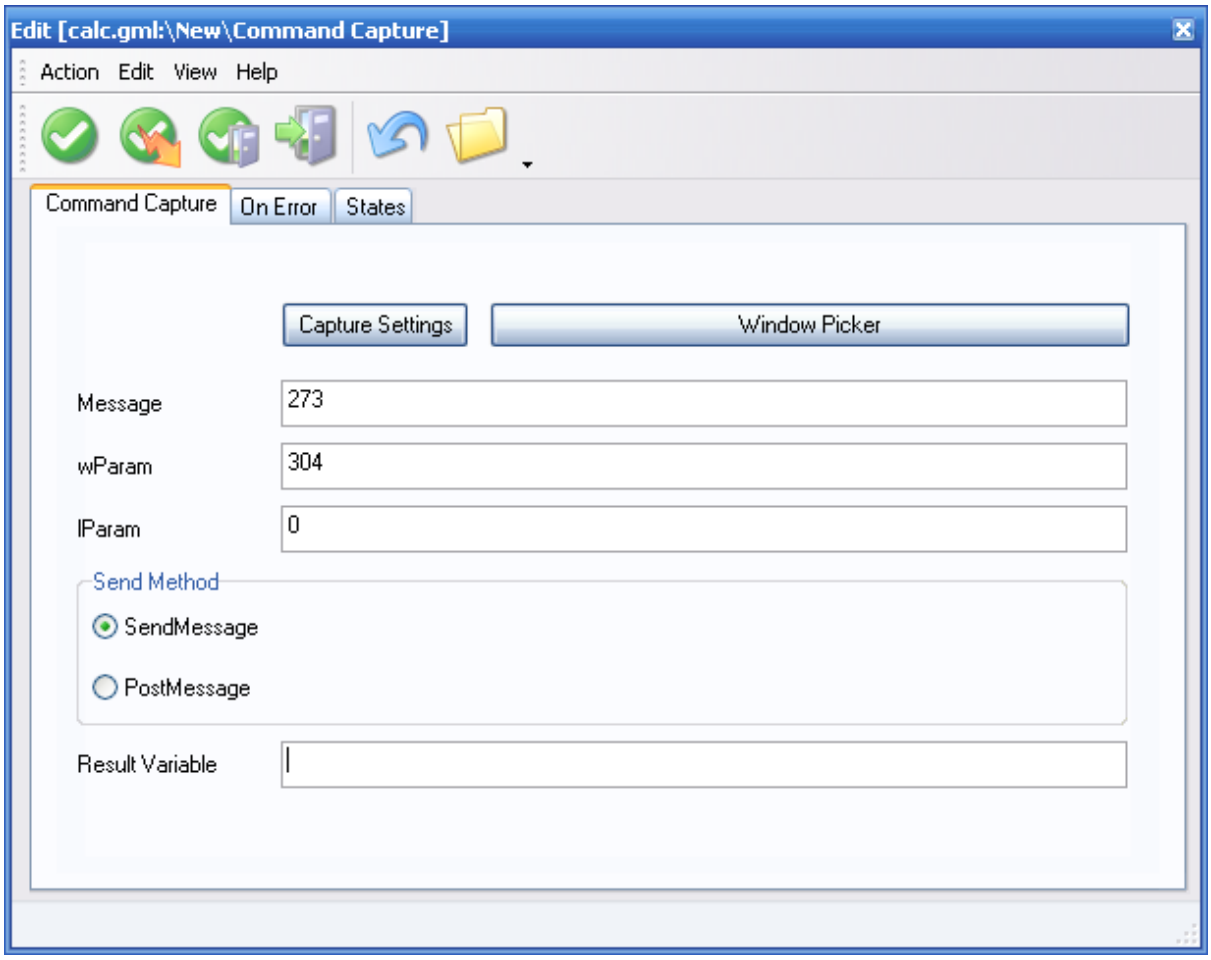
Now switch the Calculator to **Scientific** mode (view menu) and then press the **Stop Watching** button on the Command Capture window. Expand the Calculator nodes in the Command Capture window and look for a message of "Command" type.

© 2008 Promixis, LLC



As there are several "Command" messages, we will test them in turn to see which does what we want. Highlight the first command and then switch the Calculator back to **Standard** mode using its view menu. Now press the **Test** button on the Command Capture dialog. This should switch the Calculator back to Scientific mode. Press **OK** and note that the message settings have been automatically entered into the action editor.

© 2008 Promixis, LLC



If you press the **Window Picker** button you can see that the target window has also been set automatically.

Switch the Calculator back to standard mode, highlight the Command Capture action in Girder and press **F5** (Test Action). The Calculator should switch to scientific mode.

You can hook this Action to another Remote button.

7.4 Running a Macro when a Program Starts

Introduction

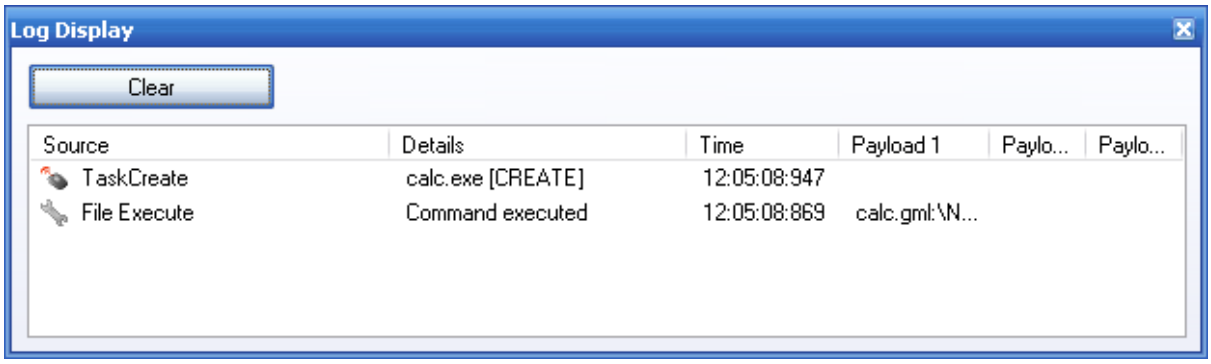
In previous examples, we have used events generated by user operation of remotes. Girder offers many other kinds of events and in this example we will use one of these to take action whenever an application starts. We will also use Macros for the first time in order to execute a short sequence of actions.

The TaskCreate Plugin

Start by making sure the TaskCreate plugin is enabled (ticked in the list in the Plugins section of the Settings dialog).

Next start the Logger (**F4**) and then start the Windows Calculator. The Logger should show an event which was generated when calc.exe started.

© 2008 Promixis, LLC



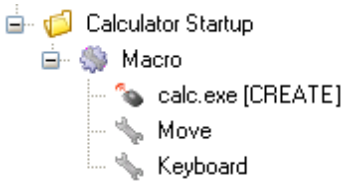
Creating a Macro

Create a new Group and name it "Calculator Startup" (either continue with the file from the previous examples or create a new one). Highlight this group and click on the Add Macro toolbar button to add a new Macro.

Drag the TaskCreate Event from the Logger onto the Macro to create an Event Node. This Macro will now execute whenever the calculator starts.

Drag a **Move** action from the Windows group onto the macro. Set the Target to **Match Foreground Task** (since the calculator has just started it will be the foreground application). Set **X** and **Y** to 0.

Drag a Keyboard action onto the macro (note that you can drop the action into position within the macro since the order of actions within the macro matters). Again, set the target to **Match Foreground Task**. Enter "*v" in **Text to send** (this means **Ctl-v**, the shortcut for paste).



Every time the calculator is launched, it will be moved to the top left of the screen and any numbers on the clipboard will be pasted into the register.

7.5 Creating an Action using Tree Script

In this example, we will create some brand new Actions which can be used in Girder Trees exactly like any other Action and which have their own Action Editors. The Actions will generate a cool balloon hint on the Girder notification icon.

The Tree Script Plugin

We are going to use the Tree Script Plugin, so visit the Settings dialog and make sure it is enabled.

Creating the Action Editor

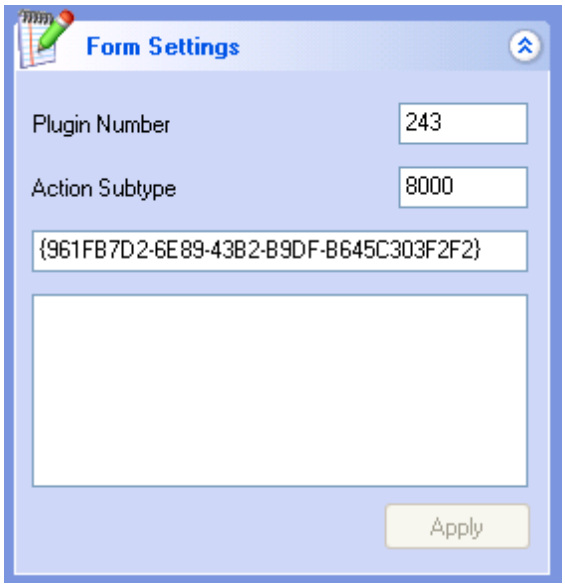
First lets create the Editor Forms. Start the Action Form Designer (it is a separate application and there is a shortcut on the Start Menu) . Create a new file called **Balloon.xml** in the **%GIRDER%\plugins\UI** directory. On the **Edit** menu, click **Add Action Group**. In the **Form Tree**,

© 2008 Promixis, LLC

right click the "Action Group" and **Add Group**, change the name of the new Group to "Balloon Hints".

Right click the "Balloon Hints" Group and **Add DUI Form** twice. Change the name of the first form to "Show Hint" and the second to "Hide Hint".

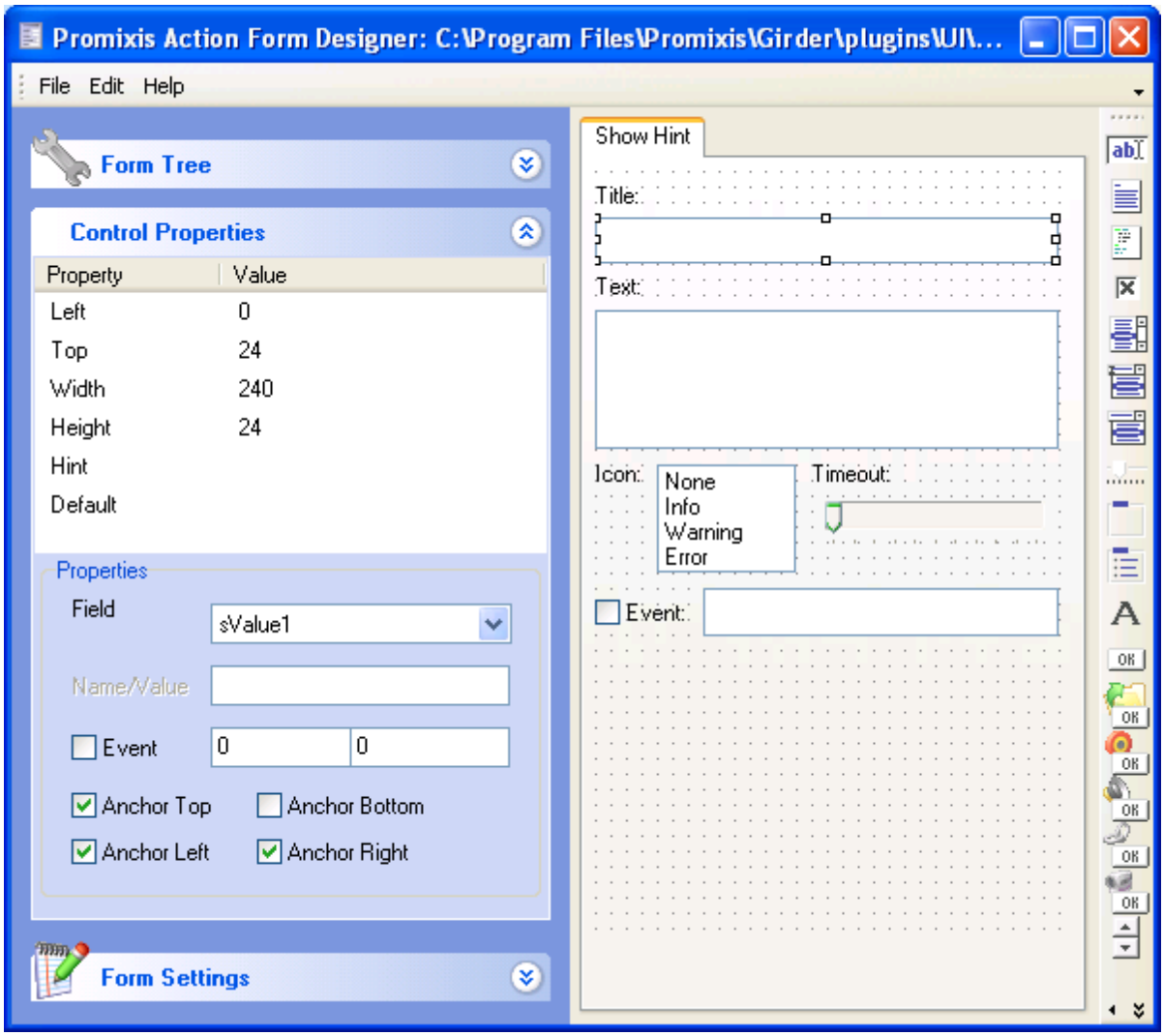
In the **Form Settings** set **Plugin Number** to 243 for both forms. Set **Action Subtype** to 8000 for the first form and 8001 for the second.



Tip: **Plugin Number** must be 243, the Tree Script Plugin. **Action Subtypes** are arbitrary, but must be unique amongst all Tree Script Actions. You can check availability of numbers using the Variable Inspector by expanding the **Actions** table within the **treescript** table.

The Hide Hint form does not need any controls, but the Show Hint form is more complex.

© 2008 Promixis, LLC



The text labels are created with the **Add a Label** button. The controls running left to right, top to bottom, using default values except as stated, are:

- Edit Box. Field = sValue1; Anchor Right ticked. (showing in the screenshot).
- Memo. Field = sValue2; Anchor Right ticked.
- List Box. Field = iValue1; Values = {None=0, Info=1, Warning=2, Error=3}, Default = 0.
- Slider. Field = iValue2; Min = 10; Max = 60; Default = 10.
- Checkbox. Field = bValue1; Caption = "Event".
- Edit Box. Field = sValue3; Anchor Right ticked. Default = "BalloonEvent".

Save the file and exit the Designer.

Creating the Script

Now create a new file called "Balloon.lua" in the **%GIRDER%\plugins\treescript** directory. Place the following code in this file.

```
treescript.Action[8000] = {}
treescript.Action[8000].OnAction = function(Action, Event)
```

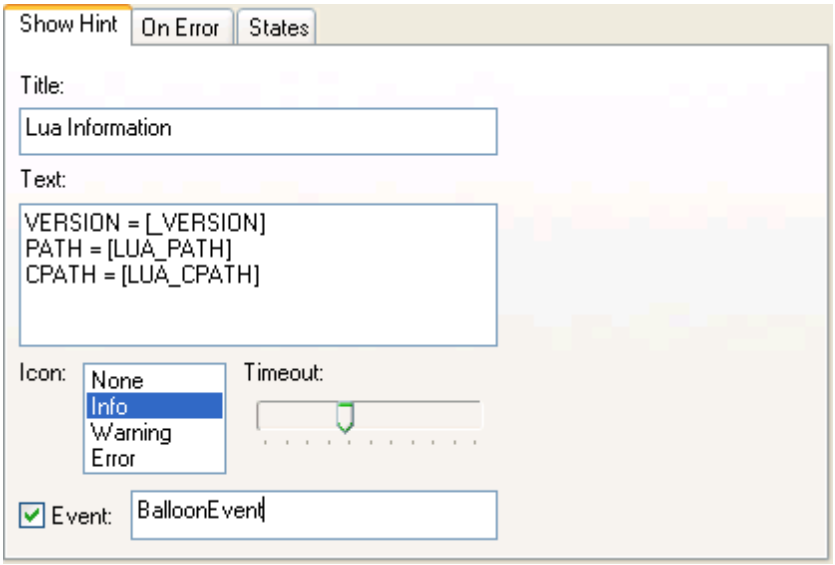
© 2008 Promixis, LLC

```
  if (Action.bValue1 ~= 0) then
    gir.ShowBalloonHint(Action.sValue1, gir.ParseString(Action.sValue2),
         Action.iValue2, Action.iValue1, Action.sValue3)
  else
    gir.ShowBalloonHint(Action.sValue1, gir.ParseString(Action.sValue2),
         Action.iValue2, Action.iValue1, nil)
  end
  return true
end
treescript.Action[8001] = {}
treescript.Action[8001].OnAction = function(Action, Event)
 gir.HideBalloonHint()
  return true
end
return "balloon.xml"
```

This script gets run automatically (because it is in a special directory) when the TreeScript Plugin is started and when a script reset occurs. The script compiles the functions and references them in a defined Global Table structure for later use. It then returns the name of the Dynamic User Interface definition file. The TreeScript Plugin reads this file and installs the Actions in the task box. Later, when the Action is triggered, TreeScript finds the **OnAction** function and executes it giving it the action parameters set by the user when the Action was last edited.

Testing the Actions

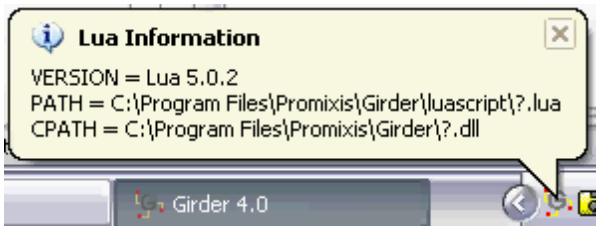
Close down and restart Girder. The Actions task box should now have a new Group **Balloon Hints** with two new Actions. To test them create a new Group in the Girder Tree and add one of each. Complete the **Show Hint** Action Editor as follows.



Notice that this is the form we created earlier. The construction in the Text field works using **gir.ParseString** in the script. This expands Lua variable names enclosed in square brackets into values.

Executing the **Show Hint** action produces a result like this.

© 2008 Promixis, LLC



You can tick the **Event** checkbox and enter an **Event String** (defaults to BalloonEvent). This event will be generated when the user clicks on the balloon.

Conclusion

This example just scratches the surface of the capabilities of the TreeScript Plugin. It can be used to create new Conditionals as well as Actions and also new pages for the Settings dialog. It is capable of producing complex interactive multi-page user interface dialogs. See the Tree Script Plugin section for full details.

7.6 Transport Tutorial

(GIRDER PRO)

This is an advanced topic that you can safely skip until you actually want to know how to support new hardware.

Transport Library Reference

7.6.1 Introduction

Girder's transport plugin / system covers Serial, TCP/IP and HID (USB/Bluetooth) communications. These are the most prevalent ways of communicating with external devices on the market. A nice bonus feature of the transport system is that it doesn't matter what the actual underlying hardware is (Serial, TCP/IP) from a programming perspective they are all the same, allowing you to write a driver once and then connect to it through any means.

For example you have a serially connected amplifier for which a driver was written. You however want to connect it through a TCP/IP to Serial converter like the GlobalCache. Well no fear as the transport plugin makes that trivial. Simply change one or two lines in the initialization code and the plugin talks over tcp/ip!

Next we'll explore the difference between Asynchronous and Synchronous operation and how that affects us.

© 2008 Promixis, LLC

7.6.2 Asynchronous Operation

Girder uses a system call "Asynchronous Communication" to handle input and output. This page explains what that is and how it helps us.

In general input output can work in two modes, synchronous and asynchronous. An example of synchronous operation is the following **pseudo code**

```
1: Obj = transport.new( TCPIP )
2:  Obj:Open("promixis.com",  4998)
3:  Obj:Write("version\r")
4: local response = Obj:ReadLine()
5:  Obj:Destroy()
```

Imagine this as a phone conversation.

Line 1: We pick up the phone
Line 2: We dial the person we want to talk too (the Promixis Global Cache Emulator in this case)
Line 3: We request the device version.
Line 4: We read all the input until end of line
Line 5: We close the connection

While all this is happening the program cannot do anything else, it is 'blocked' by the conversation and has to wait for it to finish. So this all happens synchronously within those 5 lines. This is fine for small programs, however for an application like Girder that has to be able to handle many different devices at the same time and still be responsive and handle all your automation tasks that is not acceptable. That is why Girder has a mechanism called 'Asynchronous Communications' or 'Non-Blocking'. This requires a little bit of different thinking on the programmers part but once you understand this it is very powerful.

Some **pseudo code** to illustrate what this is.

```
1.function Callback ( event, data )
2.   if ( event == NEW_CONNCTION ) then
3.      Obj:Write("version\r")
4.   end
5.   if ( event == DATA_ARRIVED ) then
6.     print(data)
7.   end
8.   if ( event == CONNECTION_CLOSED ) then
9.
10.   end
11.   if ( event == TIMEOUT ) then
12.     print('too  bad')
13.   end
14.end
15.
16.Obj = transport.new( TCPIP )
17.Obj:SetCallback(  TERMINATED,  "\r",  Callback)
18.Obj:Connect("promixis.com",   4998)
```

On first glance this looks a lot more complicated but trust me it is not really. Let step through it bit by bit. We'll start at the bottom.

Line 16, we simply create a new transport object. (e.g. pick up the phone)
Line 17 is very interesting two things happen here. First we tell the transport that the data that we are expecting is TERMINATED by an endline (\r) character secondly we tell it what function to call when it receives an event. Examples of events are 'new connection', 'closed

© 2008 Promixis, LLC

connection' or the 'end line' was detected (DATA_ARRIVED). Image you call someone up on the phone but don't wait at the phone until they answer. Instead you ask your question and go off to do other things. Then when the other party responds you get notified.

The server will respond with something like this:

```
version,0,2.0.8-12r2
```

So in time the following happens:

Line 18 is executed.

1. The script exits! (No blocking!!) Girder can go about it's other business.
2. Callback is called with the "NEW_CONNECTION" event once the connection is established.
3. Callback is called onces with "DATA_ARRIVED" event with data containing the response.
4. Callback is called with "CONNECTION_CLOSED" event once the server closes the connection. (the virtual global cache is setup to close the connection after 5 seconds of inactivity)

The advantage of this being method being that this doesn't block and it can handle hundreds of connections at the same time. The blocking method would require a new thread for each device which doesn't scale very well. On the next page you'll find a working example of this pseudo code.

Note from the field: Girder has been used to control over 200 projectors at the same time and can notify and process the responses of all 200 projectors within 5 seconds using minimal resources.

7.6.3 Low Level Example

Here is example code showing you how to talk to a Global Cache to get the version information using the low level transport functions.

We are running a virtual Global Cache on the Promixis webserver for you guys to test out. It supports most of the Global Cache command set, however not the newer stuff like comport speed settings. It also closes the connection after 5 seconds of no activity. Please do not abuse this service.

TIP 1: Open the Interactive Lua console (F7) to see errors and normal output!
TIP 2: Drag the line at the right hand bottom of the main window up and then drag and drop the Interactive Lua Console into that.
TIP 3: Read the Global Cache API here.

```
local c = {} -- this can be local as the callback holds a reference to 'c'.

function Callback( data, event )

   if ( event == transport.constants.event.CONNECTIONESTABLISHED ) then
       if ( data == 0 ) then
           print("New  Connection")
           c:Write('version\r')
       else
           print("New Connection Failed: ", data)
           c:Close()
           c = nil
           return
       end

       return
   end
```

© 2008 Promixis, LLC

```
        if ( event == transport.constants.event.RXCHAR ) then
        print(data)
        -- could call c:Close() here but we'll wait for the server to close.
    end

    if ( event == transport.constants.event.CONNECTIONCLOSED ) then
        print("Connection  Closed")
        c:Close()
        c = nil
        return
    end

end

c = transport.New(transport.constants.transport.GIP)
c:Callback(transport.constants.parser.TERMINATED, '\r', 1000, Callback)
c:Open('www.promixis.com',4998)
```

If you refer back to the Asynchronous operation page you see that this example closely follows pseudo code there. Now not much is happening in this code yet but the following chapters will expand on the ideas presented here.

7.6.4 Basic Transport Classes

Now that we have a feeling for how the transport plugins works under the hood lets have a look at some classes that make your life considerably easier when dealing with asynchronous IO.

The class that everything builds upon is called "transport.Base". It handles all the low level stuff and gives us a more human readable way of handling the I/O and queues input/output. So let's build a Global Cache driver.

Tip: Tools of the trade, to edit Lua files we like to use SciTE.

Create a new lua file called "tutorial1.lua" in the Luascript\transport\devices directory.

```
local Super = require('transport.Base')
local prepare = require('transport.PrepareSettings')
local constants = require('transport.constants')
local table = require('table')
local print = print
local math = require('math')


module("transport.devices.tutorial1")
```

IMPORTANT: The module name you give here must be the same as the filename. So if you name the file "Tutorial 1 and 2.lua" the module name must be "transport.devices.Tutorial 1 and 2"

The code above takes care of some basic Lua module packaging. This prevents us from polluting the global namespace and keeps things nice and tidy. The next piece is the configuration table. It contains the settings on how to connect to the device, serial, tcp or HID. Note that this table is built by calling the function prepare (or actually transport. PrepareSettings). Since our classes can be instantiated for Serial, TCP or HID transports we'll need to have 3 tables with settings one for each transport. Instead of creating these manually this function does the hard work. The first parameter is the base table, the 3 next parameters are the overrides that apply to serial, tcp and hid transports respectively. As you can see below we only override the Serial transport. This is typical.

© 2008 Promixis, LLC

```
local DefaultSettings = prepare(
    -- first base settigs.
    {
        Transport = {
            Post = {
                NoResponseTimeout = 2000,
            },
        },
        Parser = {
            Type = constants.parser.TERMINATED,
            Terminator = '\r',
            Timeout = 1500,
        },
    },
    -- overrides for serial transport
    {
        Transport = {
            Post = {
                Baud = 9600,
                Flow = 'N',
                Parity = 'N',
                DataBits = 8,
                StopBits = 1,
            },
        },
    }
)
```

Transport.Post.NoResponseTimeout is the time a a response can take at maximum. Transport.Parser table handles how the underlying transport should analyze the incoming data. For the Global Cache we know it is \r terminated so we specify a Type of constants. parser.TERMINATED and set the Terminator to '\r\. The timeout should be smaller than the NoResponseTimeout but about the same order of magnitude.
The second parameter to prepare holds the overrides for the serial transport. For the Global Cache this does not make much sense but it's here as an example.

Alright, now comes the fun stuff.

```
local GlobalCache = Super:Subclass ( {

    Name = 'Global Cache TT1',
    Description = 'Global Cache Transport Tutorial 1',

    GetSettings = function ( self )
        return table.copy(DefaultSettings[self.TransportType])
    end,

    OnConnectFailed = function ( self )
        print("Connect Failed")
    end,

    OnConnected = function ( self )
        self:Send("version")
    end,

    OnBuildCommand = function (self, data )
        return data ..'\r'
    end,

    OnReceiveData = function(self, Event)
```

© 2008 Promixis, LLC

```
        print(self.Name .. " Received:  " .. Event:GetData())
    end,

    OnDisconnected = function(self)
        self:Close()
    end,

} )
```

That's it! For a basic working class this is all that you need! Lets step through the lines and examine what they do.

```
Name = 'Global Cache TT1',
Description = 'Global Cache Transport Tutorial 1',
```

This gives the transport a name and description. Nothing spectacular here. The `GetSettings` function is marginally more interesting. The base or super class calls this to retrieve the transport settings.

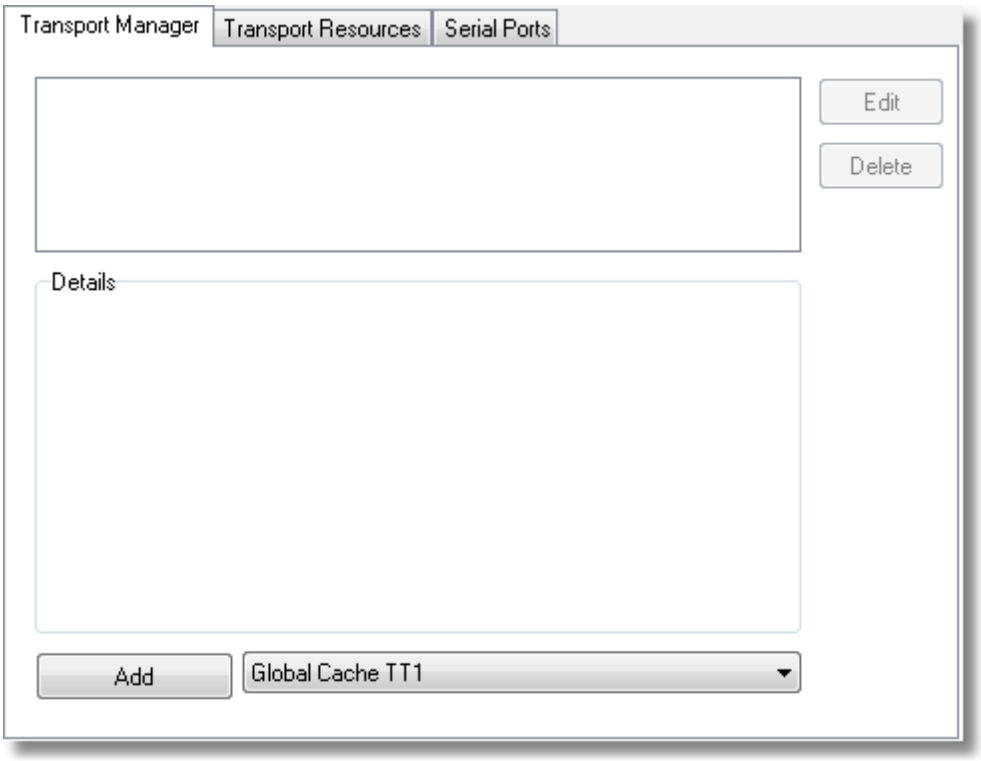
The OnBuildCommand function is called right before data is sent to the device and allows you to add a checksum or other data expected by the device. The Global Cache expects commands to be terminated by a '\r' so here we append this.

Now we have one last bit of business that needs to be taken care of. We need to specify a function that is exported out of the module, this is done by not declaring it local. So the function New below takes care of creating a new Global Cache Object, we also export Name and Description as this is used by the Transport Manager UI.

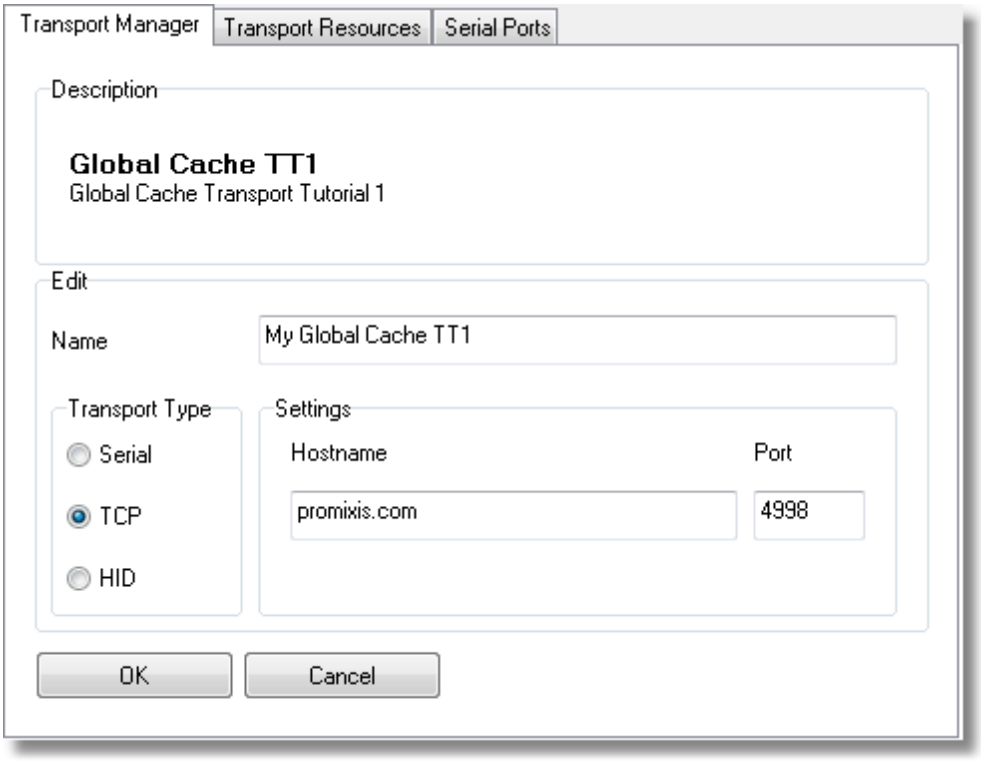
```
Name = GlobalCache.Name
Description = GlobalCache.Description
function New (self,settings)
    return GlobalCache:New(settings)
end
```

And that is really it for the whole file. Impressed? I thought so. Now to actually use this open the Component Manager and go to the Transport Manager. You should see an window like this:

© 2008 Promixis, LLC



Note how our newly created Object appears next to the Add button. If it does not appear for you do a script reset and reopen the Transport Manager.



© 2008 Promixis, LLC

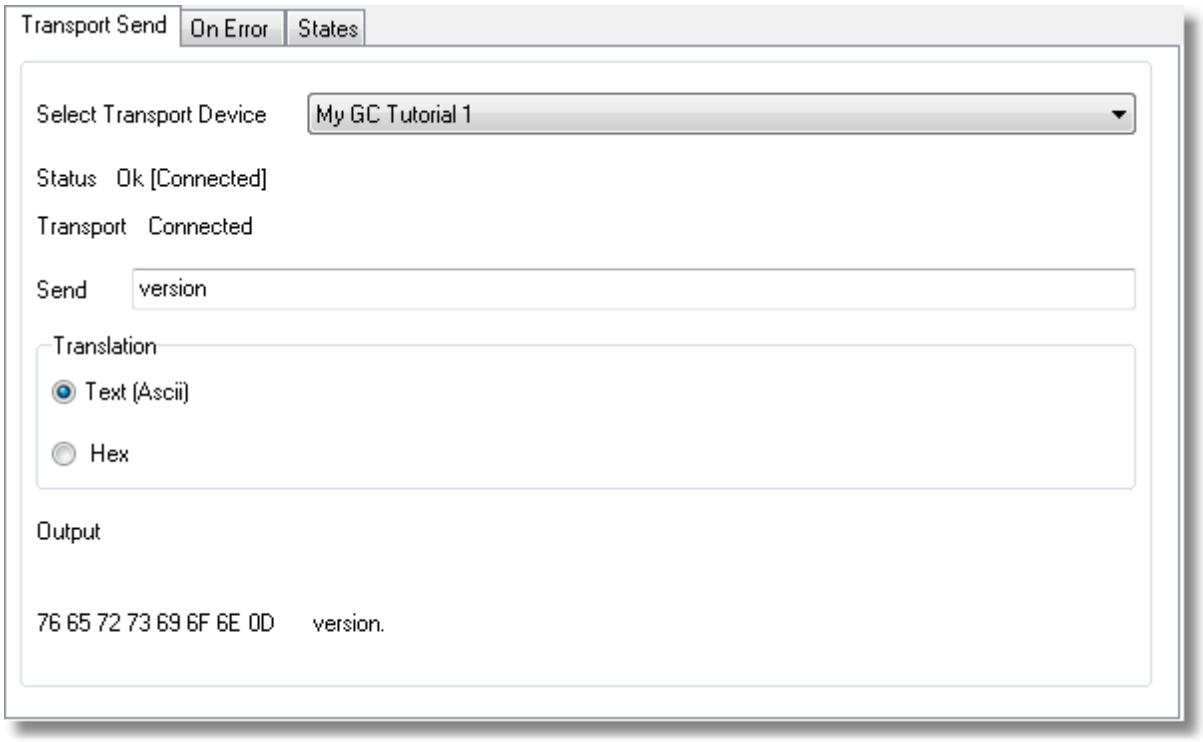
Enter "promixis.com" and 4998 (or your own Global Cache information) and you are ready to go!!

The Interactive Lua console should now show something like this:

"version,0,2.0.8-12r2"

Sending commands from Actions

You can also use an action to send data to the device. On the main Girder interface in the Action list on the left hand side, find the Transport\Transport Send action.



In this chapter we learned how to create a very basic class that allows us to talk to a remote host. Not much of this is specific to the Global Cache yet, except for the parsing. In the next chapter we are going to expand this example a bit.

For your reference here is the full tutorial1.lua

```
--[[
   Global Cache Transport Tutorial Example 1
--]]

local Super = require('transport.Base')
local prepare = require('transport.PrepareSettings')
local constants = require('transport.constants')
local table = require('table')
local print = print
local math = require('math')


module("transport.devices.tutorial1")
```

© 2008 Promixis, LLC

```
local DefaultSettings = prepare(
    -- first base settigs.
    {
        Transport = {
            Post = {
                NoResponseTimeout = 2000,
            },
        },
        Parser = {
            Type = constants.parser.TERMINATED,
            Terminator = '\r',
            Timeout = 1000,
        },
    },
    -- overrides for serial transport
    {
        Transport = {
            Post = {
                Baud = 9600,
                Flow = 'N',
                Parity = 'N',
                DataBits = 8,
                StopBits = 1,
            },
        },
    }
)

local GlobalCache = Super:Subclass ( {

    Name = 'Global Cache TT1',
    Description = 'Global Cache Transport Tutorial 1',

    Initialize = function(self )
        Super.Initialize(self)
    end,

    GetSettings = function ( self )
        return table.copy(DefaultSettings[self.TransportType])
    end,

    OnConnectFailed = function ( self )
        print("Connect Failed")
    end,

    OnConnected = function ( self )
        self:Send("version")
    end,

    OnBuildCommand = function (self, data )
        return data ..'\r'
    end,

    OnReceiveData = function(self, Event)
        print(self.Name .. " Received: " .. Event:GetData())
    end,

    OnDisconnected = function(self)
        self:Close()
    end,
```

© 2008 Promixis, LLC

```
} )


Name = GlobalCache.Name
Description = GlobalCache.Description
function New (self,settings)
    return GlobalCache:New(settings)
end
```

7.6.5 Transaction Based

Typically the communication between a computer and a device follows a request -> response, request -> response system. One 'request->response' cycle is called a 'Transaction'. We provide a class that helps deal with devices that use this system.

Note that this system allows for 'responses' without 'requests' for example a status messages.

The first two parts of this file are very similar to the Basic Transport example:

```
local Super = require('transport.TransactionBased')
local Transaction = require('transport.Transaction')
local constants = require('transport.constants')
local prepare = require('transport.PrepareSettings')
local table = require('table')
local string = require('string')
local print = print
local tonumber = tonumber

module("transport.devices.tutorial2")
```

As in the original example this sets up the module for our driver. We included three different items. transport.TransactionBased is the new superclass that handles all the complicated stuff, transport.Transaction contains the base classes for the transactions themselves.

```
local DefaultSettings = prepare(
    -- first base settigs.
    {
        Transport = {
            Post = {
                NoResponseTimeout = 2000,
            },
        },
        Parser = {
            Type = constants.parser.TERMINATED,
            Terminator = '\r',
            Timeout = 1000,
        },
        ConnectionMonitor = {
            Interval = 60000,
            RetryInterval = 10000,
        },
        MaxQueuedItems = 20,
    },
    -- overrides for serial transport
    {
        Transport = {
            Post = {
                Baud = 9600,
                Flow = 'N',
```

© 2008 Promixis, LLC

```
                Parity = 'N',
                DataBits = 8,
                StopBits = 1,
            },
        },
    }
)
```

The DefaultSettings bit is almost the same as before. No surprise there as we are still connecting to the same device. The one difference being the Connection Manager. This will try to reconnect to the Global Cache if for any reason the connection was broken.

Now comes the interesting part. The transaction based approach allows us to extract the communication details from the transport class into small classes that do nothing but deal with one specific command. Lets look at the base transaction for the Global Cache.

Base Transaction

```
local GCTransaction = Transaction:Subclass ({

    Send = {
        Prefix = '',
        Suffix = '\r',
        CommandSeparator = ',',
        ParameterSeparator = ',',
    },

    Response = {
        Params = {
            [1] = {
                Parse = function(self, parameter) return parameter end,
            },
            [2] = {
                Parse = function(self, parameter)  return parameter end,
            },
        },
    },

    IsError = function (self, data)
        if string.find(data,'^unknowncommand') then
            return true
        end
        return false
    end,

    AnalyseResponse = function(self, data, obj)
        if self:IsError(data) then
            return nil,nil, self.ResponseCodes.Error
        end
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d+),(.+)')
        return cmd, {a,b}
    end,

    SetResults = function(self, obj, a,b)
        print(obj.Name .. ' ' .. self.Send.Command, a,b)
    end,

    OnNoResponse = function(self, obj)
        print("no  response  received")
    end,
})
```

© 2008 Promixis, LLC

This is the base transaction class for the Global Cache. It has a few members that are of interest. First there is Send Group.

- Prefix is the string that is prefixed to a command. The Global Cache does not use this.
- Suffix is the end of command string, in this case that is '\r'.
- CommandSeparator is the character that sits in between the command and the parameters.
- ParameterSeparator is the character that sits in between

The next group is Response. This is where we define the structure of the response. It holds a list of arrays with Parse functions to check if the parameters are correct. Return nil if you found a parameter to not be correct.

Next we have AnalyzeResponse. This is called whenever a full set of data has arrived (determinted by the parser you specified in the DefaultSettings table). The function is expected to return the command name and an array filled with parameters. The command name has to match Response.Response this is how the system determines what object should receive the response data. You specify response.Response in the child class. An example is given a bit further below.

The "SetResults" function is called when AnalyzeReponse return the correct command and all parse functions returned data as expected. Now we need to subclass this class to make it actually specific to an command in the Global Cache. note that you can (and sometimes have to) override any of the members above to make it fit with the functionality and command you are trying to implement.

Last but not least there is OnNoResponse. This is called if the `NoResponseTimeout` specified in DefaultSettings expires.

GetVersion - Simple example

```
local GetVersion = GCTransaction:Subclass ( {

    Send = {
        Command = "version",
    },

    Response = {
        Command = 'version',
    },
})
```

Wow! That is very short! All we needed to do in this subclass is tell it what the command is to send in Send.Command and what Response to expect in Response.Command. Doesn't get easier that this.

GetDevices - Multiple Lines of data

Some command return multiple lines of data, getdevices comes to mind. This can be handled by defining a 'EndResponse' Table.

```
local GetDevices = GCTransaction:Subclass ( {

    Send = {
        Command = "getdevices",
    },

    Response = {
        Command = 'device',
    },
```

© 2008 Promixis, LLC

```
    EndResponse = {
        Command = 'endlistdevices',
    },

    SetResults = function(self, obj, a,b)
        print(obj.Name .. " found device: ", a,b)
        obj:_GetModuleVersion(tonumber(a))
    end,

})
```

This Transaction will keep processing until it receives 'endlistdevices'. The next transaction will demonstrate how to pass parameters along:

GetModuleVersion - Passing Parameters

```
local GetModuleVersion = GCTransaction:Subclass ( {

    Send = {
        Command = "getversion",
        Params = {
            [1] = {
                Type = 'number',
                Description = 'Module  Number',
                Optional = false,
            },
        },
    },

    Response = {
        Command = 'version',
    },

    SetResults = function(self, obj, a,b)
        print(obj.Name .. " module:  " ..a .. " has version: "..b)
    end,
})
```

As you can see the Send table has a new member called "Params" with contains a list of parameters that this transaction expects. In this case we expect one parameter of the type integer and it is not optional.

The Global Cache v2 class.

Last but not least we define the actual transport class. Remember 'Super' is defined as transport.TransactionBased, giving our class the tools to handle transactions.

```
local GlobalCache = Super:Subclass ( {

    Name = 'Global Cache TT2',
    Description = 'Global Cache Transport Tutorial 2',
    GUIDefaults = {
        allowedtransports = {
            [constants.transport.GIP] = true,
            [constants.transport.SERIAL] = true,
        },
        defaulttransport = constants.transport.GIP,
        hostname = "www.promixis.com",
        port = 4998,
    },
```

© 2008 Promixis, LLC

```
    GetSettings = function ( self )
        return table.copy(DefaultSettings[self.TransportType])
    end,

    OnConnected = function (self)
        self:_GetVersion()
        self:_GetDevices();
    end,

    OnDisconnected = function(self)
        self:Close()
    end,

    _GetVersion = function ( self )
        self.TransactionManager:QueueCommandClass( GetVersion )
    end,

    _GetModuleVersion = function ( self, Module )
        self.TransactionManager:QueueCommandClass( GetModuleVersion, Module )
    end,

    _GetDevices = function ( self )
        self.TransactionManager:QueueCommandClass( GetDevices )
    end,

} )
```

This stuff looks quite similar to the basic example, with a few differences.

1. There is a member field called "`GUIDefaults`". This tells the Transport Manager UI what Transport elements it should display and what the default values are.
2. The _GetVersion, _GetModuleVersion and _GetDevices member functions. They have but one line of code:

```
self.TransactionManager:QueueCommandClass( GetVersion )
```

All this does is create an instance of the Transaction object and queue it into the transport system. Easy as pie!

Again to use this new llua file go to the Transport manager to enable it.

And for your reference here is the full tutorial2.lua file

```
--[[
   Global Cache Transport Tutorial Example 1
--]]

local Super = require('transport.TransactionBased')
local Transaction = require('transport.Transaction')
local constants = require('transport.constants')
local prepare = require('transport.PrepareSettings')
local table = require('table')
local string = require('string')
local print = print
local tonumber = tonumber

module("transport.devices.tutorial2")

local DefaultSettings = prepare(
```

© 2008 Promixis, LLC

```
        -- first base settigs.
        {
            Transport = {
                Post = {
                    NoResponseTimeout = 2000,
                },
            },
            Parser = {
                Type = constants.parser.TERMINATED,
                Terminator = '\r',
                Timeout = 1000,
            },
            ConnectionMonitor = {
                Interval = 60000,
                RetryInterval = 10000,
            },
            MaxQueuedItems = 20,
        },
        -- overrides for serial transport
        {
            Transport = {
                Post = {
                    Baud = 9600,
                    Flow = 'N',
                    Parity = 'N',
                    DataBits = 8,
                    StopBits = 1,
                },
            },
        }
    )


    local GCTransaction = Transaction:Subclass ({

        Send = {
            Prefix = '',
            Suffix = '\r',
            CommandSeparator = ',',
            ParameterSeparator = ',',
        },

        Response = {
            Params = {
                [1] = {
                    Parse = function(self, parameter) return parameter end,
                },
                [2] = {
                    Parse = function(self, parameter)  return parameter end,
                },
            },
        },

        IsError = function (self, data)
            if string.find(data,'^unknowncommand') then
                return true
            end
            return false
        end,
```

© 2008 Promixis, LLC

```
    AnalyseResponse = function(self, data, obj)
        if self:IsError(data) then
            return nil,nil, self.ResponseCodes.Error
        end
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d+),(.+)')
        return cmd, {a,b}
    end,

    SetResults = function(self, obj, a,b)
        print(obj.Name .. ' ' .. self.Send.Command, a,b)
    end,

    OnNoResponse = function(self, obj)
        print("no response received")
    end,

})

local GetVersion = GCTransaction:Subclass ( {

    Send = {
        Command = "version",
    },

    Response = {
        Command = 'version',
    },
})

local GetDevices = GCTransaction:Subclass ( {

    Send = {
        Command = "getdevices",
    },

    Response = {
        Command = 'device',
    },

    EndResponse = {
        Command = 'endlistdevices',
    },

    SetResults = function(self, obj, a,b)
        print(obj.Name .. " found device: ", a,b)
        obj:_GetModuleVersion(tonumber(a))
    end,

})

local GetModuleVersion = GCTransaction:Subclass ( {

    Send = {
        Command = "getversion",
        Params = {
            [1] = {
                Type = 'number',
                Description = 'Module Number',
                Optional = false,
            },
        },
```

© 2008 Promixis, LLC

```
    },

    Response = {
        Command = 'version',
    },

    SetResults = function(self, obj, a,b)
        print(obj.Name .. " module:  " ..a .. " has version: "..b)
    end,
})

local GlobalCache = Super:Subclass ( {

    Name = 'Global Cache TT2',
    Description = 'Global Cache Transport Tutorial 2',
    GUIDefaults = {
        allowedtransports = {
            [constants.transport.GIP] = true,
            [constants.transport.SERIAL] = true,
        },
        defaulttransport = constants.transport.GIP,
        hostname = "192.168.1.70",
        port = 4998,
    },

    GetSettings = function ( self )
        return table.copy(DefaultSettings[self.TransportType])
    end,

    OnConnected = function (self)
        self:_GetVersion()
        self:_GetDevices();
    end,

    -- Every now and again we send the 'getversion' command to see if our connection
is still alive.
    -- if not OnDisconnected is called and we try to reconnect through the connection
monitor.
    OnConnectionCheck = function (self)
        if self.Transport:GetLastEvent ():GetTimeSince () > 10000 then
            self:_GetVersion ()
        end
    end,

    _GetVersion = function ( self )
        self.TransactionManager:QueueCommandClass( GetVersion )
    end,

    _GetModuleVersion = function ( self, Module )
        self.TransactionManager:QueueCommandClass( GetModuleVersion, Module )
    end,

    _GetDevices = function ( self )
        self.TransactionManager:QueueCommandClass( GetDevices )
    end,

} )


Name = GlobalCache.Name
```

© 2008 Promixis, LLC

```
Description = GlobalCache.Description
GUIDefaults = GlobalCache.GUIDefaults
function New (self, settings)
    return GlobalCache:New(settings)
end
```

7.6.6 Device Manager Integration

Now that we have a framework to talk to the Global Cache we would like to hook this into the Device Manager! There are a few things that need to be done for that.

1. We need to detect what modules are installed in your Global Cache, that way we can handle any Global Cache model without knowing anything about it before hand.
2. Once we know what modules are inside the Global Cache we need to add Devices into the Device Manager for each connector (remember an IR port, a Relay or a Sensor is considered a connector in Global Cache Lingo.)
3. Hook state changes inside the Global Cache to the Device Manager (sensor is triggered)
4. Hook state changes inside the Device Manager to the Global Cache (Someone pushes a button on the Device Manager web interface)

Let's walk through this code in the order that stuff happens.

Detecting the available modules

```
local GetDevices = GCTransaction:Subclass ( {

    Send = {
        Command = "getdevices",
    },

    Response = {
        Command = 'device',
    },

    EndResponse = {
        Command = 'endlistdevices',
    },

    SetResults = function(self, obj, a,b)
        obj:AddDevice(a,b)
    end,

})
```

The GetDevices Transaction queries the Global Cache and calls obj:AddDevice(<module>, <type>) for every module it finds. Obj in this context is the GlobalCache Object defined below. Let's have a look at what AddDevice does:

```
    AddDevice = function ( self,a ,b )

        local _,_,cnt, moduletype = string.find(b,'(%d)  (.+)$')

        if moduletype == 'IR' then
            for i=1,cnt do
                local connector = a .. ':' .. i
                self:_GetStateAndType(connector)
            end
        end

        if moduletype == 'RELAY' then
```

© 2008 Promixis, LLC

```
            for i=1,cnt do
                local tid = a .. ':' .. i -- tid = Transport ID , in our case the
connector address.
                self:AddRelay(tid)
            end
        end

    end,
```

So first we split the 'b' parameter in a one digit number and the rest of the characters. The Global Cache sends a string like "3 RELAY". Which would parse cnt = 3 and moduletype into 'RELAY'.
Next we check what module was found. If it is a Relay we simply call AddRelay which adds the device, if it is an IR module it could be an IR output or a digital Sensor. To find out we send the GetStateAndType transaction. The Global Cache will answer 'getstate' on an ir port with 'unknowncommand' and 'state,<connector>,<state>' for an sensor. Let's look at how we handle this bit

```
local GetStateAndType = GCTransaction:Subclass ( {

    Send = {
        Command = "getstate",
        Params = {
            [1] = {
                Type = 'string',
                Description = 'connector  address',
                Optional = false,
            },
        },
    },

    AnalyseResponse = function(self, data, obj)
        if self:IsError(data) then
            obj:AddIR(self.SendParams[1])
            return nil,nil, self.ResponseCodes.Error
        end
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d:%d),(.+)')
        return cmd, {a,b}
    end,

    Response = {
        Command = 'state',
    },

    SetResults = function(self, obj, a,b)
        obj:AddSensor(a,b)
    end,
})
```

As you can see in AnayzeResponse if we detect an error we know it is an IR out port and we call the GlobalCache object's AddIR member function. Otherwise if we receive a 'state, <connector>,<state>' response we know we are dealing with an IR sensor and call obj: AddSensor(). Let's have a closer look at these

Create Devices and Event handlers

```
    -- adds a relay to Girder's DM system.
    AddRelay = function ( self, connector )

        local deviceproperties = {
```

© 2008 Promixis, LLC

```
            ID = DM.makeid(self,  connector),
            Location = 'GlobalCache',
            Name = 'Relay '.. connector,
            Description = 'Global Cache Relay',
            Status = DeviceManager.Devices.Statuses.Ok,
            Provider = self:GetComponent().Provider,
        }

        local device =  (DeviceManager.Devices.Classes.Switch:New  (deviceproperties))
        local handler = {

            Device = device,

            GC2DM = function ( self, state )
                if tonumber(state) == 1 then
                    return 'On'
                else
                    return 'Off'
                end
            end,

            DM2GC = function ( self, state )
                if state == 'On' then
                    return 1
                else
                    return 0
                end
            end,

            -- DeviceCommand = initiated from Girder / Computer -> Means update
hardware.
            DeviceCommand = function ( self, handler, Command, Device, Control,
newvalue, connector)
                self:_SetState(connector, handler:DM2GC(newvalue))

            end,

            -- DeviceChange = initiated from device it self -> Means update Girder's
Device  Manager.
            DeviceChange = function ( self, handler, newvalue, connector )

                self:Event(DM.Events.DeviceState, handler.Device, 'Switch',  handler:
GC2DM(newvalue))
            end,

        }
        self.Devices[connector] = device
        self.Handlers[connector] = handler
        self:Event (DM.Events.DeviceAdd, device)
        self:_GetState(connector)   -- retrieve the state of this connector.

    end,
```

So here is the function that actually creates a Device Manager Devices, the functions that handle the interaction and exports it! Let's step through it bit by bit.

First we fill out the `deviceproperties` table. Pretty straight forward, DM.makeid creates a unique ID based upon the name of the transport instance (which is unique) and an identifier that is specific to this implementation and can really be anything. We chose to simply use the hardware connector id. Next we create the Device Manager Device object.

© 2008 Promixis, LLC

```
local device =  (DeviceManager.Devices.Classes.Switch:New  (deviceproperties))
```

A relay can be modeled nicely by a Switch thus we use that. You can see all the available Device types in the Girder Variable inspector.

The handler table is used to keep track of how to,.. handle,.. the device we just created. We store the Device object and two functions that translate between Global Cache values and Device Manager values. Lastly we have two very important functions. DeviceCommand and DeviceChange. These handle the interaction between Girder's Device Manager and the hardware.

The last bit of this function stores both the handler and the device in our object and Sends an Event to the Transport Provider indicating that we have created new Devices that it should add to the Device Manager system. The very last line queries the state for the relay.

The functions AddIR and AddSensor are very much the same as this one with some modification for the specific device types. That is all there is to creating a fully operational Global Cache Device Manager driver about 400 lines of code!

And for your reference here is the full tutorial3.lua file

```
--[[

   Global Cache Transport Tutorial Example 3

--]]

local Super = require('transport.TransactionBased')
local Transaction = require('transport.Transaction')
local constants = require('transport.constants')
local prepare = require('transport.PrepareSettings')
local DM = require('transport.DM')
local table = require('table')
local string = require('string')
local print = print
local tonumber = tonumber
local pairs = pairs
local DeviceManager = DeviceManager
local ComponentManager = ComponentManager
local FormatCCF = transport.support.tcp.FormatCCF

module("transport.devices.tutorial3")

local DefaultSettings = prepare(
    -- first base settigs.
    {
        Transport = {
            Post = {
                NoResponseTimeout = 2000,
            },
        },
        Parser = {
            Type = constants.parser.TERMINATED,
            Terminator = '\r',
            Timeout = 1000,
        },
        ConnectionMonitor = {
            Interval = 60000,
            RetryInterval = 10000,
```

© 2008 Promixis, LLC

```
        },
        MaxQueuedItems = 20,
    }
)


-- Global Cache transaction base class.
local GCTransaction = Transaction:Subclass ({

    Send = {
        Prefix = '',
        Suffix = '\r',
        CommandSeparator = ',',
        ParameterSeparator = ',',
    },

    Response = {
        Params = {
            [1] = {
                Parse = function(self, parameter) return parameter end,
            },
            [2] = {
                Parse = function(self, parameter)  return parameter end,
            },
        },
    },

    IsError = function (self, data)
        if string.find(data,'^unknowncommand') then
            return true
        end
        return false
    end,

    AnalyseResponse = function(self, data)
        if self:IsError(data) then
            return nil,nil, self.ResponseCodes.Error
        end
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d+),(.+)')
        return cmd, {a,b}
    end,

    SetResults = function(self, obj, a,b)
        --  print(self.Send.Command,  a,b)
    end,

    OnNoResponse = function(self, obj)
        self:Log(3, "Global Cache no response received")
    end,
})

-- Queries the version of the GC
local GetVersion = GCTransaction:Subclass ( {

    Send = {
        Command = "version",
    },

    Response = {
        Command = 'version',
```

© 2008 Promixis, LLC

```
    },
})

-- lists all devices in the GC
local GetDevices = GCTransaction:Subclass ( {

    Send = {
        Command = "getdevices",
    },

    Response = {
        Command = 'device',
    },

    EndResponse = {
        Command = 'endlistdevices',
    },

    SetResults = function(self, obj, a,b)
        obj:AddDevice(a,b)
    end,

})

-- this command is used to determine if the device at
-- connector address is a sensor or an ir-output port.
-- it will call AddIR or AddSensor correspondly.
local GetStateAndType = GCTransaction:Subclass ( {

    Send = {
        Command = "getstate",
        Params = {
            [1] = {
                Type = 'string',
                Description = 'connector  address',
                Optional = false,
            },
        },
    },

    AnalyseResponse = function(self, data, obj)
        if self:IsError(data) then
            obj:AddIR(self.SendParams[1])
            return nil,nil, self.ResponseCodes.Error
        end
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d:%d),(.+)')
        return cmd, {a,b}
    end,

    Response = {
        Command = 'state',
    },

    SetResults = function(self, obj, a,b)
        obj:AddSensor(a,b)
    end,
})

-- gets the state of a relay or sensor.
local GetState = GCTransaction:Subclass ( {
```

© 2008 Promixis, LLC

```
    Send = {
        Command = "getstate",
        Params = {
            [1] = {
                Type = 'string',
                Description = 'connector  address',
                Optional = false,
            },
        },
    },

    AnalyseResponse = function(self, data, obj)
        if self:IsError(data) then
            return nil,nil, self.ResponseCodes.Error
        end
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d:%d),(.+)')
        return cmd, {a,b}
    end,

    Response = {
        Command = 'state',
    },

    SetResults = function(self, obj, a,b)
        obj:UpdateState(a,b)
    end,
})

-- sets the state of the relays
local SetState = GCTransaction:Subclass ( {

    Send = {
        Command = "setstate",
        Params = {
            [1] = {
                Type = 'string',
                Description = 'connector  address',
                Optional = false,
            },
            [2] = {
                Type = 'number',
                Description = 'state',
                Optional = false,
            },
        },
    },

    AnalyseResponse = function(self, data, obj)
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d:%d),(%d)')
        return cmd, {a,b}
    end,

    Response = {
        Command = 'state',
    },

    SetResults = function(self, obj, a,b)
        obj:UpdateState(a,b)
    end,
```

© 2008 Promixis, LLC

```
})

-- This is a neat little transaction, it is a 'listen only' one. It gets queued and
sits on the queue
-- only responding to 'statechange' events from the GC.
local StateChange = GCTransaction:Subclass ( {

    ResponseOnly = true,
    Persist = true,

    AnalyseResponse = function(self, data, obj)
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d:%d),(%d)')
        return cmd, {a,b}
    end,

    Response = {
        Command = 'statechange',
    },

    SetResults = function(self, obj, a,b)
        obj:UpdateState(a,b)
    end,
})


-- Handles sending the IR codes, expects the ir code in GC format.
local SendIR = GCTransaction:Subclass ( {

    Send = {
        Command = "sendir",
        Params = {
            {
                Type = 'string',
                Description = 'ircode',
                Optional = false,
            },
        },
    },

    AnalyseResponse = function(self, data, obj)
        if self:IsError(data) then
            obj:UpdateState(a,'Error')
            return nil,nil, self.ResponseCodes.Error
        end
        local _, _, cmd, a,b = string.find(data, '([^,]+),(%d:%d),(.+)$')
        return cmd, {a,b}
    end,

    Response = {
        Command = 'completeir',
    },

    SetResults = function(self, obj, a,b)
        obj:UpdateState(a,'Ok')
    end,
})


--[[
     The Transport.TransactionBased Class.
```

© 2008 Promixis, LLC

```
--]]

local GlobalCache = Super:Subclass ( {

    Name = 'Global Cache TT3',
    Description = 'Global Cache Transport Tutorial 3',
    GUIDefaults = {
        allowedtransports = {
            [constants.transport.GIP] = true,
        },
        defaulttransport = constants.transport.GIP,
        hostname = "192.168.1.70",
        port = 4998,
    },

    Initialize = function ( self )
        Super.Initialize(self)
        self.Devices = {}
        self.Handlers = {}
    end,

    GetSettings = function ( self )
        return table.copy(DefaultSettings[self.TransportType])
    end,

    -- Alright! We are connected to something, let's try to get the device list
    OnConnected = function (self)
        self:_GetDevices()
        self:_StateChangeListener()
    end,

    -- OK the GC has gone away, remove all the devices.
    OnDisconnected = function(self)
        self:_ClearDevices()
    end,

    -- Every now and again we send the 'getversion' command to see if our connection
is still alive.
    -- if not OnDisconnected is called and we try to reconnect through the connection
monitor.
    OnConnectionCheck = function (self)
        if self.Transport:GetLastEvent ():GetTimeSince () > 10000 then
            self:_GetVersion ()
        end
    end,

    --[[
         The actual function handling the transactions
    --]]
    _SendIR = function ( self, Connector, CCF, Repeat )
        local _,_,mod, con = string.find(Connector, "(%d):(%d)")
        if not mod or not con then
            return false
        end

        local s = FormatCCF( CCF, mod, con, Repeat )
        if not s then
            return false
        end
        s = string.sub(s, 8)
```

© 2008 Promixis, LLC

```
        self.TransactionManager:QueueCommandClass( SendIR, s )
        return true
    end,

    _GetStateAndType = function ( self, Connector )
        self.TransactionManager:QueueCommandClass( GetStateAndType, Connector )
    end,

    _GetState = function (self, Connector )
        self.TransactionManager:QueueCommandClass( GetState, Connector )
    end,

    _StateChangeListener = function ( self )
        self.TransactionManager:QueueCommandClass( StateChange )
    end,

    _GetVersion = function ( self )
        self.TransactionManager:QueueCommandClass( GetVersion )
    end,

    _GetModuleVersion = function ( self, Module )
        self.TransactionManager:QueueCommandClass( GetModuleVersion, Module )
    end,

    _GetDevices = function ( self )
        self:_ClearDevices()
        self.TransactionManager:QueueCommandClass( GetDevices )
    end,


    _SetState = function ( self, connector, state )
        self.TransactionManager:QueueCommandClass( SetState, connector, state )
    end,


    --[[  Device  Manager  Functionality

        This part is tailored to the Global Cache functionality but generic enough to
be
        of use for future implementations.

        We keep 2 tables:

        self.Devices [ connector ] = DM Device
        self.Handlers[ connector ] = {
              DeviceCommand = function ( self, handler, Command, Device, Control,
newvalue,  connector)   end
             DeviceChange = function ( self, handler, newvalue, connector )  end
       }

        self.Devices simply keeps track of all the devices that we have exported to the
DM
        self.Handlers is a table that is used to keep track of functions that operate
on the device
        or respond to event from the device.

        DeviceCommand is called when Girder generates a state change for one of the
Device's  controls.
```

© 2008 Promixis, LLC

```
        DeviceChange is called when the hardware generates a state change and this
needs to be progated into the Girder DM system

    --]]

    -- Removes the devices if the GC goes missing.
    _ClearDevices = function ( self )
        for connector, device in pairs(self.Devices) do
            self:Event (DM.Events.DeviceDelete, device)
        end
        self.Devices = {}
        self.Handlers = {}
    end,

    -- GetDevices is called during startup and shutdown of this instance
    -- to add or remove any devices we might have at that point.
    -- This plugin does not have any fixed devices, it queries for them
    -- at start up.
    GetDevices = function ( self )
        local dev = {}
        for tid, device in pairs(self.Devices) do
            dev[ device:GetID() ] = device
        end
        return dev
    end,

    -- This is called by the Transport Manager Provider if from Girder the state of
    -- a control changes, we need to figure out what control is being updated and
    -- update the hardware to reflect this change. Note that once the hardware is
update
    -- we have to send an event back into the DM system changing the state of the
control.
    -- think of this as a 'request to change state' and the state will be changed by
us.
    DeviceCommand = function(self, Command, Device, Control, newvalue, connector)

        local handler = self.Handlers[connector]
        if not handler then
            return
        end
        if handler.DeviceCommand then
            handler.DeviceCommand(self, handler, Command, Device, Control, newvalue,
connector)
        end
    end,


    -- This function is called when the state of a Sensor or Relay changes
    -- this event is then handed off to the handler table that belongs to that device.
    UpdateState = function ( self, connector, state )
        local handler = self.Handlers[connector]
        if not handler then
            return
        end
        if handler.DeviceChange then
            handler.DeviceChange(self, handler, state, connector)
        end
    end,
```

© 2008 Promixis, LLC

```
    -- Adds an IR output device to Girder.
    AddIR = function ( self, connector )
        --  Sensor
        local deviceproperties = {
            ID = DM.makeid(self,  connector),
            Location = 'GlobalCache',
            Name = 'IR '.. connector,
            Description = 'Global Cache IR output',
            Status = DeviceManager.Devices.Statuses.Ok,
            Provider = self:GetComponent().Provider,
        }

        local device =  (DeviceManager.Devices.Classes.InfraRedTransmitterCCF:New
(deviceproperties))
        local handler = {

            Device = device,

            -- DeviceCommand = initiated from Girder / Computer -> Means update
hardware.
            DeviceCommand = function ( self, handler, Command, Device, Control,
newvalue, connector)
                if Control:GetID() ~= 'SendCCF' then
                    return
                end
                local repeatcount =  (Device:GetControl ('Repeat'):GetValue ())

                if not self:_SendIR(connector, newvalue, repeatcount) then
                    self:Event(DM.Events.DeviceState, handler.Device, 'SendCCF',
'Error')
                end
            end,
            -- DeviceChange = initiated from device it self -> Means update Girder's
Device  Manager.
            DeviceChange = function ( self, handler, newvalue, connector )
                self:Event(DM.Events.DeviceState, handler.Device, 'SendCCF', newvalue)

            end,

        }
        self.Devices[connector] = device
        self.Handlers[connector] = handler
        self:Event (DM.Events.DeviceAdd, device)
        self:Event(DM.Events.DeviceState, handler.Device, 'SendCCF', 'Ok')
    end,

    -- Adds an digital sensor to Girder
    AddSensor = function ( self, connector, state )

        local deviceproperties = {
            ID = DM.makeid(self,  connector),
            Location = 'GlobalCache',
            Name = 'Sensor  '.. connector,
            Description = 'Global Cache Sensor',
            Status = DeviceManager.Devices.Statuses.Ok,
            Provider = self:GetComponent().Provider,
        }

        local device =  (DeviceManager.Devices.Classes.Sensor:New  (deviceproperties))
```

© 2008 Promixis, LLC

```
        local handler = {

            Device = device,

            GC2DM = function ( self, state )
                if tonumber(state) == 1 then
                    return "Ready"
                else
                    return "Not  Ready"
                end
            end,

            -- DeviceCommand = initiated from Girder / Computer -> Means update
hardware.
            DeviceCommand = function ( self, handler, Command, Device, Control,
newvalue, connector)
            end,
            -- DeviceChange = initiated from device it self -> Means update Girder's
Device  Manager.
            DeviceChange = function ( self, handler, newvalue, connector )
                self:Event(DM.Events.DeviceState, handler.Device, 'Condition',
handler:GC2DM(newvalue))
            end,

        }
        self.Devices[connector] = device
        self.Handlers[connector] = handler
        self:Event (DM.Events.DeviceAdd, device)
        self:Event (DM.Events.DeviceState, device, 'Condition', handler:GC2DM(state))
    end,

    -- adds a relay to Girder's DM system.
    AddRelay = function ( self, connector )

        local deviceproperties = {
            ID = DM.makeid(self,  connector),
            Location = 'GlobalCache',
            Name = 'Relay  '.. connector,
            Description = 'Global  Cache  Relay',
            Status = DeviceManager.Devices.Statuses.Ok,
            Provider = self:GetComponent().Provider,
        }

        local device =  (DeviceManager.Devices.Classes.Switch:New  (deviceproperties))
        local handler = {

            Device = device,

            GC2DM = function ( self, state )
                if tonumber(state) == 1 then
                    return 'On'
                else
                    return 'Off'
                end
            end,

            DM2GC = function ( self, state )
                if state == 'On' then
                    return 1
                else
```

© 2008 Promixis, LLC

```
                    return 0
                end
            end,

            -- DeviceCommand = initiated from Girder / Computer -> Means update
hardware.
            DeviceCommand = function ( self, handler, Command, Device, Control,
newvalue, connector)
                self:_SetState(connector, handler:DM2GC(newvalue))

            end,

            -- DeviceChange = initiated from device it self -> Means update Girder's
Device  Manager.
            DeviceChange = function ( self, handler, newvalue, connector )

                self:Event(DM.Events.DeviceState, handler.Device, 'Switch',  handler:
GC2DM(newvalue))
            end,

        }
        self.Devices[connector] = device
        self.Handlers[connector] = handler
        self:Event (DM.Events.DeviceAdd, device)
        self:_GetState(connector)   -- retrieve the state of this connector.

    end,

    -- This function is called during initialization when all the devices are listed
by  the  'getdevices'
    -- / _GetDevices  command.
    -- It determuines the type of device and acts appropriately.
    --
    -- if it is an IR devices, we still do not know if it is a sensor or IR device
    -- to tell them appart we need to do a getstate on them. If it is an IR device the
GC  will  respond  with  "unknowncommand"
    -- otherwise it repsonds with 'state,3:1,0' So depending on the response we either
call  'AddIR'  or  'AddSensor'
    --
    -- If it is an Relay module we can simply add 3 (cnt) relay devices to the system.

    AddDevice = function ( self,a ,b )

        local _,_,cnt, moduletype = string.find(b,'(%d)  (.+)$')

        if moduletype == 'IR' then
            for i=1,cnt do
                local connector = a .. ':' .. i
                self:_GetStateAndType(connector)
            end
        end

        if moduletype == 'RELAY' then
            for i=1,cnt do
                local connector = a .. ':' .. i -- tid = Transport ID , in our case
the connector address.
                self:AddRelay(connector)
            end
        end
```

© 2008 Promixis, LLC

```
    end,

} )


Name = GlobalCache.Name
Description = GlobalCache.Description
GUIDefaults = GlobalCache.GUIDefaults

function New (self, settings)
    return GlobalCache:New(settings)
end
```

7.6.7 Simple example

Here is a simple example on how to talk to either a serial device connected to your computer, or a serial device connected through a Global Cache or a TCP/IP connected device. It's all the same! Create a file called 'tutorial4.lua' in the 'luascript/transport/devices' directory. Then copy and paste the whole lot below into that. Now that is it! Simply open the Transport Manager, create an instance of this file connected to your favorite device. To send some data to the device use the 'Transport/Transport Send' Action from the Main Girder window. If the connected succeeded it will list this device in the dropdown box. This file will work for quite a few device and with minor changes for a lot of devices, don't be intimidated if you don't understand what is going on, just play with the values and tweak things. This is the fun part!

```
--[[
   Simple Tutorial 4
--]]

local Super = require('transport.Base')
local prepare = require('transport.PrepareSettings')
local constants = require('transport.constants')
local table = require('table')
local print = print
local math = require('math')

module("transport.devices.tutorial4")

local DefaultSettings = prepare(
    {
        Transport = {
            Post = {
                NoResponseTimeout = 2000,
            },
        },
        Parser = {
            Type = constants.parser.TERMINATED,
            Terminator = '\r',
            Timeout = 1000,
        },
    },
    {
        Transport = {
            Post = {
                Baud = 9600,
                Flow = 'N',
                Parity = 'N',
                DataBits = 8,
                StopBits = 1,
```

© 2008 Promixis, LLC

```
            },
        },
    }
)

local Object = Super:Subclass ( {

    Name = 'Simple Example Tutorial 4',
    Description = [[<b>Simple Example, tutorial 4</b><br>
<color=#444444>You can send commands to this device by using the transport.Transport
Send action! Ask questions about this on the <a href=www.promixis.com/forums>forum</
a></color>]],
    GUIDefaults = {
        allowedtransports = {
            [constants.transport.GIP] = true,
            [constants.transport.SERIAL] = true,
        },
        defaulttransport = constants.transport.GIP,
        hostname = "192.168.1.70",
        port = 4998,
    },

    GetSettings = function ( self )
        return table.copy(DefaultSettings[self.TransportType])
    end,

    OnConnectFailed = function ( self )
        print(self.Name .. ": Connect Failed")
    end,

    OnBuildCommand = function (self, data )
        return data ..'\r'
    end,

    OnReceiveData = function(self, Event)
        print(self.Name .. " Received: " .. Event:GetData())
    end,

} )


Name = Object.Name
Description = Object.Description
GUIDefaults = Object.GUIDefaults
function New (self,settings)
    return Object:New(settings)
end
```

© 2008 Promixis, LLC

Part

8 Events Reference

This section describes the Events that may be generated by Girder and its Plugins. Event Nodes in the Girder Tree may be trained to recognize these Events.

8.1 Predefined Events

Predefined events are a set of standard events representing all possible Mapped Events. Program Definition GML files which are written to be shared should recognize Events from this set. Standard controllers will generate these events without configuration.

Transport events

PLAY, PAUSE, PLAY/PAUSE, RECORD, STOP, NEXT, PREVIOUS, FAST FORWARD, FAST REVERSE, EJECT.

Playlist events

REPEAT, SHUFFLE.

Audio events

MASTER VOLUME UP, MASTER VOLUME DOWN, MASTER MUTE, WAVE VOLUME UP, WAVE VOLUME DOWN, WAVE MUTE, APPLICATION VOLUME UP, APPLICATION VOLUME DOWN, APPLICATION MUTE.

Channel events

CHANNEL UP, CHANNEL DOWN.

Menu events

OK, ENTER, ARROW UP, ARROW DOWN, ARROW LEFT, ARROW RIGHT, MENU.

Number pad events

KEY 1, KEY 2, KEY 3, KEY 4, KEY 5, KEY 6, KEY 7, KEY 8, KEY 9, KEY 0, +10, +100.

Mouse events

MOUSE UP, MOUSE DOWN, MOUSE LEFT, MOUSE RIGHT, MOUSE UPRIGHT, MOUSE DOWNRIGHT, MOUSE DOWNLEFT, MOUSE UPLEFT, MOUSE MODE TOGGLE, MOUSEBUTTON LEFT, MOUSEBUTTON MIDDLE, MOUSEBUTTON RIGHT, MOUSEBUTTON LEFT HOLD (toggle drag hold), MOUSEBUTTON LEFT DOUBLE CLICK.

Computer control events

SHUTDOWN, SUSPEND, HIBERNATE, RESTART, MONITOR TOGGLE, MONITOR OFF, MONITOR ON, TASKMANAGER NEXT, TASKMANAGER PREVIOUS, TASKMANAGER SELECT.

© 2008 Promixis, LLC

Window control events

MINIMIZE, MAXIMIZE, RESTORE, CLOSE.

Application control events

POWER, EXIT, WEATHER CURRENT CONDITIONS, WEATHER FORECAST, WEATHER SATELLITE, WINAMP, WINDOWS MEDIA PLAYER, WINDOWS MEDIA PLAYER CLASSIC, POWERDVD, ZOOM PLAYER, WINDVD, WINDVD 6, THEATRETEK 2, ITUNES, J RIVER MEDIA CENTER, WWW, CD, DVD, TV, TUNER, VCR, MD, VIDEO, SATELLITE, CABLE, AUX.

DVD player events

TIME, MARK, INFO, SUBTITLE MENU, ROOT MENU, AUDIO MENU, TITLE MENU, SUBTITLE CYCLE, ANGLE CYCLE, AUDIO CYCLE.

Alphabetic keyboard events

KEY A, KEY B, KEY C, KEY D, KEY E, KEY F, KEY G, KEY H, KEY I, KEY J, KEY K, KEY L, KEY M, KEY N, KEY O, KEY P, KEY Q, KEY R, KEY S, KEY T, KEY U, KEY V, KEY W, KEY X, KEY Y, KEY Z, KEY SHIFT, KEY CONTROL, KEY SPACE, TILDE, TAB.

Control key events

INSERT, HOME, PAGE UP, PAGE DOWN, END, DELETE.

Function key events

KEY F1, KEY F2, KEY F3, KEY F4, KEY F5, KEY F6, KEY F7, KEY F8, KEY F9, KEY F10, KEY F11, KEY F12.

8.2 Wildcard Events

The events listed for the Event Device "On *** Event" are wildcards which recognize multiple actual event strings.

On Any Event: Recognizes any event from any device.

On*DeviceName*Event: Recognizes any event generated by a specific device (Plugin).

8.3 Girder Events

These events are generated by Girder itself and are always available.

FileLoaded: Generated when a GML file is loaded into the Girder Tree.

FileClose: Generated when a GML file is closed and removed from the Girder Tree.

GirderOpen: Generated when Girder starts.

GirderClose: Generated when Girder closes down.

GirderEnable: Generated when plugins are enabled in the Girder UI.

GirderDisable: Generated when plugins are disabled in the Girder UI.

ScriptEnable: Generated when the scripting engine is enabled (for example, after a Scripting Engine Reset).

© 2008 Promixis, LLC

ScriptDisable: Generated when the scripting engine is disabled (for example, before a Scripting Engine Reset).

OnBatteryLow: Generated when the battery low alarm occurs on a laptop.

OnPowerStatusChanged: Generated when the battery status on a laptop computer changes. Provides detailed information in a payload which can be used in Lua scripting (see Lua Language Reference).

Lua Variable	Value	Meaning
pld1	0	AC Offline (running on battery)
pld1	1	AC Online (running on mains)
pld1	255	Unknown AC line status
pld2	0..	Number of seconds battery life remaining
pld2	-1	Battery life remaining is unknown
pld3	0..100	Percentage of battery life remaining
pld3	-1	Battery percentage is unknown
pld4	1	Battery charge status is high
pld4	2	Battery charge status is low
pld4	4	Battery charge status is critically low
pld4	8	Battery is charging
pld4	128	Battery is absent
pld4	255	Battery status is unknown

OnQuerySuspend: Generated when the computer is preparing to go into Suspend (Hibernate) power save mode.

OnSuspendFailed: Generated after OnQuerySuspend if the computer fails to go into Suspend (Standby or Hibernate) power save mode.

OnResumeAutomatic: Generated when the computer resumes normal operation from Suspend (Standby or Hibernate) power save mode to handle an event (such as an incomming call on the modem, for example).

OnSuspend: Generated after OnQuerySuspend last thing before the computer goes into Suspend (Standby or Hibernate) power save mode.

OnResumeCritical: Generated when the computer resumes normal operation after a critical suspension (usually power failure). The computer may have suspended without prior notification, so some clean-up may be required.

OnResumeSuspend: Generated when the computer resumes normal operation from Suspend (Standby or Hibernate) power save mode due to user demand.

On*Standby*: These events mirror On*Suspend* events above. Some systems may generate these for Standby as opposed to Hibernate. Tested systems do not distinguish between the two power save modes and generate On*Suspend* in either case.

OnDisplayChange: Generated whenever the resolution, color depth or refresh frequency of an attached monitor is changed.

OnACPower: Generated when a laptop computer changes from using battery power to using

© 2008 Promixis, LLC

AC power.

OnBattery: Generated when a laptop computer looses AC power and begins using its battery.

8.4 Mapping Devices

Event Mapping translates Raw Events from devices into Predefined Events used in the Girder Tree. For example, the Keyboard event string "2600000E" is translated to the Keyboard Mapping event string "ARROW UP" by default.

Girder is shipped with a standard Mapping Device for some controllers including the Keyboard which is enabled with the associated Plugin.

Additional Mapping Devices may be created using the Event Mapping Editor or the Add Remote Wizard. Mapping Devices appear as additional Event Devices in the Event Properties editor.

Generally, it is best to use the Predefined Events device which will recognize the event regardless of the Mapping Device that generated it. However it may sometimes be useful to distinguish events according to the Mapping Device.

8.5 Raw Events

These events are generated by Plugins. They can be recognized directly without setting up a mapping. Some plugins provide a list of the events that they can generate which is made available in the **Event String** drop-down of the Event Editor. Others provide an **Event Dialog** which can be shown by pressing the button beside the **Event String** setting. Still others rely entirely on the learning system for event selection.

Raw event generation is documented in the Plugins Reference.

8.6 Lua Events

It is possible to use Lua Scripting to create Event Devices and register Event Strings for them. These devices are prefixed **Lua:** in the Event Editor Event Device drop-down. This facility is used by Girder as shipped to provide lists of the events generated by many of the Pro features including the Scheduler, the X10 Home Automation system and Girder to Girder Networking.

Instructions for creating new Lua Events are given in the Scripting Events section of **Programming The Girder UI** in the Reference Manual.

8.7 Command Line and COM Event Generation

Generating Events from the Command Line

The Girder distribution includes two small stand-alone applications **event.exe** and **csevent.exe** which generate Girder Events. These can be run from the Windows command line, from batch files or from Windows shortcuts.

Event.exe requires at least one parameter, the **event string**. This can optionally be followed by the **device number** and up to three **payload strings**. If the device number is not specified, it defaults to 18 which shows as "Girder" in the event log. There is also an optional non-positional parameter **-verbose** which causes an informational window to be shown in addition to sending the Event. This may be useful for diagnostic purposes.

Example command lines.

```
"C:\Program Files\Promixis\Girder\event.exe" MyEvent
"C:\Program Files\Promixis\Girder\event.exe" MyEvent 1000 "Parameter One"
"C:\Program Files\Promixis\Girder\event.exe" MyEvent 18 -verbose
"C:\Program Files\Promixis\Girder\event.exe" MyEvent 18 one two three -verbose
```

© 2008 Promixis, LLC

Csevent.exe enables events to be generated from another machine over the network. It requires one of the pro versions of Girder and the Communications Server Plugin must be enabled. It requires the following parameters in order. **Name** or **IP Address** of Girder machine; **Port** number of server (20005 by default); **Password** of server ("girder" by default); the **event string**; the **device number**; an optional **payload** string.

Example command lines.

```
"C:\Program Files\Promixis\Girder\event.exe" cruncher 20005 girder hello 18
"C:\Program Files\Promixis\Girder\event.exe" 192.168.1.1 20005 girder hi 18 load
```

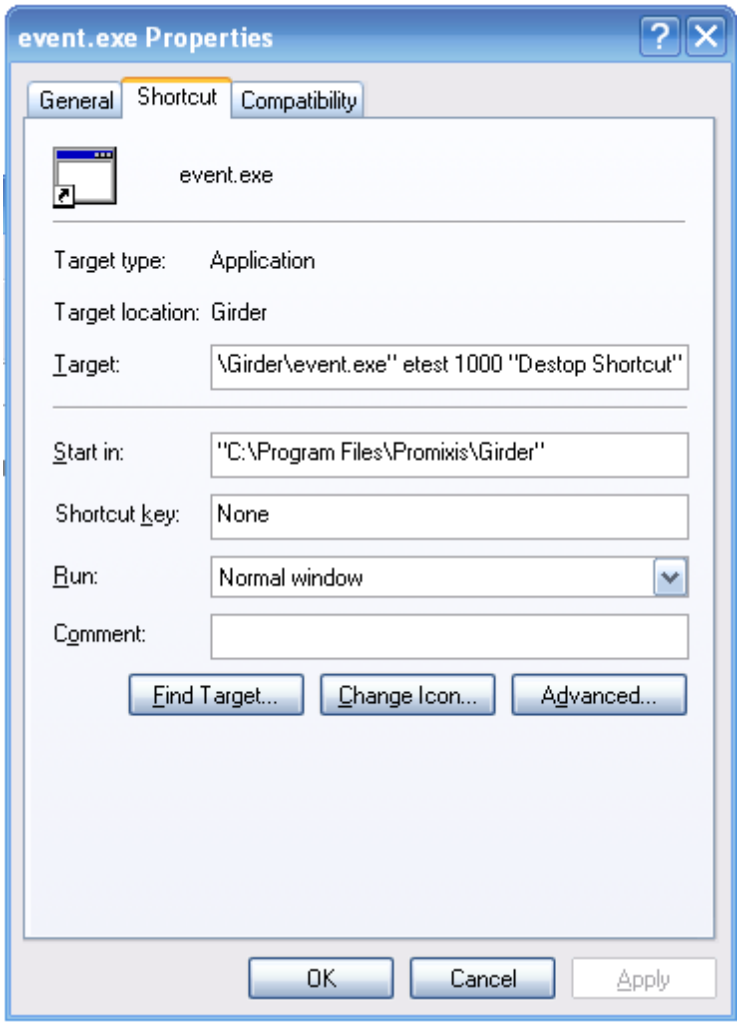
Generating Events from Shortcut Icons

Creating desktop or folder shortcuts is a quick and easy way to make simple on-screen control panels or start menu items for Girder.

To create an Event generating shortcut, right-click the desktop or a folder and take **new** on the menu. Browse to "event.exe" oe "csevent.exe" in the Girder installation directory. Add the necessary parameters after and outside the quote marks round the full path. If a parameter includes embedded spaces (in one of the payload strings for example) enclose the parameter in quote marks. Finally, give the shortcut an appropriate name.

Right-click on the created shortcut and take **Properties** on the menu which will show a dialog similar to the following.

© 2008 Promixis, LLC



Using this dialog, the parameters can be easily changed, text for the tooltip can be added ("Comment") and the icon can be changed.

Generating Events from Other Programs Using COM

Girder publishes an ActiveX COM object called **GIRDERX.Girder**. This object has a single method **TriggerEvent**.
`TriggerEvent([eventstring], [eventdevice], [pld1], [pld2], [pld3], [eventmod])`
eventstring: The Girder event string.

eventdevice: The Girder event device number.

pld1..pld3: Up to three payload strings.

eventmod: Event modifier - 0 = None, 1 = Down, 2 = Up, 4 = Repeat.

This technique can be used from almost any programming or scripting system on the Windows platform, but the actual commands will depend on what is being used. For example, using J-Script in a web page, the following function can be used to generate Girder events.

```
function triggerEvent (EventString, Device, Pld1, Pld2, Pld3)
{
  var GirderEvent = new ActiveXObject("GIRDERX.Girder");
```

© 2008 Promixis, LLC

```
  GirderEvent.TriggerEvent(EventString, Device, Pld1, Pld2, Pld3, 0)
}
```

© 2008 Promixis, LLC

Part

9 Actions Reference

This section describes all of the Actions built in to Girder and all of the Actions available in the Plugins shipped with Girder.

The source of each Action is identified in the page footer. An Action whose source is a Plugin will only be available if the relevant Plugin is enabled.

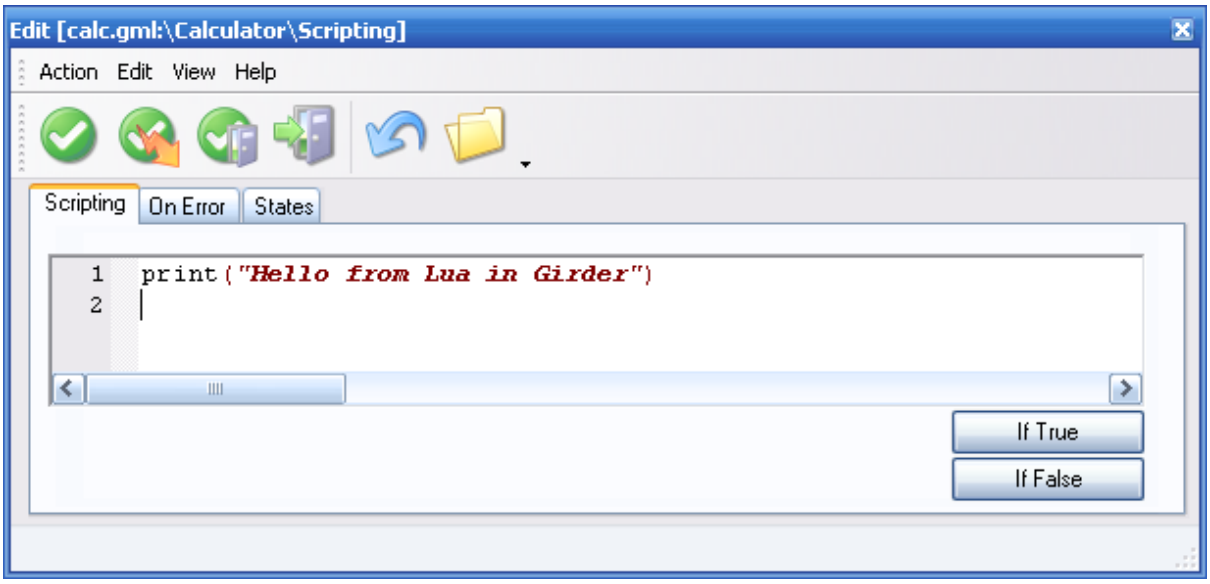
- Scripting Action for Lua scripting.
- Windows Actions control application windows.
- Flow Control Actions for writing conditional macros.
- Keyboard Actions for keyboard spoofing.
- Mouse Actions for mouse spoofing.
- Volume Actions to make audio adjustments.
- Girder Actions Group to control Girder itself.
- Monitor Actions to control the display monitor.
- OS Actions for Operating System functions.
- Command Capture Actions for message spoofing.
- Miscellaneous Actions.
- Girder 3 Legacy Actions for compatibility with old GML files.
- Scheduler Actions (Girder Pro) to generate events at specific times.
- Ambient Light Level Actions (Girder Pro).
- Girder to Girder Actions (Girder Pro) to network with other Girder machines.
- SlinkE Actions to send IR commands through a Slink-e controller.
- X10 Controls Actions to send X10 commands using CM1x device.
- IRTrans Actions to send IR commands via IRTrans device.
- PowerlincUSB Actions to send X10 commands via PowerLinc USB.
- USB-UIRT Actions to control USB-UIRT devices.
- Global Cache Actions (Girder Pro) to control Global Cache devices.
- NetRemote Actions to network with Promixis NetRemote.
- Serial Devices Actions (Girder Pro) to command serial devices.
- Voice Actions (Girder Pro) for voice prompts.
- X10 Actions (Girder Pro) for device mapped X10 Home Automation.
- Device Manager (Girder Pro) for unified home automation control.

9.1 Scripting Action

Scripting | On Error | States

Executes a Lua script specified on the Action Editor page. Optionally, the script may return a boolean.

© 2008 Promixis, LLC



Script Editor: Enter the Lua script here (see Scripting Girder).

If True: Tree Picker for node to branch to if the script returns true.

If False: Tree Picker for node to branch to if the script returns false.

Tip: You do not need to specify branches or to return anything from the script. By default, execution drops through to the next Action in a Macro or terminates.

Within the script the following global variables are defined.

EventString: String. The Event String of the triggering event.

EventDevice: Number. The Event Device number of the triggering event.

pld1 .. pld4: String or nul. The payload strings of the triggering event.

See Lua Language Reference and Lua Library Reference for full details of Lua programming.

Source: Built in.

9.2 Windows Actions

These Actions control application windows.

- Focus Action.
- Close Action.
- Show Action.
- Hide Action.
- Maximize Action.
- Minimize Action.
- Restore Action.
- Move Action.

© 2008 Promixis, LLC

- Move Relative Action.
- Resize Action.
- Center and Resize Action.
- Get Title Action.
- Copy Data Action.
- SendMessage Action.

9.2.1 Focus Action

Focus | On Error | States

Makes the Target Window active (similar to clicking on it). If the window is minimized, it will be restored and it will be brought in front of any windows overlapping it. You can also focus a control within an application, for example to enter text in a textbox. This action has no effect if the target window does not exist.

Window Picker: Select the window to focus.

Note: It is bad usability practice to change the focus without a specific command from the user. If she is typing in one window and suddenly without warning the focus switches to another, it will be very irritating. Windows goes to great lengths to prevent this and Girder goes to even greater lengths to allow it again. It is up to you to use this power wisely!

Usage:

This action can be used to switch between running applications using a remote. Add a Focus Action for each application and an Event Node set to different remote buttons for each application. See File Execute Action in the OS Actions group for an Action which can start an application that is not already running.

Source: Built in.

9.2.2 Close Action

Close | On Error | States

Sends a message to the Target Window requesting it to close. Similar to pressing the Close button in the window title bar or the Close item on the System Menu.

Window Picker: Select the window to close.

Tip: To *start* an application use the File Execute Action in the OS Actions group.

Usage:

A useful trick to conserve remote buttons is to use the same key to start and to close an application, but for safety to require a longer press to close it. Starting an application is covered under File Execute Action. To stop it, add a Close Action and set it to the application window. Add an Event Node to this and set it to the same remote button as used in the File Execute Action. However, un-check the **Down** modifier and check **Repeat** instead. You can adjust the time the button needs to be held down for by setting the **Anti Repeat**. A setting of 4000 means four seconds.

© 2008 Promixis, LLC

Source: Built in.

9.2.3 Show Action

Show | On Error | States

Makes the Target Window visible (see also Hide Action).

Window Picker: Select the window to show.

Source: Built in.

9.2.4 Hide Action

Hide | On Error | States

Hides the Target Window making it invisible without closing the application (see also Show Action).

Window Picker: Select the window to hide.

Source: Built in.

9.2.5 Maximize Action

Maximize | On Error | States

Sends a message to the Target Window requesting it to expand to full screen. Similar to pressing the Maximize button in the window title bar.

Window Picker: Select the window to maximize.

Source: Built in.

9.2.6 Minimize Action

Minimize | On Error | States

Sends a message to the Target Window requesting it to collapse to an icon or to the taskbar. Similar to pressing the Minimize button in the window title bar.

Window Picker: Select the window to minimize.

Source: Built in.

© 2008 Promixis, LLC

9.2.7 Restore Action

Restore | On Error | States

Sends a message to the Target Window requesting it to resume its previous state before it was Minimized. Similar to clicking the taskbar button.

Window Picker: Select the window to restore.

Source: Built in.

9.2.8 Move Action

Move | On Error | States

Moves the Target Window to a new position on the screen.

Window Picker: Select the window to move.

X: Pixel offset between the left side of the screen and the left side of the window.

Y: Pixel offset between the top of the screen and the top of the window.

Source: Built in.

9.2.9 Move Relative Action

Move Relative | On Error | States

Moves the Target Window by a specified number of pixels from its current position.

Window Picker: Select the window to move.

X: Number of pixels to move horizontally (negative values move left).

Y: Number of pixels to move vertically (negative values move up).

Source: Built in.

9.2.10 Resize Action

Resize | On Error | States

Resizes the Target Window to a specified number of pixels.

Window Picker: Select the window to resize.

Width: New width of the window in pixels.

Height: New height of the window in pixels.

© 2008 Promixis, LLC

Source: Built in.

9.2.11 Center and Resize Action

Center and Resize | On Error | States

Resizes the Target Window to a specified number of pixels and moves it to the center of the screen.

Window Picker: Select the window to center and resize.

Width: New width of the window in pixels.

Height: New height of the window in pixels.

Source: Built in.

9.2.12 Get Title Action

Get Title | On Error | States

Copies the title bar text of the Target Window into a specified Lua variable.

Window Picker: Select the window to get the title from.

Variable name for result: enter a valid Lua variable name.

Tip: You can examine the result of this action using the Variable Inspector (Press **F12**).

Source: Built in.

9.2.13 Copy Data Action

Copy Data | On Error | States

Sends a WM_COPYDATA message (See Windows API Documentation) to a window and optionally stores the result in a Lua variable.

Number (dwData): Numerical parameter for the message.

String (lpData}: String parameter for the message.

Save Result: Tick and supply a Lua variable name to save the result.

Targetting: Select the window to message.

Source: CopyData Plugin.

9.2.14 SendMessage Action

SendMessage | On Error | States

Uses SendMessage or PostMessage to send a message to a target window. See Windows or

© 2008 Promixis, LLC

Application documentation for details of Windows Messages. See also the Command Capture Action.

Message: The numerical message code.

wParam: The numerical word parameter.

lParam: The numerical long word parameter.

Return Variable: Lua variable name to receive the result (also tick **Save Result** and **Use SendMessage**).

Use SendMessage instead of PostMessage: Tick to use SendMessage (waits for response).

Target Button: Select the window to message.

Source: SendMessage Plugin.

9.3 Flow Control Actions

These actions provide for conditional or delayed execution of other actions.

- Wait Action delays execution of a macro.
- Window Exists? Action jumps on condition.
- Window is Foreground? Action jumps on condition.
- Checkbox Checked? Action jumps on checkbox state.
- Gosub Action runs another macro as a subroutine.
- Return Action returns from macro acting as subroutine.
- Stop Processing Event Action.

9.3.1 Wait Action

Wait | On Error | States

This action is only meaningful within a macro. It introduces a delay between completion of the action before it and starting the action after it in the macro. Optionally, the delay may be cut short by the appearance of the Target Window.

Window Picker: Select the window for the test below.

Wait For Window to open: Tick this to cut the delay short on appearance of the Target Window.

Maximum time to wait (ms): Length of the delay (1000 = one second).

Usage:

This action is useful when you want to first start an application and then immediately send it some initialization commands. Add a Macro and then add a File Execute Action to start the application. After this add a Wait Action, and place the Actions to configure the application (Keyboard Action or Command Capture Action for example) after the wait.

Some applications will respond to commands as soon as their window appears, while some require a further delay after this. Try setting the Wait Action to the application window and

© 2008 Promixis, LLC

checking **Wait For Window to open**. Set a worst-case startup time in **Maximum Time to Wait**. If the application does not respond to the initialization commands, try un-checking **Wait For Window to open** to use a fixed time delay.

If your application does require a further delay after the window opens, a good technique is to use two Wait Actions one after the other. Set the first one to **Wait For Window to open** and the second to a shorter fixed delay.

Source: Built in.

9.3.2 Window Exists? Action

Window Exists? | On Error | States

This action causes one or other specified actions or macros to be executed depending on whether the Target Window exists or does not exist.

Window Picker: Window for the test.

If Exists: Tree Picker to select the next action if the window exists.

If not Exists: Tree Picker to select the next action if the window does not exist.

Tip: Consider using a Window Conditional instead of this action.

Source: Built in.

9.3.3 Window is Foreground? Action

Window is Foreground? | On Error | States

This action causes one or other specified actions or macros to be executed depending on whether the Target Window is in the foreground (i.e. Is the active window awaiting input).

Window Picker: Window for the test.

If Is Foreground: Tree Picker to select the next action if the window is the foreground window.

If is not Foreground: Tree Picker to select the next action if the window is not the foreground window.

Tip: Consider using a Window Conditional instead of this action.

Source: Built in.

9.3.4 Checkbox Checked? Action

Checkbox Checked? | On Error | States

The Target Window must be a Checkbox control in an application window. This action causes

© 2008 Promixis, LLC

one or other specified actions or macros to be executed depending on whether the Checkbox is checked or not.

Checkbox Window Picker: Checkbox for the test.

If is Checked: Tree Picker to select the next action if the checkbox is checked.

If is not Checked: Tree Picker to select the next action if the checkbox is not checked.

Source: Built in.

9.3.5 Gosub Action

Gosub | On Error | States

This action is only meaningful within a macro. It causes another macro (or less usefully an action) to be executed and then execution to continue at the next action within the current macro. This can avoid the need to duplicate common sequences of actions in several macros.

Gosub: Tree Picker to select the macro to use as a subroutine.

Source: Built in.

9.3.6 Return Action

Return | On Error | States

This action is only meaningful within a macro used as the target of a Gosub Action. It causes an immediate return to the calling macro without executing any subsequent actions in the subroutine.

Source: Built in.

9.3.7 Stop Processing Event Action

Stop Processing Event | On Error | States

This action is only meaningful within a macro. It causes execution to terminate immediately without executing any subsequent actions.

Source: Built in.

9.4 Keyboard Actions

This action provides keyboard spoofing (sends keystrokes to an application).

- Keyboard Action
- Diamond Key Entry Action

© 2008 Promixis, LLC

9.4.1 Keyboard Action

Keyboard | On Error | States

Send specified keystrokes to the Target Window as if from the keyboard (keyboard spoofing).

Window Picker: Window to send the keystrokes to.

Text to send: The sequence of keystrokes to send (see encoding below).

Force keys: Tick if it does not work un-ticked.

Keystroke encoding

Typing (printable) keys are entered as lower-case characters. Non-printable keys are entered as a name enclosed in angle brackets (e.g. <F1>). Printable or non-printable keys may be prefixed by one or more modifier characters:

@: Alt Key.
^: Shift Key.
*: Control Key.
$: Windows Key.

The non-printable key names

ALT, BACKSPACE, DELETE, DOWN, END, ENTER, ESCAPE, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, HOME, INSERT, LEFT, PAGE_DOWN, PAGE_UP, RIGHT, SPACE, TAB, UP, PRINT_SCREEN, LWIN, RWIN, SCROLL_LOCK, NUM_LOCK, CTRL_BREAK, PAUSE, CAPS_LOCK, NUMPAD0, NUMPAD1, NUMPAD2, NUMPAD3, NUMPAD4, NUMPAD5, NUMPAD6, NUMPAD7, NUMPAD8, NUMPAD9, NUMPAD0, MULTIPLY, ADDITION, SUBTRACT, DECIMAL, DIVIDE, APPSKEY, LEFT_CTRL, RIGHT_CTRL, LEFT_ALT, RIGHT_ALT, LEFT_SHIFT, RIGHT_SHIFT, SLEEP, NUMPADENTER, BROWSER_BACK, BROWSER_FORWARD, BROWSER_REFRESH, BROWSER_STOP, BROWSER_SEARCH, BROWSER_FAVOURITES, BROWSER_HOME, VOLUME_MUTE, VOLUME_DOWN, VOLUME_UP, MEDIA_NEXT, MEDIA_PREV, MEDIA_STOP, MEDIA_PLAY_PAUSE, LAUNCH_APP1, LAUNCH_APP2, CTRL_DOWN, CTRL_UP, ALT_DOWN, ALT_UP, SHIFT_DOWN, SHIFT_UP, LWIN_DOWN, LWIN_UP, RWIN_DOWN, RWIN_UP, ASCII.

NOTE: Keyboard spoofing is tricky and can be unreliable. If sending keys to the top-level window does not work, try sending them to a relevant child window. For example, typing to the top-level Notepad window will not work, but the "Edit" class child window will work. As a last resort, Focus the window using the Focus Action and select **Match foreground task** as the window target for the Keyboard Action.

Source: Built in.

9.4.2 Diamond Key Entry Action

Diamond Key Entry | On Error | States

This action is used to link a remote to the OSD Diamond Keyboard. The action has a set of sub-actions as follows.

Up, Down, Left, Right: The arrow key actions that narrow down the key selection.

Undo: Reverses the previous selection step.

Show: Shows the OSD without narrowing down the selection.

© 2008 Promixis, LLC

Hide: Hides the OSD without making a selection.

Source: Diamond Key Plugin.

9.5 Mouse Actions

These actions provide mouse spoofing (control the position of the mouse pointer and provide emulation of mouse button operations).

- Move Relative Action moves the pointer by an amount.
- Move Absolute Action moves the pointer to a position.
- Mouse Left Click Action.
- Mouse Left Double Click Action.
- Clicked (Targeted) Action clicks a specific window.
- Double Clicked (Targeted) Action double clicks a specific window.
- Mouse Middle Click Action.
- Mouse Middle Double Click Action.
- Mouse Right Click Action.
- Mouse Right Double Click Action.
- Mouse Wheel Down Action.
- Mouse Wheel Up Action.
- Mouse Movement Actions for hooking up mouse control remotes.

9.5.1 Move Relative Action

Move Relative | On Error | States

Moves the mouse pointer by a specified number of pixels from its current position.

X: Number of pixels horizontally (negative values are left).

Y: Number of pixels vertically (negative values are up).

Source: Built in.

9.5.2 Move Absolute Action

Move Absolute | On Error | States

Moves the mouse pointer to a specified position on the screen.

X: Number of pixels to the right of the left hand edge.

Y: Number of pixels down from the top.

Source: Built in.

© 2008 Promixis, LLC

9.5.3 Mouse Left Click Action

Mouse Left Click | On Error | States

Simulates clicking the left mouse button.

Source: Built in.

9.5.4 Mouse Left Double Click Action

Mouse Left Double Click | On Error | States

Simulates double-clicking the left mouse button.

Source: Built in.

9.5.5 Clicked (Targeted) Action

Clicked (Targeted) | On Error | States

Simulates clicking the left mouse button on the Target Window. The targeted window may be a control, so this action can be used to press a button, for example.

Window Picker: Window to click on.

Source: Built in.

9.5.6 Double Clicked (Targeted) Action

Double Clicked (Targeted) | On Error | States

Simulates double-clicking the left mouse button on the Target Window.

Window Picker: Window to double-click on.

Source: Built in.

9.5.7 Mouse Middle Click Action

Mouse Middle Click | On Error | States

Simulates clicking the middle mouse button.

Source: Built in.

© 2008 Promixis, LLC

9.5.8 Mouse Middle Double Click Action

Mouse Middle Double Click | On Error | States

Simulates double-clicking the middle mouse button.

Source: Built in.

9.5.9 Mouse Right Click Action

Mouse Right Click | On Error | States

Simulates clicking the right mouse button.

Source: Built in.

9.5.10 Mouse Right Double Click Action

Mouse Right Double Click | On Error | States

Simulates double-clicking the right mouse button.

Source: Built in.

9.5.11 Mouse Wheel Down Action

Mouse Wheel Down | On Error | States

Simulates rolling the mouse wheel down by one click.

Source: Built in.

9.5.12 Mouse Wheel Up Action

Mouse Wheel Up | On Error | States

Simulates rolling the mouse wheel up by one click.

Source: Built in.

9.5.13 Mouse Movement Actions

Mouse Right, Mouse Up Right, Mouse Up Left, Mouse Down Right, Mouse Down Left, Mouse Left, Mouse Up, Mouse Down | On Error | States

Moves the mouse pointer in one of eight compass directions.

Initial Stepsize: Number of pixels of movement per event initially.

© 2008 Promixis, LLC

Typematic: Time in milliseconds before rate changes to Big Step.

Timeout: Time in milliseconds between events required to reset rate to Initial Stepsize.

Big Step: Number of pixels of movement per event after Typematic delay expires.

Note

These actions are intended to be hooked to events from directional buttons on controllers. The Down and the Repeat modifiers should be ticked on the event. Holding down the directional button will then move the mouse pointer first slowly for fine control and then faster for responsive movement.

Source: Built in.

9.6 Volume Actions

These actions control the computers audio settings.

- Change Volume Action.
- Get Volume Action.
- Display Volume OSD Action.
- Display Mute OSD Action.
- Change Mute Action.
- Change Volume Right Action.
- Change Volume Left Action.
- Change Balance Action.
- Set Volume Action.
- Set Balance Action.

9.6.1 Change Volume Action

Change Volume | On Error | States

Changes the level of an audio control incrementally.

Audio Picker: Select the audio control to change.

Volume Level: Percentage by which to change the audio control, positive or negative.

On Screen Display: Tick to show an on-screen display of the audio level.

Timeout (ms): Time the On Screen Display shows for.

Tip: Experiment with Volume Actions by using the Test Action button in Girder whilst observing the effect in the Windows Volume Control applet. The precise result of these actions may vary depending on audio hardware and driver software.

Source: Built in.

© 2008 Promixis, LLC

9.6.2 Get Volume Action

Get Volume | On Error | States

Sets a Lua variable (see Scripting Girder) to the percentage level of an audio control.

Audio Picker: Select the audio control to get.

Variable: The name of the Lua variable that receives the level percentage.

Source: Built in.

9.6.3 Display Volume OSD Action

Display Volume OSD | On Error | States

Displays the level of an audio control on the On Screen Display.

Audio Picker: Select the audio control to display.

Timeout (ms): Time the On Screen Display shows for.

Source: Built in.

9.6.4 Display Mute OSD Action

Display Mute OSD | On Error | States

Displays the state of an audio control using mute-style on/off symbols.

Audio Picker: Select the audio control to display.

Timeout (ms): Time the On Screen Display shows for.

Source: Built in.

9.6.5 Change Mute Action

Change Mute | On Error | States

Changes the state of a mute type audio control.

Audio Picker: Select the audio control to change.

Mute: 0 to toggle state, -1 to mute (off), 1 to unmute (on).

On Screen Display: Tick to show an on-screen display of the audio level.

Timeout (ms): Time the On Screen Display shows for.

Tip: A common mistake is to connect this action to a volume or level control. This will result in switching between maximum and minimum level. Controls with words like "mute" and "select" in their labels are generally suitable.

© 2008 Promixis, LLC

Tip: Experiment with Volume Actions by using the Test Action button in Girder whilst observing the effect in the Windows Volume Control applet. The precise result of these actions may vary depending on audio hardware and driver software.

Source: Built in.

9.6.6 Change Volume Right Action

Change Volume Right | On Error | States

Changes the level of the Right channel of a stereo audio control.

Audio Picker: Select the audio control to change.

Right Volume Level: Percentage by which to change the audio control, positive or negative.

On Screen Display: Tick to show an on-screen display of the audio level.

Timeout (ms): Time the On Screen Display shows for.

Tip: Experiment with Volume Actions by using the Test Action button in Girder whilst observing the effect in the Windows Volume Control applet. The precise result of these actions may vary depending on audio hardware and driver software.

Source: Built in.

9.6.7 Change Volume Left Action

Change Volume Left | On Error | States

Changes the level of the Left channel of a stereo audio control.

Audio Picker: Select the audio control to change.

Left Volume Level: Percentage by which to change the audio control, positive or negative.

On Screen Display: Tick to show an on-screen display of the audio level.

Timeout (ms): Time the On Screen Display shows for.

Tip: Experiment with Volume Actions by using the Test Action button in Girder whilst observing the effect in the Windows Volume Control applet. The precise result of these actions may vary depending on audio hardware and driver software.

Source: Built in.

9.6.8 Change Balance Action

Change Balance | On Error | States

Changes the level of the Left and Right channels of a stereo audio control such as to move the audio image while maintaining the overall volume.

© 2008 Promixis, LLC

Audio Picker: Select the audio control to change.

Volume Level: Increment by which to move the balance, there are 65535 steps between centred and fully right or left. Negative increments move to the right, positive to the left.

On Screen Display: Tick to show an on-screen display of the audio level.

Timeout (ms): Time the On Screen Display shows for.

Tip: Experiment with Volume Actions by using the Test Action button in Girder whilst observing the effect in the Windows Volume Control applet. The precise result of these actions may vary depending on audio hardware and driver software.

Source: Built in.

9.6.9 Set Volume Action

Set Volume | On Error | States

Sets the level of a volume control to the specified percentage.

Audio Picker: Select the audio control to set.

Volume Level: Percentage volume, 0-100.

On Screen Display: Tick to show an on-screen display of the volume level.

Timeout (ms): Time the On Screen Display shows for.

Tip: Experiment with Volume Actions by using the Test Action button in Girder whilst observing the effect in the Windows Volume Control applet. The precise result of these actions may vary depending on audio hardware and driver software.

Source: Built in.

9.6.10 Set Balance Action

Set Balance | On Error | States

Sets the level of the Left and Right channels of a stereo audio control such as to move the audio image to a set position whilst maintaining the overall volume constant.

Audio Picker: Select the audio control to set.

Balance: -100 full left, 0 center, +100 full right.

On Screen Display: Tick to show an on-screen display of the balance.

Timeout (ms): Time the On Screen Display shows for.

Tip: Experiment with Volume Actions by using the Test Action button in Girder whilst observing the effect in the Windows Volume Control applet. The precise result of these actions may vary depending on audio hardware and driver software.

© 2008 Promixis, LLC

Source: Built in.

9.7 Girder Actions Group

These actions control Girder itself.

- Enable Group/Action Action.
- Disable Group/Action Action.
- Enable Plugin Action.
- Disable Plugin Action.
- Reset State Action.
- Get Tick Count Action.
- Reset Tick Count Action.
- Variable to Clipboard Action.
- Clipboard to Variable Action.
- Reset all state counts Action.
- Get Top Action.
- Set State Action.
- Event Translation Action (Girder Pro).
- On Screen Text Display Action.

9.7.1 Enable Group/Action Action

Enable Group/Action | On Error | States

Enables a selected Group, Macro or Action.

Browse: Tree Picker to select the node to be enabled.

Source: Built in.

9.7.2 Disable Group/Action Action

Disable Group/Action | On Error | States

Disables a selected Group, Macro or Action.

Browse: Tree Picker to select the node to be disabled.

Source: Built in.

9.7.3 Enable Plugin Action

Enable Plugin | On Error | States

Enables a selected Plugin.

Plugin ID: The ID number of the plugin identified on the Plugin Settings dialog.

© 2008 Promixis, LLC

Source: Built in.

9.7.4 Disable Plugin Action

Disable Plugin | On Error | States

Disables a selected Plugin.

Plugin ID: The ID number of the plugin identified on the Plugin Settings dialog.

Source: Built in.

9.7.5 Reset State Action

Reset State | On Error | States

Resets the state of an action.

Source: Built in.

9.7.6 Get Tick Count Action

Get Tick Count | On Error | States

Saves the current timer tick count in a Lua variable.

Variable Name: Lua variable name.

Source: Built in.

9.7.7 Reset Tick Count Action

Reset Tick Count | On Error | States

Resets the timer.

Source: Built in.

9.7.8 Variable to Clipboard Action

Variable to Clipboard | On Error | States

Copies the value of a Lua variable to the system clipboard.

Variable Name: Lua variable name.

Source: Built in.

© 2008 Promixis, LLC

9.7.9 Clipboard to Variable Action

Clipboard to Variable | On Error | States

Copies the system clipboard to a Lua variable.

Variable Name: Lua variable name.

Source: Built in.

9.7.10 Reset all state counts Action

Reset all state counts | On Error | States

Resets the states of all actions.

Source: Built in.

9.7.11 Get Top Action

Get Top | On Error | States

Reports the Lua top of stack pointer in the status bar at the bottom of the Girder window. Normally this should be 0.

Source: Built in.

9.7.12 Set State Action

Set State | On Error | States

Sets the state of the selected node to the value given.

State Value: Value to set the state to.

Tree Picker: Select the node to set the state on.

Source: Built in.

9.7.13 Event Translation Action

Event Translation | On Error | States

Requires Girder Pro.

Generates another Event for each Event which triggers it. The generated Event may inherit elements from the triggering Event. This provides an alternate method, similar to event mapping, of mapping events from physical devices onto abstract events in a GML file intended for distribution.

© 2008 Promixis, LLC

CAUTION: Take care not to generate an infinite loop by generating an Event which will itself be recognized for translation by this Action. Normally, generated Events should have a different device number to the triggering Event.

Event String: This is the event to be generated. Token [EventString] can be used to insert the event string from the triggering Event.

Event Device: The device number for the generated Event. Use 0 to inherit the device number from the triggering Event.

Event Modifier: The event modifier for the generated Event. Set to "From Trigger" to inherit from the triggering device. The token [EventMod] can be used to paste the modifier from the triggering Event in string form into other elements of the generated Event.

Payload 1 .. 4: The event payloads for the generated Event. The payloads from the triggering Event are available in tokens [pld1] .. [pld4].

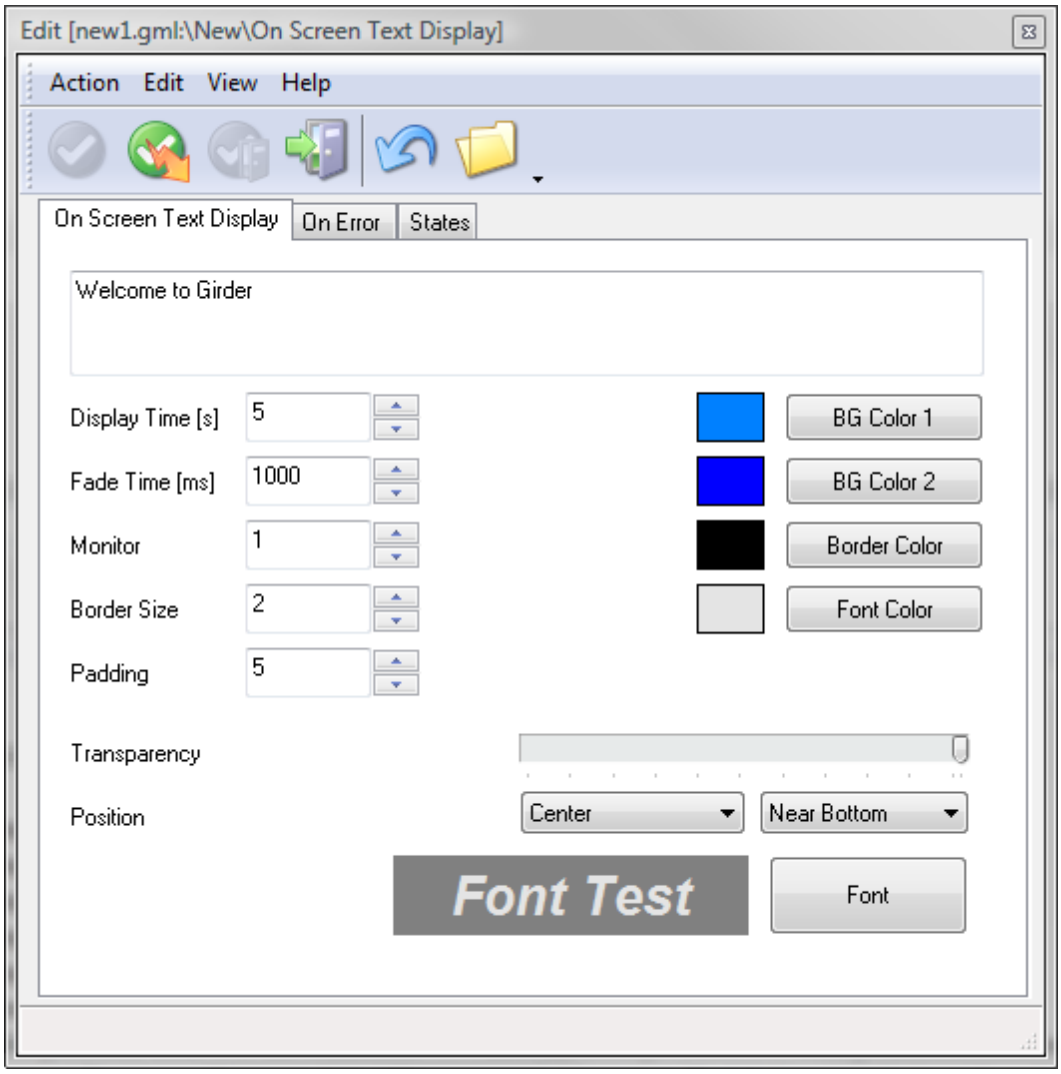
Discard the Triggering Event after this Action: If this is ticked, the triggering Event will be abandoned and not offered for matching to any further Actions in the tree.

Source: Tree Script Plugin EventExtras.lua.

9.7.14 On Screen Text Display Action

On Screen Text Display | On Error | States

© 2008 Promixis, LLC



Shows specified text in an on screen display window.

Top Box: Enter the text to be shown. You can substitute Lua variables (see Scripting Girder) in the text by specifying their name in square brackets (i.e. [LuaVar]).
Display Time: The time the OSD will be visible on screen in seconds.
Fade Time: The time the OSD will take to fade out in milli seconds (1000 is one second)
Monitor: Number of the monitor that the OSD should appear on. (1 is default, main monitor)
Border Size: Border size in pixels.
Padding: Padding inside border around text in pixels.
BG Color 1: The background color is a gradient going from BG Color 1 to BG Color 2.
BG Color 2: The end gradient Background Color.
Border Color: The color of the border.
Font Color: The Color of the Font.
Transparency: The transparency of the Window (all the way to the left is invisible, all the way to the right is opaque)
Position: The position of the OSD.
Font: The font and font size to use.

Source: Tree Script Plugin.

© 2008 Promixis, LLC

9.8 Monitor Actions

These actions allow control of the video display monitor.

- Get Resolution Action.
- Set Resolution Action.
- Screensaver Action.
- Monitor Power Management Action.

9.8.1 Get Resolution Action

Get Resolution | On Error | States

Saves the resolution of the screen to a Lua string variable (see Scripting Girder).

Variable name: Lua variable.

The value of the variable will be of the form width**x**height**x**colorbits**@**refreshrate, for example 1400x1050x32@60.

Source: Built in.

9.8.2 Set Resolution Action

Set Resolution | On Error | States

Sets the resolution of the screen.

Resolution: String of the form width**x**height**x**colorbits**@**refreshrate, for example 1400x1050x32@60.

Source: Built in.

9.8.3 Screensaver Action

Screensaver | On Error | States

Triggers the monitor screensaver if one is configured in Windows.

Source: Built in.

9.8.4 Monitor Power Management Action

Monitor Power Management | On Error | States

Sets the monitor power management state.

Action: On, Low Power or Off.

Method: Two ways of doing it. If one does not work, try the other.

© 2008 Promixis, LLC

Source: Multimonitor Extensions Plugin.

9.9 OS Actions

These actions allow control of Operating System functions.

- Shut down Action.
- Log off Action.
- Restart Action.
- Standby/Hibernate Action.
- Task Switch Action.
- File Execute Action.
- Enhanced TaskSwitcher Action.

9.9.1 Shut down Action

Shut down | On Error | States

Requests a computer shutdown (similar to "Turn off computer" on the Start menu).

Source: Built in.

9.9.2 Log off Action

Log off | On Error | States

Logs the current user out (similar to "Log off" on the Start menu).

Source: Built in.

9.9.3 Restart Action

Restart | On Error | States

Requests the computer to restart (re-boot).

Source: Built in.

9.9.4 Standby/Hibernate Action

Standby/Hibernate | On Error | States

Causes the computer to enter a standby or hibernate state to save power.

Mode: Standby (polite), Hibernate (polite), Standby (forced) or Hibernate (forced).

Source: Built in.

© 2008 Promixis, LLC

9.9.5 Task Switch Action

Task Switch | On Error | States

Shows the Task Switcher OSD for a short time. Enhanced TaskSwitcher Action is more useful.

Source: Built in.

9.9.6 File Execute Action

File Execute | On Error | States

Starts an executable (application) on the computer.

File to Execute: Path and name of the file to be executed.

Parameters: Any switches or parameters that should follow the file name (see documentation for application).

Window State: The start up state of the application window.

Note: It is a good idea to avoid using explicit directory paths particularly if your GML file is intended to be used by others. There are a number of ways of achieving this.

1. Many Windows components will run without an explicit path. For example "wmplayer.exe" starts WMP.
2. Command line symbols are expanded if available. For example "%GIRDER%designer.exe" starts the Girder DUI designer.
3. Lua variable names enclosed in square brackets are expanded to their values, so it is possible to have a start-up script work out the path and save it to a variable (usually recovering the path from the Windows Registry with the win Library Registry object).

Usage:

A common requirement is to have a remote button start an application if it is not already running or bring it to the foreground if it is. This can be achieved with a File Execute action and a Focus Action triggered by the same event. Set the File Execute to the executable name and path of the application and set the Focus to the main window of the application using the Window Picker.

Sometimes this will cause multiple instances of the application to run. If you want to avoid this, add a Window Conditional to the File Execute node, set it to the main window of the application, the **Condition** to "Window Exists" and tick the **Invert** box.

Source: Built in.

9.9.7 Enhanced TaskSwitcher Action

Enhanced TaskSwitcher | On Error | States

Controls the Enhanced TaskSwitcher On Screen Display.

Tab to Next: Select the next task.

Tab to Previous: Select the previous task.

© 2008 Promixis, LLC

Focus Window: Show the OSD.

Cancel: Hide the OSD.

Source: Enhanced TaskSwitcher Plugin.

9.10 Command Capture Actions

This action provides message spoofing (the ability to emulate Windows messages to and within an application).

- Command Capture Action.

9.10.1 Command Capture Action

Command Capture | On Error | States

Sends a Windows Message to the Target Window. Also provides a reverse engineering tool to discover the message codes for an application.

Capture Settings: Brings up the Command Capture to generate the settings for the rest of this page by logging Windows messages.

Window Picker: Window to send the message to (you can use this to view or refine the settings supplied by Capture Settings).

Message: The message code.

wParam: Word (16 bit) parameter.

lParam: Long (32 bit) parameter.

Send Method: Either SendMessage (which waits for a response) or PostMessage (which fires-and-forgets).

Result Variable: Lua variable (see Scripting Girder) to receive the result of SendMessage.

Discovering Message Codes

Sometimes the Command Capture Tool is not the best way of discovering message codes for the Command Capture Action, which is why the values can be entered manually. Many applications use standard Windows control libraries for elements such as Buttons, Check Boxes, Radio Buttons and Text Boxes. These controls are implemented as child windows which can be targeted with a standard set of messages.

Microsoft's online MSDN Library is a good source of information.

http://msdn.microsoft.com/library/

Search for "CreateWindow" for a list of the different standard control window classes available to Windows developers and follow the links for the specific control type of interest (note that Check Boxes are implemented using the BUTTON class. Follow the links through to "Button Messages" for the information needed.

For example, a "BM_SETCHECK" message can be used to check or uncheck a checkbox control. BM_SETCHECK is a "C" language symbol and the wParam uses one of three more symbols "BST_CHECKED", "BST_INDETERMINATE" and "BST_UNCHECKED". We need to decode these

© 2008 Promixis, LLC

symbols into decimal numbers. The symbols are defined in a Windows API file called "WinUser.h" which is shipped with most development systems, or a Google search will quickly reveal a copy online.

Searching within the file we find "#define BM_SETCHECK 0x00F1". This is in hexadecimal, as indicated by the "0x", and the leading zeros can be dropped giving "F1". Windows Calculator in Scientific mode will convert this to decimal 241, which is the entry for the "Message" value. Similarly, BST_CHECKED is decimal "1" which is the value for the wParam.

So executing Command Capture Action, targeted on a checkbox child window, with Message = 241 and wParam = 1 should result in that checkbox being checked.

Similarly, Message = 240 (BM_GETCHECK), wParam = 0, lParam = 0, using SendMessage and specifying a Lua variable for the result will result in the Lua variable being set to 0 (BST_UNCHECKED), 1 (BST_CHECKED) or 2 (BST_INDETERMINATE).

Source: Built in.

9.10.1.1 Command Capture Tool

The command capture tool, accessed from the Command Capture Action editor, enables you to discover the internal windows messages that an application uses. You can then take control of the application by sending these messages using the Command Capture Action.

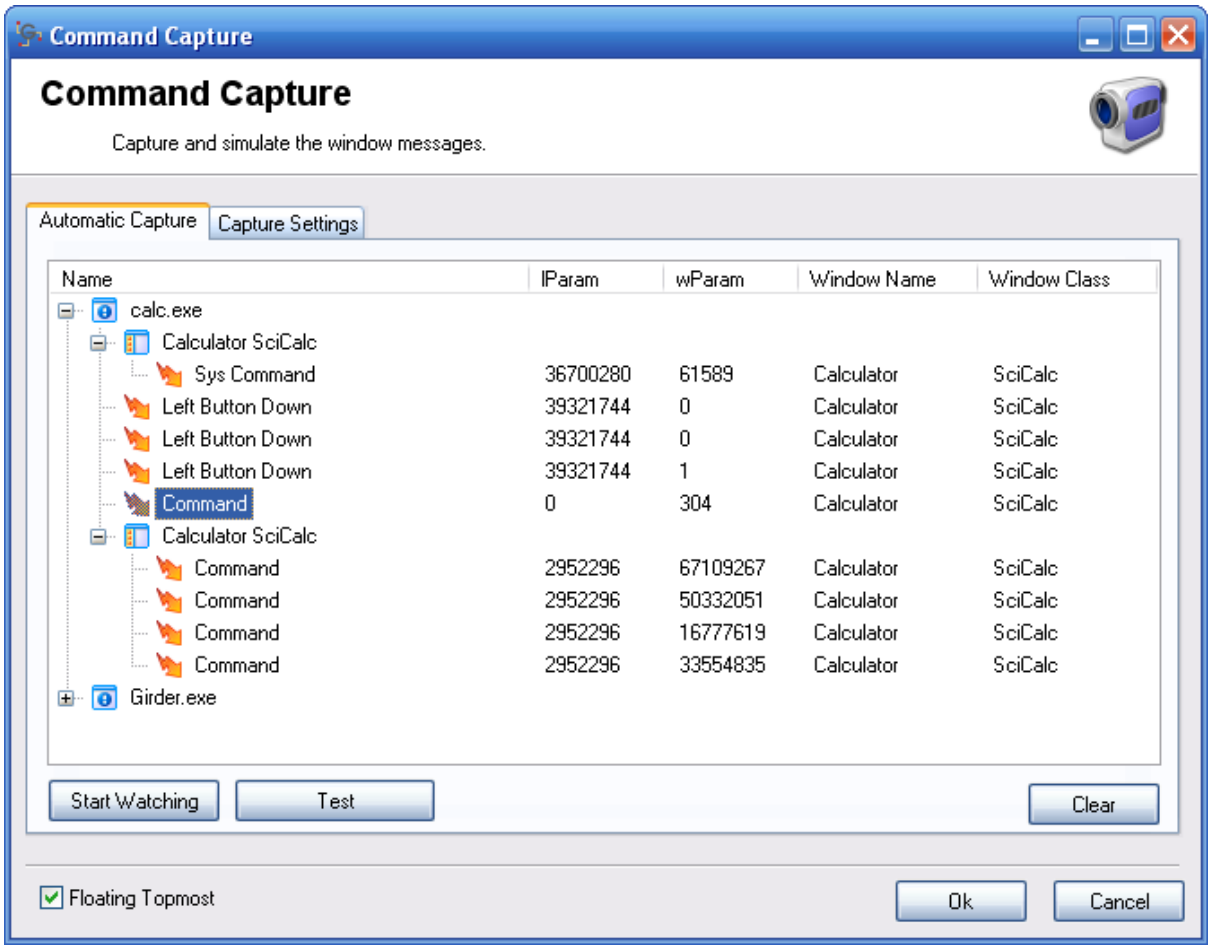
Capture Settings Tab

Full Capture of application: Tick to capture all messages for a selected application. Pick the application from the drop-down list. Press **Refresh** if the application is not on the list.

Report WM_TIMER Messages: Some applications generate timer messages frequently which would swamp the list. Normally you should leave this un-ticked for clarity, but you can tick this option to see these messages.

Automatic Capture Tab

© 2008 Promixis, LLC



Press **Start Watching** and then perform the desired action in the application manually. Press **Stop Watching**. Expand the message lists. You can highlight a message and press **Test** to see the effect of sending that message to the application.

Tip: **Command** messages are the most useful for automation. Many applications use these to link between their toolbars, menus and shortcuts and the application functionality.

Press **OK** to transfer the selected message and target window settings to the Command Capture Action editor.

9.11 Miscellaneous Actions

- Play Wav Action plays a sound file.
- Talking clock Action.
- SimpleTimer Action.

9.11.1 Play Wav Action

Play Wav | On Error | States

Plays a Wav (audio) file through the computers audio system.

File To Play: The path and name to the file to be played.

© 2008 Promixis, LLC

Source: Built in.

9.11.2 Talking clock Action

Talking clock | On Error | States

Speaks the current time through the computers audio system.

Source: Built in.

9.11.3 SimpleTimer Action

SimpleTimer | On Error | States

Allows an action or macro to be run after a fixed delay or repeatedly at fixed intervals.

Timer ID: 50 timers are available. Select which timer to use.

Timeout(ms}: Enter the timer interval in milliseconds to start the timer. Enter 0 to stop the timer.

Recurring: Tick if the timer should restart automatically when it expires.

On Start: Tree Picker to select any action or macro to run when the timer is started.

On Cancel: Tree Picker to select any action or macro to run when the timer is canceled.

On Timer: Tree Picker to select the action or macro to run when the timeout interval is reached.

Source: SimpleTimer Plugin.

9.12 Girder 3 Legacy Actions

These actions are provided for compatibility with Girder 3. They have been superseded by new, more powerful Actions as follows.

Built-in Legacy Actions

Simple OSD: use On Screen Text Display Action
Shut down: use Shut down Action
Command: use Command Capture Action
Sys Command: use Command Capture Action
Left Click: use Mouse Left Click Action
Right Click: use Mouse Right Click Action
Left Double Click: use Mouse Left Double Click Action
Right Double Click: use Mouse Right Double Click Action
App Command: use Command Capture Action
Keyboard(G3): use Keyboard Action

Provided by the SendMessage Plugin

See SendMessage Plugin Topic.

© 2008 Promixis, LLC

SendMessage Legacy: use Command Capture Action

9.13 Scheduler Actions

Requires Girder Pro.

These Actions generate periodic events at set times.

NB: These actions are deprecated. You are strongly recommended to use the **Event Scheduler** in the Automation section of the Settings dialog to define schedulers. See Scheduler in the Home Automation Applications section.

- Hourly Schedule Action.
- Daily Schedule Action.
- Weekly Schedule Action.
- Monthly Schedule Action.
- Sunset Schedule Action.
- Sunrise Schedule Action.

Tip: You must trigger these actions to install the scheduler which is bound to the Lua system using the SchedulerName. This must be unique for each scheduler. Best way is to attach a Girder Events/ScriptEnable Event Node to the action so it gets executed every time the Lua scripting system starts. After creating a new action, press the Test Action button to install it for the first time.

Source: Scheduler Plugin

9.14 Ambient Light Level Actions

Requires Girder Pro.

These actions are used to change the light level in a zone which must have been previously defined on the Light Zones tab of the Scheduler Automation Application on the Settings dialog.

- Set Area Light Level Action.
- Adjust Area Light Level Action.

9.14.1 Set Area Light Level Action

Set Area Light Level Action | On Error | States

Requires Girder Pro.

This action is used to change the light level in a zone which must have been previously defined on the Light Zones tab of the Scheduler Automation Application on the Settings dialog.

Area: Pick the Light Zone to be set.

Set Light Level: Check the level to set the zone to.

Source: Tree Script Plugin AmbientLight UI.lua.

© 2008 Promixis, LLC

9.14.2 Adjust Area Light Level Action

Adjust Area Light Level Action | On Error | States

Requires Girder Pro.

This action is used to change the light level in a zone which must have been previously defined on the Light Zones tab of the Scheduler Automation Application on the Settings dialog.

Area: Pick the Light Zone to be adjusted.

Amount: Set the number of steps to change the light level by.

For example, if the light level is "Little Dark", adjusting it by +2 would result in "Mostly Light" or -3 would give "Full Dark" because the level "saturates" at this value.

Source: Tree Script Plugin AmbientLight UI.lua.

9.15 Girder to Girder Actions

Requires Girder Pro.

- Remote Event Pump Action.
- Remote Scripting Action.
- Remote Event Action.
- Push GML Action.
- Check Remote GML Version Action.

9.15.1 Remote Event Pump Action

Remote Event Pump | On Error | States

Requires Girder Pro.

Sends any Events which trigger this action to a specified remote Girder or to all remote Girders over the network.

Server Name or IP Address of Remote Girder: The pick-list offers all Girder clients that are currently online. You can also type names or IP addresses of Girder clients not presently online.

Broadcast to all available Girder Clients: It this is ticked, the above is ignored and the Event is sent to all clients which are online when the Action is triggered.

Prefix for remote Event String: Any text in this box is prefixed to the event string of the event which triggers the Action.

Remote Event Device: This can be set to zero to retain the triggering device number, or specify a different device number to use.

Local Event generated if Action fails: A local Event with the Communication Server device number (232) and this event string will be generated if generating the remote event fails.

Source: Tree Script Plugin G2G UI.lua.

© 2008 Promixis, LLC

9.15.2 Remote Scripting Action

Remote Scripting | On Error | States

Requires Girder Pro.

Executes a Lua script on one or all remote Girders. If the script has errors, they will be raised in the local as well as the remote Girder Lua Console.

Server Name or IP Address of Remote Girder: The pick-list offers all Girder clients that are currently online. You can also type names or IP addresses of Girder clients not presently online.

Broadcast to all available Girder Clients: It this is ticked, the above is ignored and the Event is sent to all clients which are online when the Action is triggered.

Script: Enter the Lua script. Functions from the girder2girder Library can be used in this script to send data back "home".

Source: Tree Script Plugin G2G UI.lua.

9.15.3 Remote Event Action

Remote Event | On Error | States

Requires Girder Pro.

Generates an Event on one or more remote Girders.

Server Name or IP Address of Remote Girder: The pick-list offers all Girder clients that are currently online. You can also type names or IP addresses of Girder clients not presently online.

Broadcast to all available Girder Clients: It this is ticked, the above is ignored and the Event is sent to all clients which are online when the Action is triggered.

Event String: The event string of the Event to be raised remotely.

Event Device: The device number of the Event to be raised remotely. Default 232, Communications Server.

Event Modifier: The modifier of the Event to be raised remotely.

Payload 1 .. 4: Payloads of the Event to be raised remotely.

Source: Tree Script Plugin G2G UI.lua.

9.15.4 Push GML Action

Push GML | On Error | States

Requires Girder Pro.

Copies a specified GML file to one or more remote Girders and loads it into Girder. Can also be used to update a GML file that is already loaded.

Server Name or IP Address of Remote Girder: The pick-list offers all Girder clients that are

© 2008 Promixis, LLC

currently online. You can also type names or IP addresses of Girder clients not presently online.

Broadcast to all available Girder Clients: It this is ticked, the above is ignored and the file is sent to all clients which are online when the Action is triggered.

GML File: Provides a file picker to select the GML file, or the file path and name may be typed into the edit box.

Source: Tree Script Plugin G2G UI.lua.

9.15.5 Check Remote GML Version Action

Check Remote GML Version | On Error | States

Requires Girder Pro.

Tests one or more remote Girders for the version and presence of a specified GML file. The test uses the GUID and Version of a specified local GML file as the reference point for the test. One of three Events on the "Lua: Girder to Girder" Event Device is generated per client. **GMLOK** is generated if the GML file is loaded and has the same version number. **GMLNotLoaded** is generated if the GML file is not loaded. **GMLOutdated** is generated if the GML file is loaded but has a different version number. The first parameter of all the Events is the Client name, the second is the GML file internal name and the third (not present for **GMLNotLoaded**) is the version number of the GML file in the client.

Server Name or IP Address of Remote Girder: The pick-list offers all Girder clients that are currently online. You can also type names or IP addresses of Girder clients not presently online.

Broadcast to all available Girder Clients: It this is ticked, the above is ignored and the test is done on all clients which are online when the Action is triggered.

GML File: Provides a file picker to select the reference GML file, or the file path and name may be typed into the edit box. The GUID and version number are displayed if a valid GML file is selected.

Source: Tree Script Plugin G2G UI.lua.

9.16 SlinkE Actions

This Action sends Consumer IR commands using a Slink-e control device.

- Send IR (SlinkE) Action.

9.16.1 Send IR (SlinkE) Action

Send IR | On Error | States

Sends an IR command via a Slink-e controller.

Carrier(Hz): Enter the carrier frequency in cycles per second of the signal.

Ports: Tick the boxes for the port or ports on which the signal should be transmitted.

Signal: Copy the IR signal string into this box.

© 2008 Promixis, LLC

Source: SlinkE Plugin

9.17 X10 Controls Actions

These Actions send X10 home automation commands using CM1x interface devices. In Girder Pro, you may find the X10 Actions easier to use and more powerful.

- X10 On Action.
- X10 Off Action.
- X10 Dim Action.
- X10 Brighten Action.
- X10 Device Status Action.
- X10 All Lights On Action.
- X10 All Lights Off Action.

9.17.1 X10 On Action

On | On Error | States

Sends X10 commands via CM1x to turn on an individual unit.

House Code: Alphabetic part of the X10 address A..P.

Device Number: Numeric part of the X10 address 1..16.

Source: X10-CM1X Plugin

9.17.2 X10 Off Action

Off | On Error | States

Sends X10 commands via CM1x to turn off an individual unit.

House Code: Alphabetic part of the X10 address A..P.

Device Number: Numeric part of the X10 address 1..16.

Source: X10-CM1X Plugin

9.17.3 X10 Dim Action

Dim | On Error | States

Sends X10 commands via CM1x to reduce the brightness of an individual X10 unit.

House Code: Alphabetic part of the X10 address A..P.

Device Number: Numeric part of the X10 address 1..16.

Dim step size: Number of increments to reduce the brightness by.

© 2008 Promixis, LLC

Source: X10-CM1X Plugin

9.17.4 X10 Brighten Action

Brighten | On Error | States

Sends X10 commands via CM1x to increase the brightness of an individual X10 unit.

House Code: Alphabetic part of the X10 address A..P.

Device Number: Numeric part of the X10 address 1..16.

Dim step size: Number of increments to increase the brightness by.

Source: X10-CM1X Plugin

9.17.5 X10 Device Status Action

Device Status | On Error | States

Sends X10 commands via CM1x to interrogate the status of an individual X10 unit.

House Code: Alphabetic part of the X10 address A..P.

Device Number: Numeric part of the X10 address 1..16.

Tip: The response comes in as a Girder Event.

Source: X10-CM1X Plugin

9.17.6 X10 All Lights On Action

All Lights On | On Error | States

Sends an X10 command via CM1x to turn all lights on a house code on.

House Code: Alphabetic part of the X10 address A..P.

Source: X10-CM1X Plugin

9.17.7 X10 All Lights Off Action

All Lights Off | On Error | States

Sends an X10 command via CM1x to turn all lights on a house code off.

House Code: Alphabetic part of the X10 address A..P.

Source: X10-CM1X Plugin

© 2008 Promixis, LLC

9.18 IRTrans Actions

- Send IR Code via irserver Action

9.18.1 Send IR Code via irserver Action

Send IR Code via irserver Action | On Error | States

Remote: Select the Remote to use.

Command: Select the command to send.

Bus: Select the device bus.

Addressmask: Enter the device address mask.

LED Select: Select the LED.

Source: IRTrans Plugin

9.19 PowerlincUSB Actions

This Action sends X10 home automation commands using the Powerlinc USB interface device. In Girder Pro, you may find the X10 Actions easier to use and more powerful.

- SendX10 Action

9.19.1 SendX10 Action

Send X10 Action | On Error | States

Send an X10 command out using a PowerLinc USB device.

House Code: Enter the house code address letter.

Unit Code: Enter the unit code address number.

Command: Select the command to send.

Source: PowerLinc USB Plugin

9.20 USB-UIRT Actions

- USB-UIRT Action

9.20.1 USB-UIRT Action

USB-UIRT Action | On Error | States

Provides two sub-actions, **Transmit IR Command** and **Control Indicator LED**.

Transmit IR Command

© 2008 Promixis, LLC

IR Code to Transmit: Enter the code to be transmitted either in UIRT or in Pronto CCF format.

Repeat burst: Enter the number of times the code should be repeated.

Carrier: Enter the carrier frequency.

Wait: Wait a given number of milliseconds of IR "quiet" before transmitting.

Don't wait: Tick to transmit without waiting for quiet.

Trigger the following when finished: Press **Browse** to Select a node to run once transmission is complete.

USB-UIRT Configuration: Press to bring up a device configuration dialog.

Learn IR Code: Press to teach the device a new remote button.

Control Indicator LED

Indicator LED: Turn off or on; pulse off or on.

Duration: Enter the duration of the required pulse.

Source: USB-UIRT driver Plugin

9.21 Global Cache Actions

Requires Girder Pro.

These actions allow commands to be sent to a Global Cache device to cause it to output IR signals and serial port data and to switch relays on and off.

- Send IR CCF Action.
- Send IR GC Action.
- Stop IR Action.
- Set Relay Action.

Note that Girder also comes with a Global Cache Component, you can use either but not both at the same time. If you would like to use Girder's IR Profile functionality with the Global Cache you may not use these functions.

9.21.1 Send IR CCF Action

Send IR CCF | On Error | States

Requires Girder Pro.

Sends an IR command defined in CCF format once or repeatedly.

IP Address: The IP address of the Global Cache unit.

CCF: The IR code in CCF format.

Module: The Module Number (see Global Cache Plugin)

© 2008 Promixis, LLC

Address: The IR Channel Number.

Repeat: Number times to send the command or 0 to send until stopped. Some hardware requires a repeat larger than one. If you are having trouble try settings this to 3 or higher.

Source: Tree Script Plugin

9.21.2 Send IR GC Action

Send IR GC | On Error | States

Requires Girder Pro.

Sends an IR command defined in GC format.

IP Address: The IP address of the Global Cache unit.

GC: The IR code in GC format.

Module: The Module Number (see Global Cache Plugin)

Address: The IR Channel Number.

Source: Tree Script Plugin

9.21.3 Stop IR Action

Stop IR | On Error | States

Requires Girder Pro.

Stops sending any IR transmission on the selected channel.

IP Address: The IP address of the Global Cache unit.

Module: The Module Number (see Global Cache Plugin)

Address: The IR Channel Number.

Source: Tree Script Plugin

9.21.4 Send ASCII Action

Send ASCII | On Error | States

Requires Girder Pro.

Sends an ASCII string out on one of the Global Cache serial ports.

IP Address: The IP address of the Global Cache unit.

Text: The ASCII text to send.

© 2008 Promixis, LLC

Line Feed: Tick to send an ASCII line feed control character after the Text.

Carriage Return: Tick to send an ASCII carriage return control character after the Text and Line Feed if selected.

Port: The Port Number (see Global Cache Plugin)

Source: Tree Script Plugin

9.21.5 Set Relay Action

Set Relay | On Error | States

Requires Girder Pro.

Turns the selected Global Cache relay on or off.

IP Address: The IP address of the Global Cache unit.

State: Tick to turn the relay on, un-tick for off.

Module: The Module Number (see Global Cache Plugin)

Address: The Relay Number.

Source: Tree Script Plugin

9.22 NetRemote Actions

The "All" version acts on all connected NetRemote clients while the "Client" version specifies an individual target NetRemote. If you only have one Client, the "All" version is simpler and more portable.

- Set Variable Action (All and Client Versions)
- Set Image Action (All and Client Versions)
- Jump Action (All and Client Versions)

9.22.1 Set Variable Action

Set Variable | On Error | States

Sets a NetRemote variable in the client or clients. The variable can be used as a caption on a Frame or Button to display information.

Variable: The name of the NetRemote variable to be set.

Value: The value to set it to (note that you can embed Girder Lua variables in the value by enclosing their names in square brackets).

Client (Not in **All** version): Pick the name of the NetRemote client to work with.

Source: Tree Script Plugin

© 2008 Promixis, LLC

9.22.2 Set Image Action

Set Image | On Error | States

Sets a NetRemote image variable using an image from a file. The image variable can be displayed in a Frame or Button on NetRemote.

Image: The name of the NetRemote image variable to be set.

Filename: The path and filename of an image file to use as the source.

Client (Not in **All** version): Pick the name of the NetRemote client to work with.

Source: Tree Script Plugin

9.22.3 Jump Action

Jump | On Error | States

Causes the NetRemote client or clients to change page.

Page Group: The name of the NetRemote Page Group (or Device) containing the target page.

Page: The name of the target Page (or Panel).

Client (Not in **All** version): Pick the name of the NetRemote client to work with.

Source: Tree Script Plugin

9.23 Serial Devices Actions

Requires Girder Pro.

- Serial Send Action
- Set Log Level Action

9.23.1 Serial Send Action

Serial Send | On Error | States

Requires Girder Pro.

Sends a command to a device configured with the Generic Serial Plugin.

Serial Device: Select the serial device to send to.

Send: The string to send. This may be ASCII text or hexadecimal or binary strings.

Send As: Select the way the string is formatted.

Source: Generic Serial Plugin.

© 2008 Promixis, LLC

9.23.2 Set Log Level Action

Set Log Level | On Error | States

Requires Girder Pro.

Sets the level of diagnostic logging used for a device configured with the Generic Serial Plugin.

Serial Device: Select the serial device.

Set Log Level: Select the required log level.

Source: Generic Serial Plugin.

9.24 Voice Actions

Requires Girder Pro.

- Voice Speak Action
- Voice Volume Action
- Voice Enable/Disable Action
- Voice Override Times Action

9.24.1 Voice Speak Action

Voice Speak | On Error | States

Requires Girder Pro.

Speaks the text provided. Note that this depends on settings made in the Voice Application.

Speak: The text to be spoken. May include Lua variable names enclosed in square brackets to speak the value of the variable.

Asynchronous: If unticked, subsequent actions will not be executed until the speech has completed.

Test: Press to test the speech.

Source: Tree Script Plugin

9.24.2 Voice Volume Action

Voice Volume | On Error | States

Requires Girder Pro.

Adjusts the audio volume for speech.

Set Voice Volume: 0 (silent) to 100%.

Source: Tree Script Plugin

© 2008 Promixis, LLC

9.24.3 Voice Enable/Disable Action

Voice Enable/Disable | On Error | States

Requires Girder Pro.

Enables or disables speaking.

Enable voice: Tick to enable, un-tick to disable.

Source: Tree Script Plugin

9.24.4 Voice Override Times Action

Voice Override Times | On Error | States

Requires Girder Pro.

Enables or disables the voice override times set in Voice Application.

Override Voice Time settings: Tick to allow voice at any time, un-tick to allow only at times set in the Voice application settings.

Source: Tree Script Plugin

9.25 X10 Actions

Requires Girder Pro.

- Device Command Action
- House/Unit Code Commands Action
- SmartHome Commands Action

9.25.1 Device Command Action

Device Command | On Error | States

Requires Girder Pro.

Sends an X10 command to a named device (see X10 Home Automation).

Device: Select the named X10 device from the drop-down list. The corresponding house and unit code is shown along with the current device status if the device supports this.

Command: Select the command to send from the drop-down list. The available commands depend on the device type.

Restore to previous state in: Tick this and enter the required delay in Hours, Minutes and Seconds.

Source: Tree Script Plugin

© 2008 Promixis, LLC

9.25.2 House/Unit Code Commands Action

House/Unit Code Commands | On Error | States

Requires Girder Pro.

Sends an X10 command. Use Device Command Action unless the device or command is not supported.

House Code: Select the house code part of the address.

Unit Code: Select the unit code part of the address.

Function Code: Select the function code to send.

Repeat Count: Select the number of times to send the command.

Source: Tree Script Plugin

9.25.3 SmartHome Commands Action

SmartHome Commands | On Error | States

Requires Girder Pro.

Sends SmartHome specific commands (see the SmartHome manual).

Select Sequence: Select the command sequence to be sent.

Source: Tree Script Plugin

9.26 Device Manager

The Device Manager is Girder's unified method of controlling a heterogenous collection of devices. This functionality requires Girder Pro.

- Generic action can control any device
- Lighting action controls devices related to lights
- Appliances action controls appliance devices.
- Security Actions controls security systems
- Audio/Video action controls amplifiers, projects and the like.
- HVAC action controls you thermostat and AC units

9.26.1 Generic Action

Generic Commands | On Error | States

Requires Girder Pro.

© 2008 Promixis, LLC

The Generic Action sets the value of a control on a device, this will work for any class of device.

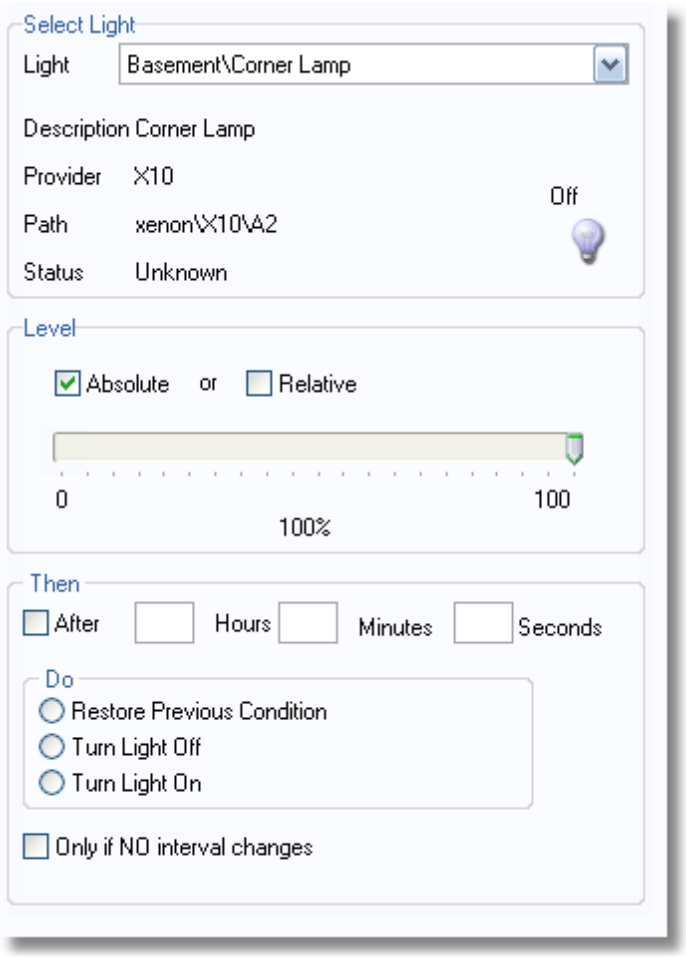
Source: Device Manager

9.26.2 Lighting Action

Lighting Commands | On Error | States

Requires Girder Pro.

© 2008 Promixis, LLC



The Lighting action set the levels of lighting devices. You can do so by setting the level absolute or relative, meaning set a light to a specific value or relative to it's current value. Secondly this action allows you to specify what to do after a specific amount of time. Either go back to the original level or turn the light on or off.

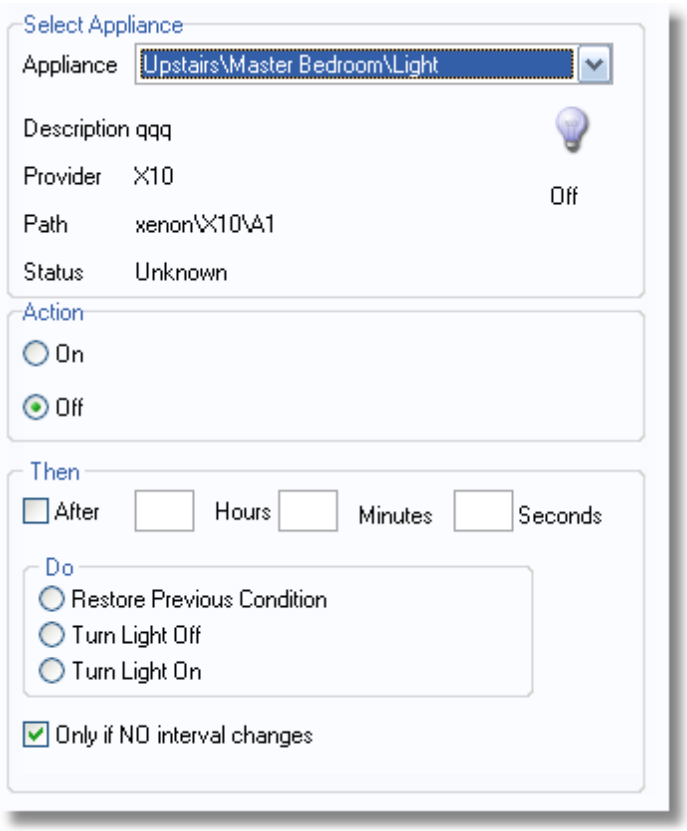
Source: Device Manager

9.26.3 Appliance Action

Appliance Commands | On Error | States

Requires Girder Pro.

© 2008 Promixis, LLC



The Appliance action switches an appliance on or off. Secondly this action allows you to specify what to do after a specific amount of time. Either go back to the original level or turn the light on or off.

Source: Device Manager

9.26.4 Security Actions

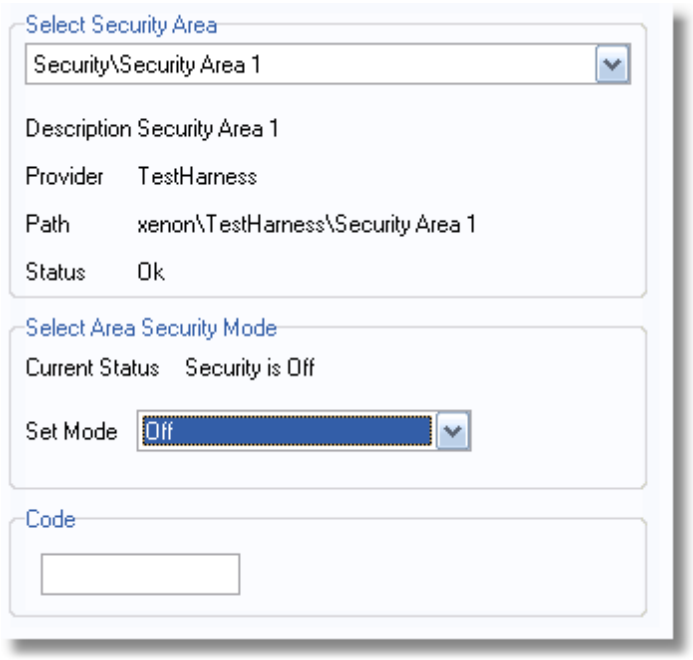
- Security Zone
- Security Area

9.26.4.1 Security Area

Security Commands | On Error | States

Requires Girder Pro.

© 2008 Promixis, LLC



This action allows you to set the security mode of an area.

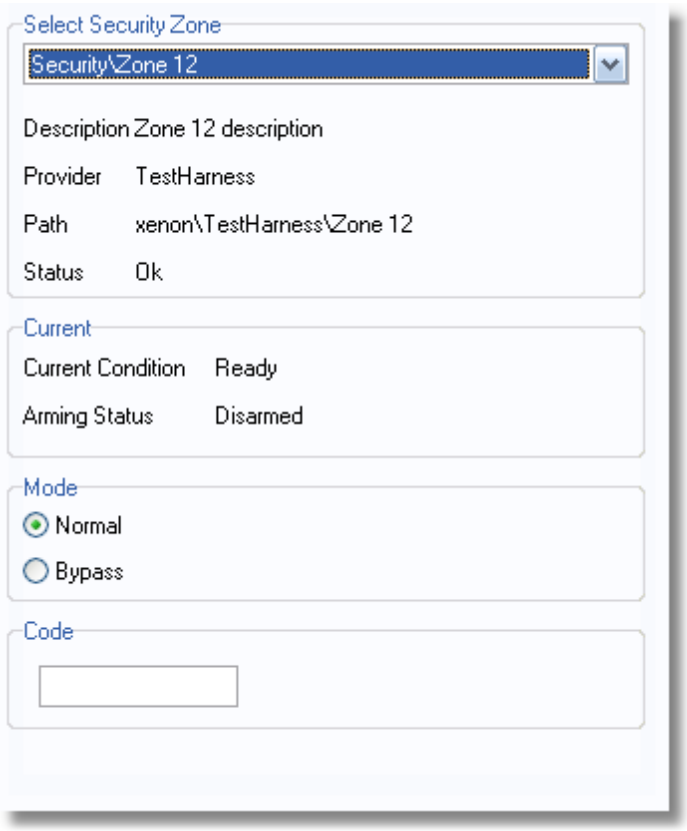
Source: Device Manager

9.26.4.2 Security Zone

Security Commands | On Error | States

Requires Girder Pro.

© 2008 Promixis, LLC



This action allows you to set the security mode of a zone.

Source: Device Manager

9.26.5 Audio/Video Actions

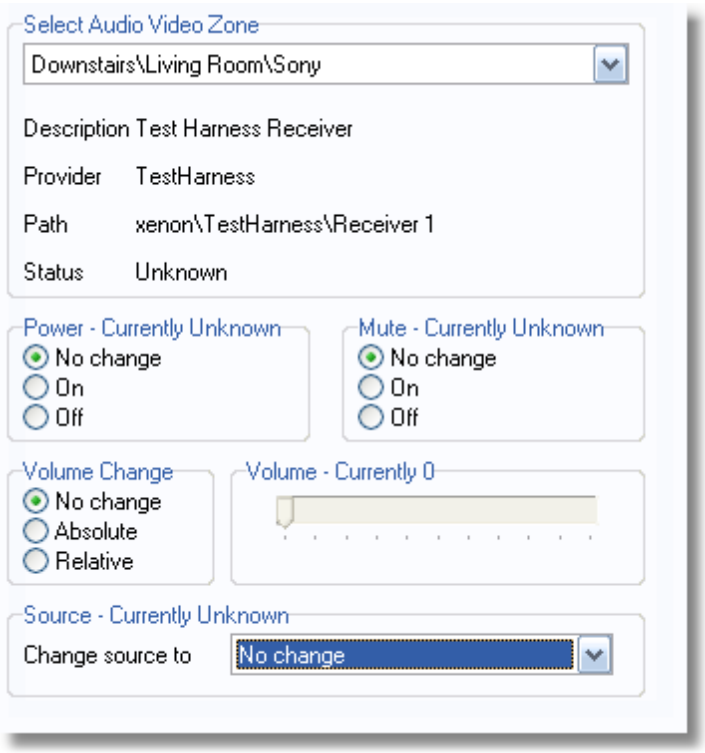
- Receiver
- Balance
- Bass, sets the bass level for an AV source
- Treble, sets the treble level for an AV source
- Transport

9.26.5.1 Receiver Action

Audio / Video Receiver Command | On Error | States

Requires Girder Pro.

© 2008 Promixis, LLC



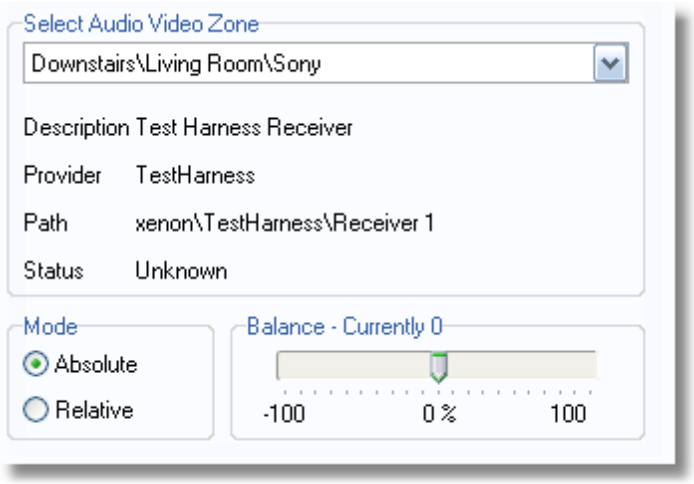
This action allows you to change the various properties of the receiver.

Source: Device Manager

9.26.5.2 Balance Action

Audio / Video Balance Command | On Error | States

Requires Girder Pro.



This action allows you to change the various properties of the receiver.

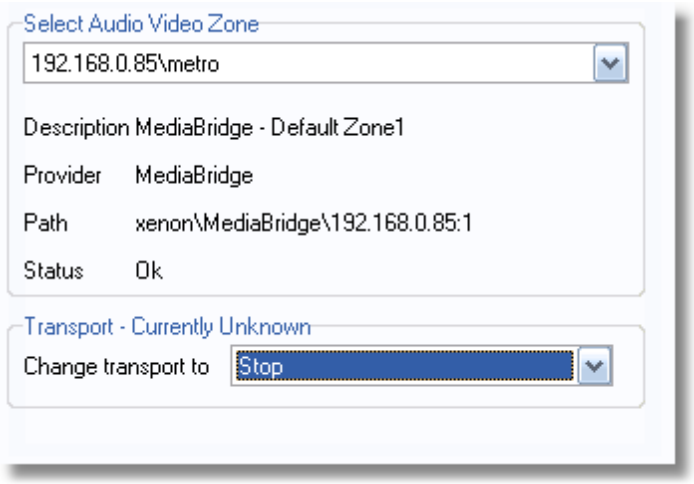
© 2008 Promixis, LLC

Source: Device Manager

9.26.5.3 Transport Action

Audio / Video Transport Command | On Error | States

Requires Girder Pro.



This action allows you to control the transport controls for an audio source. Transport controls are for example; play, pause and stop.

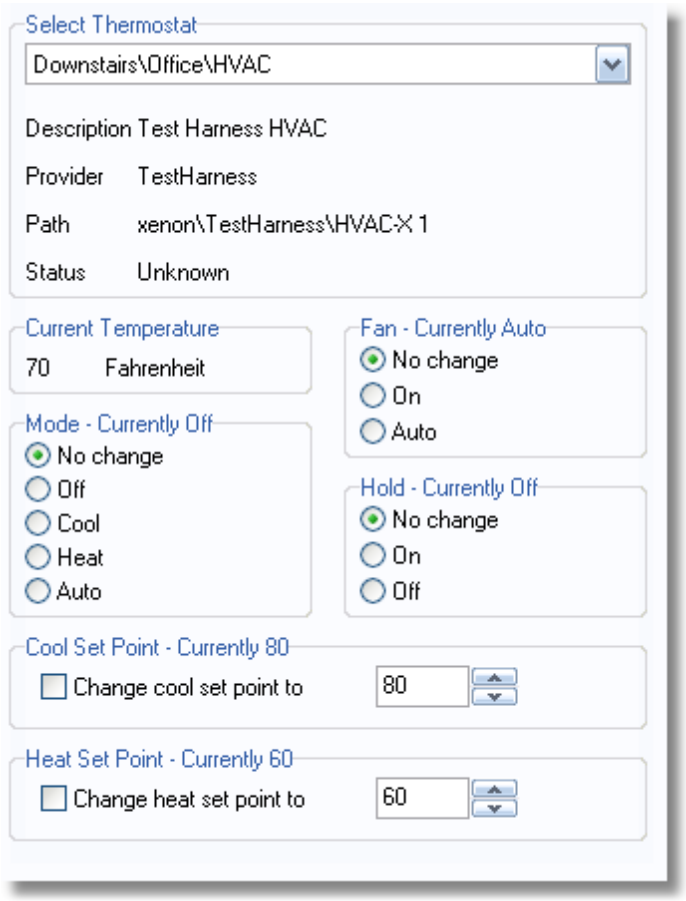
Source: Device Manager

9.26.6 HVAC Action

HVAC Command | On Error | States

Requires Girder Pro.

© 2008 Promixis, LLC

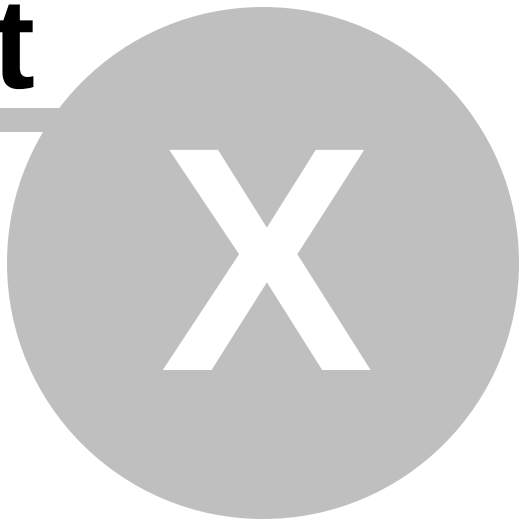


This action allows you to change the various properties of the HVAC system.

Source: Device Manager

© 2008 Promixis, LLC

Part



10 Conditionals Reference

This section describes the Conditionals available in the Plugins shipped with Girder.

The source of each Conditional is identified in the page footer. A Conditional will only be available if the relevant Plugin is enabled.

- Window Conditional.
- Event Filter (Girder Pro).
- Automation Conditionals (Girder Pro).

10.1 Window Conditional

Window Conditional | On Error

Makes a group or macro conditional on the existence or foreground characteristic of the Target Window.

Window Picker: Select the window to test.

Window Exists: Test depends on the existence of otherwise of the target window.

Window Is Foreground: Test depends on whether the target window is the foreground window.

Invert: Tick to reverse the logic of the test (Window does not exist; Window is not foreground).

Usage:

A common use for the Window Conditional is to enable or disable a group containing actions triggered by a remote depending on which application is in the foreground. This enables one remote to control several applications. For example, you might have a media player like **WMP** and a separate DVD player like **PowerDVD** and you want the transport controls on your remote to control whichever is currently in use.

To achieve this, put the Actions which control WMP in one group and those for PowerDVD in another (you can also put each group in its own GML file). Trigger each set of Actions with the same remote events (you can use the events from the **Mapping Device** or the **Raw Events** from the remote). For example, the **PLAY** mapped event would trigger an action to drive the WMP play function in one Group and the PowerDVD play function in the other. At this point, if both WMP and PowerDVD are running, the **PLAY** button will start both playing simultaneously.

To fix this, attach a Window Conditional to each Group. Use the Window Picker to set the **Conditional** on the WMP Group to the WMP application window and the one on the PowerDVD group to the PowerDVD application window. Set both to condition "Window Is Foreground".

Use the Focus Action in the Windows Actions group to change the foreground window. You will need to use a separate remote button for each application and the focus actions will need to be outside the conditional groups discussed above. Use the File Execute Action in the OS Actions group to start an application.

Source: Window Conditionals Plugin

© 2008 Promixis, LLC

10.2 Event Filter

Event Filter | On Error

Requires Girder Pro.

This conditional allows or suppresses Events from specific devices from triggering Actions on or under the node to which it is attached.

Suppress events from devices in this range: All Events from devices on or above the lower limit and on or below the upper limit will be suppressed unless they are also in the "Except" range.

Except events from devices in this range: Events from devices on or above the lower limit and on or below the upper limit will be passed even if they are also in the "Suppress" range.

Source: Tree Script Plugin EventExtras.lua.

10.3 Automation Conditionals

- Ambient Light Conditional.
- Remote Girder Conditional.

10.3.1 Ambient Light Conditional

Ambient Light Conditional | On Error

Requires Girder Pro.

This conditional enables or disables a node and its descendants depending on the level of a light zone. The zone must have been previously defined on the Light Zones tab of the Scheduler Automation Application on the Settings dialog.

Ambient light level: Select the light zone to be tested.

Operator: Select the test (note that "above" means "more light").

Level: The reference level to test against.

Source: Tree Script Plugin AmbientLight UI.lua.

10.3.2 Remote Girder Conditional

Remote Girder Conditional | On Error

Requires Girder Pro.

This conditional enables or disables a node and its descendants depending on the availability or otherwise of a remote Girder node.

Server Name or IP Address of Remote Girder: Select the server name or IP address of the remote Girder to be tested.

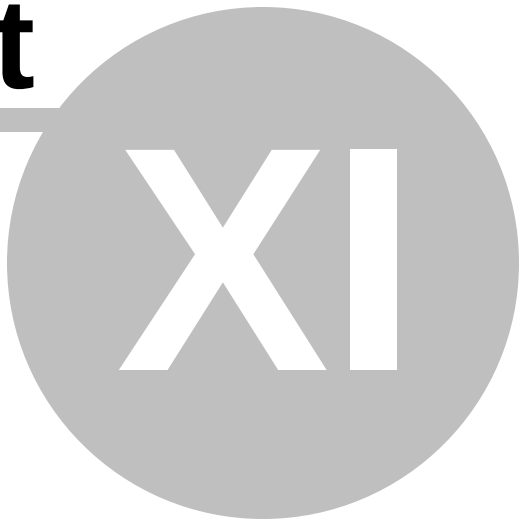
Invert logic: If not ticked, the node is enabled if the remote Girder is online. If ticked, the node is enabled if the remote Girder is offline or unknown.

© 2008 Promixis, LLC

Source: Tree Script Plugin G2G UI.lua.

© 2008 Promixis, LLC

Part



11 Lua Language Reference

11.1 Lua Introduction

Girder uses Lua as its scripting language. Lua is an extension programming language designed to support general procedural programming with data description facilities. Lua is intended to be used as a powerful, light-weight configuration language.

Lua is implemented as a library, written in C. Lua has no notion of a "main" program; it only works embedded in a host client, in this case Girder. This host program controls Lua via an API. The API is also made available to Plugin developers.

Lua can be augmented to cope with a wide range of different domains, creating customized languages sharing a syntactical framework. Girder adds an extensive library of functions to the basic Lua engine and Plugins add yet more. The extensions provided by Girder and its standard Plugins are listed and described in the Lua Library Reference.

Lua is free software, and is provided as usual with no guarantees. Lua is licensed under the terms of the MIT license. The official URL is.

http://www.lua.org/

Up-to-date information about Lua-related resources can be found at the lua-users wiki.

http://lua-users.org/

The Lua language and its implementation have been designed and written by Waldemar Celes, Roberto Ierusalimschy and Luiz Henrique de Figueiredo at Tecgraf, the Computer Graphics Technology Group, Department of Computer Science, of PUC-Rio (the Pontifical Catholic University of Rio de Janeiro) in Brazil.

© 2008 Promixis, LLC

This section of the Girder Manual is adapted from Programming Lua 5: Quick Reference Guide by Kein-Hong Man.

11.2 Girder Script Containers

There are several places in Girder you can put Lua Script.

1. In GML files using the Scripting Action. This code will be compiled to a chunk and executed whenever the Action is triggered by an Event.
2. Files in the **%GIRDER%\luascript\startup*.lua** directory will be executed automatically on startup or script reset.
3. Files in the **luascript** directory or in subdirectories below it can be executed from other scripts using the Lua **require** function.
4. The Tree Script Plugin executes special purpose Lua files in the **%GIRDER%\plugins\treescript** directory to create new Actions, Conditionals and Configuration Pages.
5. The Generic Serial Plugin (Girder Pro) executes special purpose Lua files in the **%GIRDER%\plugins\serial** directory to create device drivers for serial devices.
6. The HID Plugin stores short scripts on its configuration page to parse data from HID devices like joysticks and game pads to generate Girder Events.
7. The Web Server Plugin (Girder Pro) processes Lua scripting embedded in web pages when they are served to external web browsers.

Method 1 is probably the best approach for general scripting, at least at the start. It has the advantages that the script is visible in the Girder user interface and you have the services of a good syntax highlighting Lua editor.

If you are writing reusable code (functions intended to be used from other scripts) they will need to be run every time the scripting system is initialized. This can be achieved as follows.

1. Create a new Macro node and rename it "OnScriptEnable" (or anything meaningful).
2. Insert an Event node on the Macro and set **Event Device** = "Girder Events", **Event String** = "ScriptEnable".
3. Insert any number of Scripting Action nodes on the Macro as containers for the reusable scripts.

11.3 Lexical Conventions

Reserved Words and Other Tokens

Lua is *case-sensitive*. *Identifiers* in Lua can be any string of letters, digits, and underscores, not beginning with a digit. Any character considered alphabetic by the current locale can be used in an identifier. The following *keywords* are reserved.

```
and       break     do        else      elseif
end       false     for       function  if
in        local     nil       not       or
repeat    return    then      true      until
while
```

The following strings denote other tokens.

```
+   -    *    /    ^    =    ~=   <=   >=
<   >    ==   (    )    {    }    [    ]
;   :    ,    .    ..   ...
```

By convention, identifiers starting with an underscore followed by uppercase letters (such as **_VERSION**) are reserved for internal variables used by Lua.

Number constants

Numerical constants may have an optional fractional part and an optional decimal exponent. The exponent is delimited from the mantissa with an upper or lower case "E" and may be signed.

```
n = 3; n = 3.0; n = 0.3e1; n = 30.0E-1; --All these assign the same value.
```

© 2008 Promixis, LLC

String Literals

Literal strings can be delimited by matching single or double quotes, and can contain any 8-bit value, including embedded zeros (represented by **\0**). The following C-like escape sequences represent specific bytes.

```
\ddd    1 to 3 digits decimal value of byte.
\a    bell \007
\b    backspace \008
\f    form feed \012
\n    newline \010
\r    carriage return \013
\t    horizontal tab \009
\v    vertical tab \011
\
            Backslash followed by an actual new line inserts \n.
The following pairs insert the second character literally.
\\    \"    \'    \[    \]
```

Literal strings can also be delimited by matching "[[...]]". In strings delimited this way, escape sequences are not interpreted and the string is entered exactly as is including any embedded newlines. However, when the opening "[[" is immediately followed by a newline, that newline is ignored.

The following examples assign the same string literal.

```
s = "Girder users \"reach\nfor the moon\" with Lua"
s = 'Girder users "reach\nfor the moon" with Lua'
s = [[
Girder users "reach
for the moon" with Lua]]
```

Comments

The first line of a chunk is skipped if it starts with "#" (for Unix scripting).

A *short comment* starts with a double hyphen "--" and runs until the end of the line.

A *long comment* starts with "--[[" and ends with a balanced "]]". It may be multiline and may contain nested "[[...]]" pairs.

```
--[[
This is a
multi-line comment.
]]
-- This is a single line comment.
print("Some code") -- Rest of this line is a comment
```

Block comments can be used to enable and disable code during testing.

```
--[[
print("This is not compiled")
--]]
---[[
print("This is compiled")
--]]
```

Adding a single hyphen converts the opening delimiter into a single-line comment which exposes the hyphens in front of the closing delimiter making this line also a single-line comment.

11.4 Values and Types

Lua is *dynamically typed*. Values carry their own type and Lua does not have type definitions.

Types

© 2008 Promixis, LLC

There are eight basic types in Lua.

nil: Type of **nil**, which is different from any other value.

boolean: Type of the values **false** and **true**. Both **nil** and **false** make a condition false, any other value makes it true.

number: Double-precision floating-point numbers. (In Girder, other Lua implementations may define the number type differently.)

string: Arrays of characters. May contain any 8-bit character, including embedded nulls.

table: Implements associative arrays. Tables can be indexed with any value (except **nil**). Tables can be heterogeneous containing values of all types (except **nil**). They are the sole data structuring mechanism in Lua and may be used to represent ordinary arrays, symbol tables, sets, records, graphs, trees, etc.

function: Functions are *first-class* values in Lua. They can be stored in variables, passed as arguments, and returned as results. Executing a function definition assigns the compiled function to a variable.

userdata: This type is provided to allow arbitrary C data to be stored in Lua variables. Corresponds to a block of raw memory and has no pre-defined operations except assignment and identity test.

thread: Represents independent threads of execution for Coroutines.

More about Types

The **type** function returns a string describing the type of a given value.

Operations for userdata values (and for tables) can be defined using *metatables*. Userdata values cannot be created or modified in Lua, only through the C API. Functions can also be defined in C using the API.

To represent records, the field name is used as an index. **a.name** is equivalent to **a["name"]**. The value of a table field can be of any type (except **nil**). Table fields may contain functions, and carry *methods*.

Tables, functions, and userdata values are *objects*, variables contain only references to them. Assignment, parameter passing, and function returns always manipulate references to such values and do not imply any kind of copy.

Coercion

At run time, a string is converted to a number if it is used in an arithmetic operation, and vice versa. A reasonable format preserving the *exact* value of the number is used. (Use **string.format** for printing numbers instead.)

11.5 Variables

There are three kinds of variables in Lua, *global variables*, *local variables*, and *table fields*. Conceptually all possible global variables and table fields exist and are initialized with the value **nil**. Thus accessing any variable will succeed, if only by giving the value **nil**. Variables can also be assigned back to **nil** when no longer required (this does release memory).

Variables are global unless declared local or accessed using table syntax.

Local variables

© 2008 Promixis, LLC

A variable is declared local as follows.
```
local a, b
```
When a local variable is non-**nil** it hides the global variable of the same name.

Table fields

Square brackets or field syntax are used to access a table as follows.
```
table["index"] = "value"
table.index = "value"
```

The meaning of accesses to global variables and table fields can be changed via *metatables*. For example, an access to an indexed variable **t[i]** is equivalent to a call getmetatable(t). __index(t,i).

Global environments

All global variables live as fields in Lua tables, called *environment tables* or simply *environments*. Lua functions and functions defined in C all initially share the same environment table, but each Lua function carries its own reference to an environment which may be changed. A Lua function inherits the environment from the function that created it. The environment table of a Lua function is accessed using **setfenv** and **getfenv**.

11.6 Statements and blocks

Chunks

The unit of execution of Lua is called a *chunk*. A chunk is simply a sequence of statements which are executed sequentially. Statements are delimited by syntax and white space, but can optionally be terminated with a semicolon. Newlines count as whitespace and statements may be split over more than one line. Each statement can be optionally followed by a semicolon.

Lua handles a chunk as the body of an anonymous function. Chunks can define local variables and return values. A chunk may be stored in a file or in a string. Precompiled binary chunks (using **luac**) can be used interchangeably with chunks in source form. Detection is automatic.

Blocks

A block is a list of statements. Syntactically, a block is equal to a chunk. A block may be explicitly delimited within a chunk or a block to produce a local scope.

do BLOCK **end**

Explicit blocks are useful to control the scope of variable declarations, or to add a **return** or **break** statement in the middle of another block.

Function definitions and some control structures produce implicit blocks.

Visibility

Lua is lexically scoped allowing blocks to be defined within other blocks. The scope of local variables begins at the first statement *after* their declaration and lasts until the end of the innermost block that includes the declaration.

Local variables can be freely accessed in blocks defined inside their scope. This includes functions, which may be defined inside blocks including inside other function definitions. A local variable used by an inner function is called an *upvalue*, or *external local variable*, inside the inner function. A variable of the same name in an inner scope has precedence. Each instance of an anonymous function (or closure) defines new instances of local variables.

© 2008 Promixis, LLC

Assignment statements

Lua allows multiple assignment. The syntax for assignment defines a list of variables on the left side and a list of expressions on the right side.

```
a, b, c = 1, "JH", 2.1
```

Before the assignment, the list of values is *adjusted* to the length of the list of variables. Excess values are thrown away. If there is a shortage, the list is extended with as many **nil** values as needed. Lua first evaluates all expressions, and only then are the assignments made. Thus the following is an *exchange*.

```
x, y = y, x
```

The meaning of assignments to global variables and table fields can be changed via metatables.

Local variable declarations may include an initial assignment.

```
local a, b = 1, "JH"
```

Because a chunk is also a block, local variables can be declared outside any explicit block. Such local variables die when the chunk ends.

11.7 Control Structures

Control structures in Lua have the usual meaning and familiar syntax.

While, repeat and if structures

```
while EXP do BLOCK end
repeat BLOCK until EXP
if EXP then BLOCK      -- One of these
elseif EXP then BLOCK  -- Zero or more of these
else BLOCK                    -- Zero or one of these
end                           -- And one of these
```

The condition expression EXP of a control structure may return any value. Both **false** and **nil** are considered false. Other values are considered true, including the number 0 and the empty string.

For structures

The **for** statement has numeric and generic forms.

```
for c = start, end , step do BLOCK end
for ix, ... in explist do BLOCK end
```

In the numeric form, the step is optional and defaults to 1. All control expressions are evaluated once before the loop starts and must result in numbers. The counter is scoped within the block and the behavior is *undefined* if you assign to it.

In the generic form, explist is evaluated once, giving an *iterator* function, a *state*, and an initial index value. The iterator function is called repeatedly with the state and current index value. It returns the next index plus the specified number of additional values. Behavior is undefined if the index is reassigned within the loop.

Functions pairs and ipairs are provided in the basic library to iterate tables in generic for structures. Pairs produces key, value pairs from any table in undefined order. ipairs produces index, value pairs from numerically indexed tables (lists) in numeric order.

```
for k, v in pairs(table) do print(k, v) end
```

Exiting Loops

return is used to return values from a function or from a chunk, even within a loop. **break** can be used to terminate the execution of the innermost **while**, **repeat**, or **for** loop, skipping to the next statement after the loop.

return and **break** statements can only be written as the last statement of a block. An explicit

© 2008 Promixis, LLC

inner block can used to work round this restriction by writing **do return end** or **do break end**.

11.8 Expressions and Operators

The Lua operator list and precedence, from the lower to the higher priority.

Assoc	Operators	Description
left	**or**	Logical OR
left	**and**	Logical AND
left	**<**, **>**, **<=**, **>=**, **~=**, **==**	Relational operators
right	**..**	Concatenation
left	**+**, **-**	Arithmetic addition, subtraction
left	*****, **/**	Arithmetic multiplication, division
right	**not**, **-**(unary)	Logical NOT, unary minus
right	**^**	Exponentiation (**__pow** or metamethod)

An expression enclosed in parentheses is always evaluated before being used in an outer expression and always results in exactly one value (the first value returned or **nil** if no value is returned).

Relational Operators

Relational operators always result in **false** or **true**.

The equality operator (**==**) first compares the types of its operands. If types are different, the result is **false**. If types are the same, values are compared. Numbers and strings are compared in the usual way. Coercion is not applied to equality comparisons so **"0" == 0** evaluates **false**. Objects (tables, userdata, threads, and functions) are compared by *reference* unless a **__eq** metamethod is available.

The operator **~=** is exactly the inverse of **==**.

Order operators (**<**, **>**, **<=**, **>=**) compare pairs of numbers or pairs of strings (using the current locale) or uses the **__lt** or the **__le** metamethods.

Logical Operators

Logical operators consider both **false** and **nil** as false and anything else as true. **not** always returns **false** or **true**.

and returns its first argument if this value is **false** or **nil**, otherwise it returns its second argument. **or** returns its first argument if this value is different from **nil** and **false**, otherwise it returns its second argument. Both operators use short-cut evaluation.

The following are useful Lua idioms that use logical operators (where *b* should not be **nil** or **false**).

```
x or error()     --means: if not(x) then error() end
x = x or v       --means: if not(x) then x = v end
x = a and b or c --means: if a then x = b else x = c end
```

11.9 Table Constructors

Table constructors are expressions that create tables. Every time a constructor is evaluated, a new table is created. Constructors can be used to create empty tables, or to create a table and initialize some of its fields.

© 2008 Promixis, LLC

```
t = {}                          -- Empty table
t = {[1.1]="one", [2.1]="two"} -- Table with keys of any type
t = {1, 3, 5, 7}               -- Table with auto assigned number keys
t = {field1="one", field2="2"} -- Table with string keys
```

Initializer lists can use semi-colons in place of commas (freely mixing the two) and can have a comma or semicolon after the last element.

Each field of the form **[KEYEXP] = VALUEEXP** adds an entry to the table.

The form *name* = **VALUEEXP** is equivalent to **["***name***"] = VALUEEXP**.

Where keys are not specified, entries are assigned consecutive numerical integers, starting with 1. Fields in the other formats can be interspersed and do not affect this counting.

If the last field in the list is a function call without an explicit key, then all values returned by the call enter the list consecutively. To avoid this, enclose the function call in parentheses.

A value field can be a nested table constructor generating an inner table as a field of the outer table.

11.10 Function Calls and Definitions

Function Calls

A function call in Lua has one of the following forms.

```
r1, r2 ... = MyFunction(p1, p2 ...)  -- Normal form
table:MyMethod(p1, p2 ...)           -- Method call
MyFunction "string"                 -- Sortcut for single string parameter
MyFunction {"l1", "l2"}             -- Parameters from table constructor
```

All forms may have return lists. The method call form is equivalent to **table.MyMethod(table, p1, p2 ...)**. All argument expressions are evaluated before the call. A function can return any number of results. If the function is called as a statement, all returned values are discarded, otherwise the return list is adjusted according to the number of receiving assignments.

If called inside another expression or in the middle of a list of expressions, then its return list is adjusted to one element (the first one). If the function is called as the last element of a list of expressions, then no adjustment is made (unless enclosed in parentheses).

If target of a function call is a userdata or table, the call may still be possible if there is a metatable. If there is a __call metamethod, the table or userdata itself is passed, followed by the calling parameters.

A line break cannot be put before the '(' in a function call, to avoid some ambiguities. A semicolon can be added to disambiguate breaks.

Lua implements *proper tail calls* (or *proper tail recursion*). A tail call erases any debug information about the calling function, and can only happen with a function call in a return statement.

Function Definitions

A function definition is an executable expression, whose value has type *function*. Equivalent function definition forms.

```
function f() ... end
f = function() ... end
function a.b.f() ... end
a.b.f = function() ... end
local function f() ... end
local f; f = function() ... end
function a.b:f( ... ) ... end
```

© 2008 Promixis, LLC

`a.b.f = function(self, ... ) ... end`
When Lua pre-compiles a chunk, all its function bodies are pre-compiled too. Whenever Lua executes the function definition, the function is *instantiated* or *closed*. This *instance*, or *closure* is the final value of the expression. Different instances of the same function may refer to different external local variables and different environment tables.

An adjustment is made to the argument list if required. Parameters act as local variables that are initialized with the argument values. Results are returned using the **return** statement.

If the function is a *vararg function* (denoted by "..." at the end of the argument list) it collects all extra arguments into an implicit table parameter, called **arg**, with a field **arg.n** whose value holds the number of extra arguments. The extra arguments are found at positions 1, 2, …, *n*.

For example, if there are no extra arguments, **arg.n** = 0. If the extra arguments are 4 and 2, then **arg** is **{4, 2; n=2}**.

11.11 Metatables

Table and userdata objects in Lua may have a *metatable* that defines their behavior for certain operations. An objects behavior can be changed for some operations by setting specific fields in its metatable.

Keys in a metatable are called *events* and the values (functions), *metamethods*. Query metatables with **getmetatable** and change them with **setmetatable**.

Behavior

When Lua performs a metamethod-associated operation, it checks whether that object has a metatable with the corresponding event. If so, the function associated with that key is used to perform the operation.

The key for each operation is a string with its name prefixed by two underscores, for instance, the key for operation **add** is the string **__add**.

Metatable Keys

add (+); sub (-); mul (*); div (/): If both operands are number type or numeric strings, the normal arithmeric operation is carried out. Otherwise a metamethod is looked for in the metatable of the first operand and failing that, the second operand. If this fails a runtime error is raised. The metamethod takes two operands and returns a result.

pow (^): If both operands are number type or numeric strings, a global function **__pow** is looked for. Otherwise a metamethod is looked for in the metatable of the first and failing that, the second operand. If a suitable function is not found, a runtime error is raised. The metamethod takes two operands and returns a result.

unm (- as a prefix): If the operand is number type or a numeric string, it is negated. Otherwise a metamethod is sought in the metatable of the operand. If this fails a runtime error is raised. The metamethod takes two operands the second of which receives **nil**, and returns a result.

concat (..): If both operands are either string or number type they are converted and concatenated to a string. Otherwise, a metamethod is looked for in the metatable of the first and failing that, the second operand. If a suitable function is not found, a runtime error is raised. The metamethod takes two operands and returns a result.

eq (==): If the operands are of different types, the test is **false**. If the operands refer to the same object, or if they are identical numbers or strings, the test is **true**. Otherwise if both operands have the same metamethod **__eq** this is called for the test result. Otherwise the test is **false**. The metamethod takes two operands and returns a boolean.

© 2008 Promixis, LLC

lt (<): If both the operands are number type the result is that of a standard numeric comparison. If both the operands are strings the result is that of a standard lexical comparison. If the operands are of different types the result is **false**. Otherwise if both operands have the same metamethod **__eq** this is called for the test result. Otherwise the test is **false**. The metamethod takes two operands and returns a boolean.

le (<=): If both the operands are number type the result is that of a standard numeric comparison. If both the operands are string type the result is that of a standard lexical comparison. If the operands are of different types the result is **false**. Otherwise if both operands have the same metamethod **__eq** this is called for the test result. Otherwise if both operands have the same metamethod **__lt** this is called with the operands reversed and the result is inverted. Otherwise the test is **false**. The metamethod takes two operands and returns a boolean.

(a ~= b) is processed as **(not(a == b))**; **(a > b)** is processed as **(b < a)**; **(a >= b)** is processed as **(b <= a)**.

index (object[key]): If the object is a table v = rawget(object, key), if this is not **nil**, it is the value. Otherwise seek the **__index** metamethod of object. If this is not found, the value is **nil**. If the metamethod is a function the value is the result of executing it. If the metamethod is not a function, the index operation is applied to the metamethod object. The metamethod, if a function, takes an object of the type it is attached to and a key of any type and returns a value of any type. The metamethod may also be a table or an object which itself has an index metamethod.

newindex (object[key] = value): If the object is a table, the key is looked up in it. If found, the value is changed. Otherwise the **__newindex** metamethod is sought in a metatable of object. If this is not found, an attempt is made using rawset(object, key, value). If a metamethod is found it is called. Else an error is raised. The metamethod takes an object of the type it is attached to, a key of any type and a value of any type. It returns nothing.

call (execute as a function): If the object is a function it is executed as normal. Else a metamethod **__call** is looked for executed if found. Else an error is raised. The metamethod has a first argument which is an object of the type it is attached to. It has an arbitrary number of additional arguments which are the arguments passed in the call. It has an arbitrary number of return parameters which are the returns of the call.

tostring (tostring(object)): If the object is a string, the operation returns it (has no effect). If the object is a number, it is converted to a string and returned. Otherwise a metamethod **__tostring** is searched for and if found is called to convert the object to a string. The metamethod takes one parameter of the same type as the object it is attached to and returns a string.

gc (finalizer): A metamethod **__gc** takes one argument of the type to which it is connected. It is executed when Lua garbage collects the object. Lua ensures that, in any GC operation, finalizers are called in reverse order of object creation.

mode (weak tables): A string metatable entry keyed **__mode** controls how references held in this table effect the garbage collection of the objects to which they refer. If the string contains a "k", the keys are weak references. If the string contains a "v", the values are weak references. Weak references do not prevent garbage collection of the object to which they refer.

metatable (metatable lock): Set the **__metatable** key in the metatable to prevent access to it. **setmetatable** will raise an error if this key is present. **getmetatable** will return the value of this key, not the actual metatable. For example, set **__metatable** to an error message string.

© 2008 Promixis, LLC

11.12 Garbage Collection

Lua runs a *garbage collector* (GC) from time to time to collect all *dead objects*. All objects in Lua are subject to automatic management.

Lua uses two control numbers, a byte counter which counts the amount of dynamic memory in use and a threshold. When the number of bytes crosses the threshold, Lua runs the GC. The byte counter is adjusted, and then the threshold is reset to twice the new value of the byte counter.

Garbage-Collection Metamethods

You can set GC metamethods for userdata (*finalizers*), to coordinate Lua's GC with external resource management. Free userdata with a field **__gc** in their metatables are not collected immediately.

At the end of each GC cycle, finalizers for userdata are called in *reverse* order of their creation, among those collected in that cycle. (First finalizer called was the last one created.)

Weak Tables

A *weak table* is a table whose elements are *weak references*. If the only references to an object are weak references, then the GC will collect that object. A weak table can have weak keys, weak values, or both. If either the key or the value is collected, the whole pair is removed.

The weakness of a table is controlled by **__mode** in its metatable. If the field is a string containing character **k**, keys are weak. **v** denotes weak values.

A table used as a metatable should not have its **__mode** changed, otherwise the weak behavior of the tables controlled by this metatable is undefined.

11.13 Coroutines

Coroutines represent independent threads of execution. A coroutine suspends execution by explicitly yielding (collaborative multithreading). This is in contrast to the full preemptive multitasking discussed in the next section.

- Coroutine threads are created by calling **coroutine.create**, passing the coroutine function (it is not executed at this stage). A handle (object type *thread*) is returned.
- Executed the coroutine thread by calling **coroutine.resume**, passing the handle and arguments.
- The Coroutine executes until it terminates (via a normal return or an error) or *yields* by calling **coroutine.yield** plus optional arguments.
- **coroutine.resume** normally returns **true**, plus any values returned by the coroutine, or **false** plus an error message.
- When execution resumes, **coroutine.yield** returns the extra arguments that were passed to **coroutine.resume**.
- **coroutine.wrap** creates an alternative coroutine form which does a coroutine.create, but returns a function instead of a thread. This is just like coroutine.resume except that it does not need the thread handle.

11.14 Multitasking with Threads

Normally Girder executes your scripts one at a time in a sequence. Scripts placed in the startup directory execute one at a time in undefined order. Scripts placed in Scripting Actions execute when the event happens and Girder will not execute any other actions until your script completes its work.

There are two circumstances where this execution model is not satisfactory and we need *multitasking* -

© 2008 Promixis, LLC

1. If an action takes a long time to complete (anything over a few seconds). Typical examples are downloading files from the Internet or backing up your computer.
2. If you want to watch for something to happen and generate a Girder Event when it does. It may happen almost immediately or it may be hours or days before it happens.

The Lua language implements a limited form of multitasking with its **coroutine** mechanism. This is *cooperative multitasking* which depends on coroutines yielding control when they are not busy. Girder adds full *preemptive multitasking* to Lua which is more powerful, but also introduces issues that require caution on the part of the programmer (this means you!).

Unless your computer has more than one processor or supports hyper-threading, it can only do one thing at a time. Windows gives the illusion that many programs are running simultaneously by rapidly switching between them. These units of multitasking are called **processes** and usually there is one process per program. Processes have their own memory spaces and cannot interfere with each other except by implementing specific interfaces such as COM, cut-and-paste or keyboard and message spoofing.

Threads are like processes in that Windows rapidly switches between them giving the illusion of simultaneous execution. However threads within the same process all share access to the same memory. Note that the thread library threads discussed in this section are quite different from the thread type built into the Lua language and used by coroutines. Thread library threads are of **userdata** type and have their own methods.

Threads implemented in Lua share access to global variables and library functions. This fact allows threads to easily communicate with each other, but it also creates dangers. Suppose one thread begins to update a variable and then Windows suspends it and starts executing another thread. This second thread might then try to read the half-updated variable or, worse, to update it. This can result in a horrible mess. Fortunately, Girder provides **Mutex** objects for synchronizing access to variables and to groups of variables which are interdependent.

Girder runs all script on the same thread (the main thread) except for callback functions from long-lived objects like **Timers** and **DirectoryWatchers**. The other exception is functions explicitly run by **thread.newthread**. Functions designed to be run on additional threads require some special precautions.

1. The function must terminate before Girder can exit. (If it does not, you'll have to kill Girder using Task Manager.) If your function runs a continuous loop, make it exit if **gir.IsGirderExiting()** returns true. If it blocks on a **Cond** object, make sure this gets signalled when Girder is shutting down.
2. If more than one thread can access some global variables, protect this access in all threads using a **Mutex** object. The easiest way of doing this is to put the related variables in a table with the Mutex object.
3. If the function uses **LuaCOM** you must call **gir.CoInitialize** before using LuaCOM and **gir.CoUninitialize** before the function terminates. All LuaCOM method calls must be made on the same thread on which the COM object is created.
4. Take care using library functions from threads - they may not be "thread-safe". You can safely use **gir.TriggerEvent** and **gir.TriggerEventEx**. If in doubt, generate and event and then run the dubious code in a scripting action triggered by it. Or use **gir.RunCodeOnMainThread**.
5. Do not be a performance hog. If your thread checks some condition in a loop, include a **win.Sleep** to reduce the frequency of the checks.

There is more specific information about threading in the thread Library Reference. Remember that threading is powerful, but it can introduce subtle bugs if not done carefully.

© 2008 Promixis, LLC

Part

12 Lua Library Reference

This section provides information about the Lua standard libraries as well as the libraries added by Girder itself and by the Plugins shipped with Girder as standard.

The source of each feature is identified in the page footer. When this is a Plugin, the feature will only be available if that Plugin is enabled.

- Notation.
- Global Functions comprise the Lua Basic Function Library and some extensions relating to the Interactive Lua Scripting Console.
- cm11 Library of X10 Home Automation functions.
- comserv Library for networking Girder machines.
- coroutine Library from Lua.
- date Library for date and time calculations.
- debug Library from Lua.
- gip Library for interfacing Internet Protocol devices.
- gir Library of Girder functions.
- globalcache Library (Girder Pro) for interfacing Global Cache devices.
- Girder2Girder Library (Girder Pro) for communicating between Girder machines.
- io Library from Lua.
- keyboard Library for disabling and enabling keyboard keys.
- luacom, luacomE Library of COM Automation functions.
- lxp Library (XML Parser).
- math Library from Lua plus Girder extensions.
- mixer Library of audio control functions.
- monitor Library of multi-monitor video display functions.
- mutil Library of miscellaneous CD and DVD library functions.
- NetRemote Library for interfacing NetRemote.
- os Library from Lua.
- osd Library of On Screen Display functions.
- ptable Library a persistent table using the Windows Registry.
- scheduler Library (Girder Pro).
- serial Library (Girder Pro) for interfacing serial port devices.
- socket Library for low-level network access.
- SQL Database Library for database access.
- string Library from Lua plus Girder extensions.
- table Library from Lua plus Girder extensions.
- thread Library of multitasking functions.

© 2008 Promixis, LLC

- transport Library for data communication over serial, HID or TCP/IP.
- usbuirt Library for controlling USB-UIRT devices.
- Voice Library (Girder Pro) providing voice synthesis functions.
- win Library of low-level Windows API functions.
- xap Library of xAP Open Automation Interface functions.
- zip Library (Girder Pro) of file compression functions.

12.1 Notation

Functions are documented as follows. Names in square brackets are *formal parameters* which you replace by your own variable, a literal value (for inputs) or an expression. Output parameters are always optional - you are free to ignore all or some of them. Input parameters are sometimes optional and where this is the case it will be noted in the description. The formal parameters are listed below the syntax box with type information and a description.

```
[output1], [output2] = library.function([input1], [input2])
```
input1: Type. Description.

input2: Type. Description.

output1: Type. Description.

output2: Type. Description.

Examples of actual code using this function:
```
local v1, v2 = library.function(v3, v4)
v1 = library.function(1.2, "Parsecs")
library.function(1.9 * v3, "Par".."secs")
```

Unless otherwise noted, the first output parameter will be **nil** if an error occurs. There are two standard formal parameters which will be shown in the syntax if present but will not be further described.

res: Nil on error. This is used if there is no output parameter required except for an error indicator.

err: Number or String. This is used as an additional output parameter to carry further error information.

Formal parameter conventions:
```
[lowercase] -- an ordinary parameter.
[Capitalized] -- an object parameter (table type following object conventions)
[UPPERCASE] -- a number or string for which a set of constants are provided.
```
Some functions take a function as an input parameter - this is termed a *callback*. In these cases the callback function syntax is shown first, then the function which uses it. The function name is a formal name which can be replaced by anything you like.
```
[output] = [callback]([input1], [input2])
library.function([input3], [callback])
```
Usage example:
```
function mycallback(in1, in2) print(in1, in2); return 0; end
library.function("Hello", mycallback)
```
Objects are shown with the factory function first and then the methods of the object using colon syntax with a formal parameter holding the object reference.
```
[MyObj] = library.factory([input1])
[MyObj]:Method1([input2])
[ret] = [MyObj]:Method2()
```
Usage example:

© 2008 Promixis, LLC

```
local obj = library.factory("Hello")
obj:Method1("World")
print(obj:Method2())
```

12.2 Global Functions

Print, clear, error, assert

print displays a message on the Lua Interactive Scripting Console. **clear** removes all messages from the console (a Girder extension). **error** terminates execution and displays a message on the console. **assert** raises an error if a test value is nil or false.

```
print([message] ...)
clear()
error([message], [level])
[val] = assert([val], [message])
```

message: (string or number) Error message or value to be printed. May also be any object with a **__tostring** metamethod.

level: (number, optional) 1, the default, sets the error position to where error was called. 2 gives the calling point in the parent function etc.

val: (any type) If nil or false, an error is raised. Otherwise val is returned.

Type, tonumber, tostring, unpack

type returns a string indicating the type of a value. **tonumber** converts a string to a number with optional base conversion. **tostring** converts a value to a string. **unpack** returns the elements of a table with numeric keys as individual values.

```
[typename] = type ([value])
[number] = tonumber([value], [base])
[string] = tostring([value])
[...] = unpack([list])
```

value: The value to be operated on.

typename: (string) "nil", "number", "string", "boolean", "table", "function", "thread", or "userdata".

number: (number) The result of tonumber.

base: (number, optional) The default is base 10 and will convert any valid number string, including fractions and exponents, into a number type. For base ~= 10 the value must be a string using A-Z to represent digits 10 to 35.

string: (string) The string form of the value. Number inputs are converted automatically, userdata and table values can have a **__tostring** metamethod to do the conversion.

list: (table) Must be keyed with contiguous integers from 1 up.

Next, pairs, ipairs

next produces all the elements in a table in no particular order. **pairs** is an iterator for use in generic for which produces the elements of a table. **ipairs** is an iterator for use in generic for which produces the elements of a table in numeric order.

```
[key], [value] = next([table], [prev])
for [key], [value] in pairs([table])
for [key], [value] in ipairs([table])
```

table: (table) the table to be iterated.

prev: The key of the previous element or nil to produce the first element.

© 2008 Promixis, LLC

index: The key of the current element.

value: The value of the current element.

Require

Loads a C or Lua package. Packages add tables of functions and constants to the Lua Global Environment. **require** should be called in any script which requires the package, and it will only load the package if it has not already been loaded.
`[ret] = require([packagename])`
packagename: String. The name of the package.

ret: Table. The package table or nil if the package is unavailable.

A package written in Lua should create its function table, export it to the global environment and also return it. It should return false if it cannot install for any reason.

A package written in C should export an initialization function named **luaopen_*name*** (any periods in the name should be removed). This function will be run after the package is opened. It should use the Lua C API to create and export the library table.

require first checks the path **package.cpath** and if the file is found, loads it as a C package. If it is not found on the C path, **package.path** is checked and if the file is found it is loaded as a Lua package. In Girder, the cpath is defined as the Girder installation directory and the path is subdirectory **luascript** below this. The path strings are defined as actual absolute file paths including the file extension and with question mark used as a token which will be replaced with the package name. Multiple paths may be specified, separated by semi-colon characters.

The package name may comprise multiple parts separated by periods. The periods are replaced by path separators before the name is substituted in the path. For example the path "C:\Program Files\Promixis\Girder\luascript\?.lua" and the library name "socket.html" gives the filename "C:\Program Files\Promixis\Girder\luascript\socket\html.lua"

Dofile, loadfile, loadstring, loadlib, pcall, xpcall

dofile loads and executes the contents of a file as a Lua chunk. **loadfile** and **loadstring** load and compiles the contents of a file or a string and returns a Lua function. **loadlib** loads a C dynamic library into the Lua global environment. **pcall** and **xpcall** execute Lua functions in a protected environment with error trapping.
```
[...] = dofile([filename])
[func], [err] = loadfile([filename])
[func], [err] = loadstring([code], [name])
[func] = loadlib([libname], [funcname])
[success], [...] = pcall([func] ...)
[success], [...] = xpcall([func], [errorhandler])
```
[...]: Indicates multiple call or return parameters.

filename: (string) The filename and path of the Lua file.

func: (function) A Lua compiled function or nil for error.

code: (string) String containing a Lua chunk.

name: (string) Optional debug name for func.

libname: (string) C dynamic library name.

funcname: (string) Name of the initialization function within the library.

© 2008 Promixis, LLC

success: (boolean) **true** if the function executed without errors, else **false** and the error message is the second return parameter.

errorhandler: (function) If an error occurs, this function is passed the error message. It can embellish this and return a new error message.

_G, getfenv, setfenv

Holds, gets and sets the global environment table. **_G** is readonly and is the global environment used by all C functions and by default by Lua functions. A **__fenv** key in a functions current environment table causes an error to be raised in **setenv** and its value is returned instead of the environment table from **getenv**.

```
[envtable] = _G
[envtable] = getfenv([func])
setfenv ([func], [envtable])
```

func: (function or number) If a function, get or set the environment for it. If a number, 1 is the currently running function, 2 is its caller etc.

envtable: (table) The table used as the environment.

Getmetatable, setmetatable

Gets or sets the metatable for an object. If the current metatable has a **__metatable** key, setmetatable raises an error and getmetatable returns this value rather than the real metatable.

```
[metatable] = getmetatable([object])
setmetatable([object], [metatable])
```

metatable: (table) The table which is the metatable for the object, or nil.

object: (userdata or table) Note that it is not possible to set the metatable of a userdata from Lua.

Rawequal, rawget, rawset

Tests for equality, sets or gets table elements bypassing any metamethods.

```
[equal] = rawequal([value], [value])
[value] = rawget([table], [key])
rawset([table], [key], [value])
```

equal: (boolean) True if values are the same type and have equal values or point to the same objects.

table: (table) Table to operate on.

key: Key to use in the table. Must be non-nil.

value: Value to use. If nil indicates that table entry does not exist or should be deleted.

Gcinfo, collectgarbage

```
[inuse], [limit] = gcinfo()
collectgarbage([limit])
```

inuse: (number) Kilobytes of dynamic memory in use.

limit: (number) Garbage collection limit in kilobytes. As input parameter, this is optional defaulting to 0 which forces immediate GC. Otherwise sets a new GC threshold for one GC cycle only.

Source: Lua Basic Function Library & Built-in extensions from Girder.

© 2008 Promixis, LLC

12.3 cm11 Library

The functions send out command messages on an X10 home automation network.
```
cm11.On([house], [device])
cm11.Off([house], [device])
cm11.AllUnitsOff([house])
cm11.AllLightsOff([house])
cm11.AllLightsOn([house])
cm11.Dim([house], [device], [dim])
cm11.Bright([house], [device], [dim])
cm11.DeviceStatus([house], [device]) -- Status response is a Girder Event
```
house: String (A..P). House Code.

device: Number (1..16). Device Number.

dim: Number (1..16). Amount to dim or brighten by.

Source: X10-CM1X Plugin

12.4 comserv Library

This is used to communicate with copies of Girder running on other machines on a network (Girder Pro) and between Girder and NetRemote.

Tip: The NetRemote Library and the Girder2Girder Library provide wrappers which which are much easier than using this library directly.

DiscoverClients initiates a client discovery protocol by sending out a broadcast message on the network. **GetClientTable** and **GetConnections** return tables of status information. **GetNameInfo** and **GetAddrInfo** translate between IP addresses and DNS names. **GetHostName** returns the DNS name of the local machine.
```
[res],[err] = comserv.DiscoverClients()
[clients] = comserv.GetClientTable()
[cons],[err] = comserv.GetConnections()
[name],[err] = comserv.GetNameInfo([address])
[addresses],[err] = comserv.GetAddrInfo([name])
[name],[err] = comserv.GetHostName()
```
clients: Table of all open connections. The outer table is numerically indexed 1 up. The inner tables have the following keys - ["Type"] = string; ["Hostname"] = string; ["Recent"] = boolean; ["Port"] = number; ["Address"] = string (IP address).

cons: Table of all currently known clients. The outer table is numerically indexed 1 up. The inner tables have the following keys - ["Type"] = string; ["Hostname"] = string (IP address); ["Socket"] = number; ["Port"] = number; ["Server"] = boolean.

address: String. An IP address in the form "179.163.9.23".

name: String. A DNS host name (or a URL such as "google.com").

addresses: Table of Strings. Numerically indexed table of IP addresses associated with a DNS name (usually just one entry).

Getting a connection object

You can either establish a new connection or hook an existing one from the socket discovered by **GetConnections** or returned in the callback.
```
[callback]:([cid], [seq], [p1], [p2], [p3])
```

© 2008 Promixis, LLC

```
[Con],[err] = comserv.NewConnection([host], [port], [password], [callback])
[Con],[err] = comserv.GetConnection([socket])
```
host: String. DNS name or IP address of computer to connect to.

port: Number. Port number on remote machine from its configuration settings.

password: String. Password on remote machine from its configuration settings.

callback: Function to be called on asynchronous responses.

socket: Number. The socket number of the connection. This can be obtained from **GetConnections**.

cid: Number. Command Identifier from comserv.MSG table.

seq: Number. Sequence number of the message. This matches the return from the Transaction initiation function and can be used to correlate the response.

p1..p3: Various. Response parameters - see the table below.

Connection management

The connection object supports the following connection management methods. There are two alternate callback prototypes, the second of which has the additional feature of storing and returning a user supplied information table.

```
[callback1]:([cid], [seq], [p1], [p2], [p3])
[ret],[err] = [Con]:Callback([callback1])
[callback2]:([ct], [cid], [seq], [p1], [p2])
[ret],[err] = [Con]:Callback(function(...) return [callback2]([ct], unpack (arg)) end)
[res],[err] = [Con]:Reconnect()
[info] = [Con]:Info()
[status] = [Con]:Status()
[res],[err] = [Con]:Close()
[guid] = [Con]:RemoteInstance()
```
callback1, callback2: Function called when a message is received.

ct: Table. This is an arbitrary table pointer supplied in the Callback method and returned in every callback. It can be used to store additional connection information.

cid: Number. Command Identifier from comserv.MSG table.

seq: Number. Sequence number of the message. This matches the return from the Transaction initiation function and can be used to correlate the response.

p1..p3: Various. Response parameters - see the table below.

info: Table of information about the connection. This table is the same as the inner table of cons above.

status: Number. comserv.CMD table, LOGIN, AUTH_SUCCESS, AUTH_FAIL, CONNECT_FAILED, CLOSE. A connection ready for use has the status comserv.CMD.AUTH_SUCCESS.

guid: String. This is a unique string that identifies the runtime instance of the application being communicated with (Girder or Netremote).

Transactions

The connection object supports the following transaction initiation methods. Each returns a sequence number which can be matched with the response in the callback function. **SendEvent** and **TriggerNode** enable interworking with a Girder instance on another machine. **TransferFile**

© 2008 Promixis, LLC

, **LoadFile**, **UnloadFile PushGML** and **IsFileLoaded** enable GML files to be distributed to other Girder instances over the network. **RunLua** enables script to be sent to and executed on another Girder or Netremote instance. **SetImage** and **SetBinaryImage** enables an image file or memory buffer to be sent to and displayed on Netremote. **SendEvent** is also used to send label texts and navigation commands to Netremote as described later.

```
[seq],[err] = [Con]:Type() -- Request type of client.
[seq],[err] = [Con]:Version() -- Request version of client.
[seq],[err] = [Con]:SendEvent([eventstring],[eventdevice],[eventmod],[p1]...)
[seq],[err] = [Con]:TriggerNode([fileguid], [nodeid])
[seq],[err] = [Con]:TransferFile([filepath]) -- To Uploads directory
[seq],[err] = [Con]:LoadFile([filename]) -- From Uploads directory
[seq],[err] = [Con]:PushGML([filepath]) -- Transfer and Load in one step
[seq],[err] = [Con]:UnloadFile([fileguid])
[seq],[err] = [Con]:IsFileLoaded([fileguid])
[seq],[err] = [Con]:RunLua([luacode])
[seq],[err] = [Con]:SetImage([imagefile],[labelname])
[seq],[err] = [Con]:SetBinaryImage([labelname], [mimetype], [data])
```

seq: Number. Sequence number of message or nil on error.

eventstring: String. Girder eventstring to be generated on remote Girder or NetRemote command (see table below).

eventdevice: Number. Girder device number for event to be generated remotely. Should normally be 232 for the Communication Server device.

eventmod: Number. 0 = No Modifier; 1 = Down; 2 = Up; 4 = Repeat.

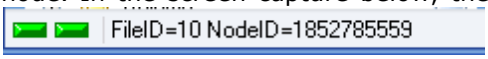
p1..p4: Up to 4 Strings. (At receiving end, first payload is IP address and second is Socket ID, followed by these four.)

filepath: String. Path and name of local file to be transferred. File has same name on remote machine and is stored in an upload directory **%GIRDER%\Uploads** in Girder.

filename: String. Name of GML file to be loaded from **%GIRDER%\Uploads** into remote Girder.

fileguid: String. GUID of GML file. To find this, right-click on the file node in the tree and select Edit. The GUID is shown in the status bar of the editor.

nodeid: Number. The Identifier of the node to trigger in the file (which must be loaded in a remote instance of Girder). When a node is highlighted in the Girder Tree, the node identifier is shown in the status bar. There are two numbers, the first identifies the file, the second the node. In the screen capture below, the node ID is 39590.



luacode: String. Code chunk to run on remote machine. If this chunk **returns** a function taking one argument, this function will be called at the remote end passing a socket number in the parameter. GetConnection can be called within the function to send messages back.

imagefile: String. Path and Name of an image file (JPG, PNG, BMP or GIF) to be displayed on NetRemote.

labelname: String. Name of a NetRemote label on which to display the image.

mimetype: String. "image/bmp"; "image/jpeg"; "image/gif"; "image/png".

data: String. Contains the bytes of the image to be sent.

Callback responses from Transactions

© 2008 Promixis, LLC

CID codes are in the table comserv.CMD.

Function	CID	P1	P2
Type	TYPE_A	string e.g. "GIRDER"	-
Version	VERSION_A	number verH	number verL
SendEvent	ACK	0	-
TriggerNode	ACK	1=OK 0=Fail	-
TransferFile	ACK	Bytes sent	-
LoadFile	ACK	0	-
PushGML	ACK	0	-
UnloadFile	ACK	0	-
IsFileLoaded	ACK	4294967295 = Not Loaded, else GML Ver.	-
RunLua	RUN_LUA_ACK	0 or error number	nil or error text
SetImage	ACK	-	-
SetBinaryImage	ACK	-	-

Other callback responses

CID	Phase	Description
CONNECT_FAILED	Login	Connection failure (i.e. unreachable node)
LOGIN	Login	Connected, Authenticating
AUTH_FAIL	Login	Authentication failure (i.e. incorrect password)
AUTH_SUCCESS	Login	Succeeded, connection ready
NACK	Connected	Unable to process transaction
UNSUPPORTED	Connected	Transaction not supported by node
CLOSE	Connected	Connection closed

NetRemote Commands for SendEvent

Command	P1	P2	Function
SETVAR	name	value	Sets the text of a label
GOHOME	panel	-	Goes to a Home Panel
GODEVICE	device	panel	Goes to a Device Panel or runs a Macro

Source: Communication Server Plugin.

12.5 coroutine Library

create creates a new coroutine thread without starting it. **resume** starts or resumes a coroutine. **yield** suspends the current coroutine. **status** returns the status of a coroutine. **wrap** is an alternative to create and resume, it does a **create** and returns a function which is used in place of **resume**.

```
[thread] = coroutine.create([func])
[err], [...] = coroutine.resume([thread], [...])
```

© 2008 Promixis, LLC

```
[...] = coroutine.yield([...])
[status] = coroutine.status([thread])
[...] = [resumefunc]([...])
[resumefunc] = coroutine.wrap([func])
```
func: (function) The body function to run on the coroutine thread.

thread: (thread) The coroutine thread.

[...]: Zero or more parameters passed between resume and yield. On first resume, the parameters appear as normal in the body function. When the body function terminates its return parameters appear as the returns of resume. Otherwise, **resume** passes parameters to **yield** and **yield** passes parameters to **resume**.

status: (string) "running", "suspended", or "dead".

Source: Lua Basic Function Library.

12.6 date Library

The date library adds a date data type for date and time calculations. Ensure the library is loaded by calling require("date") in any script that uses it.

Date object constructors

```
[Date] = date:new([datetable])
[Date] = date:new([year],[month],[day],[hour],[minute],[second])
[Date] = date:parse([datestring])
[Date] = date:now()
```
datetable: Table. Keys are "Year", "Month", "Day", "Hour", "Minute", "Second" as below.

year: Number, Four-figure year.

month: Number, 1..12.

day: Number, 1..31.

hour: Number, 0..23.

minute: Number, 0..59.

second: Number, 0..59.

datestring: String, "DD.MM.YYYY/hh:mm:ss" If date part is missing, today is used. If time part is missing, 00:00:00 is used. Seconds may be omitted. Date delimiter may be . _ or -. Any delimiter may be used between date and time.

Printing date objects

There is a tostring method enabling date objects to be printed in the form DD.MM.YYYY/hh:mm:ss.

Example:
```
print(date:now())
```

Date calculations

The Date object provides index and operation overrides which enable mathematical operations on components "Year", "Month", "Day", "Hour", "Minute", "Second".

© 2008 Promixis, LLC

Examples:
```
local d = date:now()
d.Day = d.Day + 10         -- d is exactly 10 days from now.
d.Minute = d.Minute - 67   -- 67 minutes before current time.
local b = date:parse("23.05.1958")
local s = d - b            -- difference between two dates in seconds.
```

Date comparison

Comparison operator overrides are provided which enable date tests to work as expected.

Examples:
```
local s = date:parse("01.01.2000")
local e = date:parse("31.12.2001")
local n = date:parse("10.05.2000")
if (n >= s) and (n < e) then print("in interval") end
```

Read only fields and methods

The date object has a number of information fields and methods.

Examples:
```
local s = date:now()
print(s.WeekDay)         -- Day of week as three character string.
print(s.DayOfWeek)       -- Day of week as number 1..7, Monday = 1.
print(s:isleapyear())    -- True if a leap year.
print(s:daysinmonth())   -- Days in the current month.
```

Find date of specific day in month

`[Date2] = [Date1]:findndow([offset], [dayofweek])`

Date1: Date. Any date in the month.

offset: Number. Positive values count days from start of month, negative from end.

dayofweek: Number. 1..7, 1 = Monday.

Example:

```
local d = date:now()
print(d:findndow(-1, 4)) - Date of last Wednesday in this month.
```

Test if curent time is in a specified time interval

`[res] = date.istimebetween([start], [end])`

start: String. Start of interval 24 hour time string "hh:mm" or "hhmm".

end: String. End of interval 24 hour time string "hh:mm" or "hhmm".

res: Boolean. True if current time is within the specified interval.

Source: %GIRDER%luascript\date.lua.

12.7 debug Library

These functions are provided for debugging etc., and adversely affect performance. The privacy of local variables may be violated.

© 2008 Promixis, LLC

Interactive debugging mode
```
debug.debug() -- Enter
cont          -- Leave (type on its own line)
```
Hook functions
```
[hook]([event], [line])
[hook], [mask], [count] = debug.gethook()
debug.sethook([hook], [mask], [count])
debug.sethook() -- Turns off hook.
```
event: (string) "call", "return", "tail return", "line", or "count".

line: (number) Line number for "line" or "count".

mask: (string) "c" on every call; "r" on every return; "l" on every line.

count: (number) Every count instructions.

Returns current hook settings: hook function, mask, and count.

Get function information
```
[info] = debug.getinfo([func], [what])
```
func: (function or number) Function or stack level. func = 1 is the function that called getinfo, 2 is the function that called this etc. Return is nil if invalid. In a hook, 1 is the hook function, so 2 is the function being monitored.

what: (string) If omitted gets all info. Else "n" - name, "S" - source, "l" - line number, "f" - the function itself.

info: (table) Table with entries for the set of requested information.

Access local variables
```
[name], [value] = debug.getlocal([level], [index])
debug.setlocal([level], [index], [value])
```
level: (number) Stack level (see getinfo).

index: (number) Local variables are indexed 1 up. Returns nil if index too large.

name: (string) Name of local variable.

value: Value of local variable.

Access upvalues
```
[name], [value] = debug.getupvalue([func], [index])
debug.setupvalue([func], [index], [value])
```
func: (function or number) Function or stack level. func = 1 is the function that called getinfo, 2 is the function that called this etc. Return is nil if invalid. In a hook, 1 is the hook function, so 2 is the function being monitored.

index: (number) Upvalues are indexed 1 up. Returns nil if index too large.

name: (string) Name of upvalue.

value: Value of upvalue.

Stack traceback

Returns a string with a call stack traceback. An optional message string is prefixed. Typically used with xpcall.
```
debug.traceback([message])
```
message: (string, optional) Message to be prefixed to the stack traceback.

© 2008 Promixis, LLC

Source: Lua Debug Library.

12.8 gip Library

Requires Girder Pro

Note this functionality is implemented by the transport plugin

The Generic IP Library provides a toolkit for writing TCP servers and clients in Lua.

Opening Sockets

Open makes a client connection to a remote server. Listen opens a server connection which waits for connection requests from a remote client.

```
[Socket] = gip.Open([address], [port])
[Socket] = gip.Listen([port])
```

address: String. IP address or DNS name of remote server.

port: Number. IP port to use.

Registering Callbacks

Callbacks on the Socket object are made when connections are made or broken, when data is received or when errors occur.

```
[res] = [Socket]:Callback(0) -- Disable callback.
[callback]([data], [code])
[res] = [Socket]:Callback(1, [bytes], [icto], [callback])
[res] = [Socket]:Callback(2, [terminator], [timeout], [callback])
[res] = [Socket]:Callback(4, [callback]) -- Callback every character
[res] = [Socket]:Callback(100, [callback]) -- Callback at end of connection.
```

bytes: Number. Callback after thin number of bytes received.

icto: Number (milliseconds). Maximum wait between characters.

terminator: String. Callback will be made when these bytes have been received.

timeout: Number (milliseconds). Maximum time to wait for for terminator.

code	data	notes
transport.constants.event. NEWCONNECTION (24576)	Socket Object	Incoming connection request to a Listen
transport.constants.event. CONNECTIONESTABLISHED (65536)	Number 0 = OK or error number	Outgoing connection made or failed
transport.constants.event. RXCHAR (1)	String. Rx Data	Incoming data, Connection remains open
transport.constants.event. CONNECTIONCLOSED (32768)	String. Rx Data	Remaining data, Connection closed

Writing and Polled Reading

Write transmits data over an open socket. Read can be used to get data from the reception buffer, but this is normally done with a callback.

```
[data], [bytes] = [Socket]:Read([bytes])
```

© 2008 Promixis, LLC

```
[bytes], [err] = [Socket]:Write([data])
```
bytes: Number. Maximum number of bytes to read or number of bytes read or written.

data: String. Data read or to be written.

Buffer management
```
[Socket]:SetBufferSize([size])
[Socket]:RxClear()
[Socket]:TxClear()
```

Close Socket

Close shuts down a connection and deletes the Socket object.
```
[res], [err] = [Socket]:Close()
```
size: Number. Maximum Buffer Size.

Utilities

MD5 Digests are often used for client authentication since this is more secure than transmitting plain-text passwords.
```
[digest] = gip.MD5([message])
```
message: String. The plain-text message string (often a password).

digest: String. MD5 Digest of the message (one-way encrypted).

Example - Client Mode
```
function callback(data, code)
    if (code == 16384) then
       print("Connection Closed"); return
    end
    if (code == 32768) then
        if (data == 0) then
            print("New Connection")
        else
            print("New Connection Failed: ", data)
        end
        return
    end
    print(p1)
end
c = gip.Open('www.promixis.com', 80)
c:Callback(2, '\r\n', 1000, callback)
c:Write('GET / HTTP/1.0\r\nHost: www.promixis.com\r\n\r\n')
```

Example - Server Mode
```
function clientcb(p1, p2)
    if (p2 == 16384) then
       print("Client Connection Closed")
        return
    end
    if (p2 == 32768) then
        if (p1 == 0) then
            print("New Client Connection")
        else
            print("New Client Connection Failed: ", p1)
        end
        return
    end
```

© 2008 Promixis, LLC

```
    print(p1)
end
function servercb(p1, p2)
    if (p2 == 16384) then
        print("Server Connection Closed")
        return
    end
    if (p2 == 8192) then
        print("New Incoming Connection")
        p1:Callback(2,'\n', 3000, clientcb)
        client = p1
        return
    end
    print(p1)
end
c5 = gip.Listen(12345)
c5:Callback(2, '\r\n', 1000, servercb)
```

12.9 gir Library

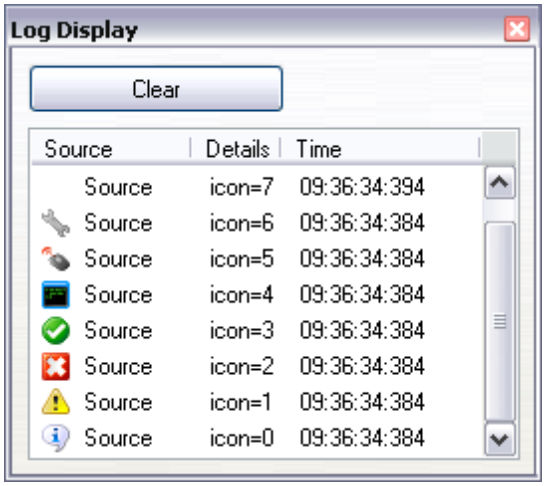
Girder-specific miscellaneous functions.

LogMessage

Writes a message to the Girder Log.
`gir.LogMessage([source], [details], [icon])`
source: String for the source column of the log.

details: String for the details column of the log.

icon: Number. See below for available icons.



The log above was produced using this example script:

```
for i=0, 7 do
   gir.LogMessage("Source", "icon="..i, i)
end
```

SendKeys

Spoof keystrokes to a window.
`gir.SendKeys([windowhandle], [keycodes])`
windowhandle: Number identifier of the window to sent the keystrokes to. nil specifies the foreground window.

© 2008 Promixis, LLC

keycodes: String identifying the keystrokes. The format of the is the same as use by the Keyboard Action.

Example:
```
-- Put calculator in hex mode - Calculator must be running
local wh = win.FindWindow("SciCalc", nil)
if (wh ~= nil) then gir.SendKeys(wh, "") end
```
NOTE: Keyboard spoofing is tricky and can be unreliable. Experiment using the Keyboard Action and Window Picker. If sending keys to the top-level window does not work, try sending them to a relevant child window. For example, typing to the top-level Notepad window will not work, but the "Edit" class child window will work. As a last resort, Focus the window (**win.ForceForegroundWindow(wh)**) and use **windowhandle** = nil.

AddEventHandler, RemoveEventHandler

Install a Lua function to be called when a Girder Event occurs. Similar in effect to attaching an Event to a Script Action in the Girder Tree. Use **AddScriptResetCallback** below rather than installing individual handlers for the **ScriptDisable** event.
```
[ret] = [callback]([eventstring],[device],[modifier],[payloads],[handler])
[handler] = gir.AddEventHandler([eventstring],[mindevice],[maxdevice],[callback])
gir.RemoveEventHandler([handler])
```
eventstring: String. In **AddEventHandler** can be a regular expression. Useful forms are - **".*"** matches any eventstring; **"^Fiddle"** matches any eventstring prefixed "Fiddle"; **"deedee$"** matches any eventstring ending "deedee". In the callback, **eventstring** is the actual eventstring.

mindevice: Number. Minimum device number to recognize events from.

maxdevice: Number. Maximum device number to recognize events from.

device: Number. The actual device number of the triggering event.

modifier: Number. 0 = None; 1 = Down; 2 = Up; 4 = Repeat.

payloads: Table containing 0..4 payload strings indexed 1..4.

Example:
```
function MyCallback(event, device, mod, payloads, id)
   print(event); print(device); print(mod)
    for i=1, 4 do print(payloads[i]) end
end
MyCallbackID = gir.AddEventHandler(".*", 18, 18, MyCallback)
-- And later in cleanup code:
gir.RemoveEventHandler(MyCallbackID); MyCallback = nil; MyCallbackID = nil
```

AddScriptResetCallback, RemoveScriptResetCallback

Installs a Lua function to be called when the Girder Lua scripting engine closes down (either because Girder is exiting or prior to a script reset). This is essentially an efficient shortcut to the **EventHandler** mechanism for the **ScriptDisable** event. It is the preferred method because it avoids multiple callbacks on this event. The callback functions are stored in a weak table which means that a separate reference must be held, usually in a table acting as an object.
```
[callback]()
gir.AddScriptResetCallback([callback])
gir.RemoveScriptResetCallback([callback])
```

TriggerEvent, TriggerEventEx

© 2008 Promixis, LLC

Generates a Girder Event.
`gir.TriggerEvent([eventstring], [eventdevice], [payload] ... )`
`gir.TriggerEventEx([eventstring], [eventdevice], [keymodifier], [payload] ... )`
eventstring: String. The event to trigger.

eventdevice: Number. An integer 25..1999 to spoof an event from a plugin device, or 18 for Girder events or 2000..2100 for user defined events.

keymodifier: Number. 0 = None; 1 = Down; 2 = Up; 4 = Repeat.

payload: String. Up to four string payload parameters can be supplied and these will appear as pld1..pld4 within the Actions triggered by this event.

Example:
```
gir.TriggerEventEx("MyEventString", 18, 0, "My Payload", "My Other Payload")
```

PluginList, EnablePlugin, DisablePlugin, LoadPlugin, UnloadPlugin, PluginStatus

Provides facilities to programmatically control the Plugins tab of Plugins Settings. **Disable** and **Enable** control whether the plugin generates events. **Load** and **Unload** install and remove the plugin including its UI components. **PluginStatus** returns the status of a Plugin.
`[plugins] = gir.PluginList()`
`[res] = gir.EnablePlugin([devnumber])`
`[res] = gir.DisablePlugin([devnumber])`
`[res] = gir.LoadPlugin([devnumber])`
`[res] = gir.UnloadPlugin([devnumber])`
`[status], [loaded] = gir.PluginStatus([devnumber])`
plugins: Table. There is an entry for every available Plugin keyed on the Plugin number, the value is the Plugin name.

devnumber: Number. The Device Number (as per Generic Plugin Information on Settings Dialog).

status: Number. 1 = Not sending events, 2 = Sending events, 3 = Error while enabling. nil if plugin not found.

loaded: Boolean. true if loaded, false if not loaded,

ShowBalloonHint, HideBalloonHint

Shows a hint balloon on the Girder notification icon.
`gir.ShowBalloonHint([title], [text], [timeout], [icon], [eventstring])`
`gir.HideBalloonHint()`
title: String. Bold text at the top of the balloon.

text: String. Multi-line text content of the balloon.

timeout: Number. How long to show the balloon 10..60 seconds.

icon: Number. 0 = None; 1 = Info; 2 = Warning; 3 = Error.

eventstring: String. A Girder (device 18) event is raised with this eventstring when the balloon is clicked.

RunCodeOnMainThread, CoInitialize, CoUninitialize, IsLuaExiting

These functions support multi-threading in Girder (see thread Table).

RunCodeOnMainThread runs a function on the Girder Main Thread regardless of which thread this function is called from. If it is called on the main thread the effect is the same as calling the

© 2008 Promixis, LLC

function directly (it is run synchronously). Otherwise it is queued to be run on the main thread and **RunCodeOnMainThread** returns while it is still pending (it is run asynchronously). **CoInitialize** initializes the Windows COM system for LuaCom on a secondary thread (it is always initialized on the main thread). You MUST balance this with a call to **CoUninitialize** before the thread exits. **IsLuaExiting** returns true in a secondary thread if the Girder Lua system is in the process of shutting down.

```
[code]()
[result], [error] = gir.RunCodeOnMainThread([code])
gir.CoInitialize()
gir.CoUninitialize()
[exiting] = gir.IsLuaExiting()
```

result: Number. 0 = function queued asynchronously, 1 = function has completed synchronously. nil on error.

error: String description of the error.

exiting: Boolean. true if Lua is exiting, false if not.

GetLocation, SetLocation, GetMail, GetUnits

Accesses the information set in the Settings Geographical Location tab and the General tab.

```
[location] = gir.GetLocation()
location.Zipcode   -- string
location.City      -- string
location.State     -- string
location.Country   -- string
location.Latitude  -- number -90.0..+90.0 degrees, negative is South.
location.Longitude -- number -180.0..+180.0 degrees, negative is West.
gir.SetLocation([location])
--
[mailsettings] = gir.GetMail()
mailsettings.Server    -- string
mailsettings.Port      -- number
mailsettings.User      -- string
mailsettings.Password  -- string
--
[units] = gir.GetUnits()
units.Temperature     -- string "Celcius" or "Fahrenheit"
```

CreateTimer

Creates and returns a timer object which may be used to execute Lua code either after a fixed time delay or periodically at equal intervals.

```
[Timer] = gir.CreateTimer([arm], [trigger], [cancel], [recurring], [childstate])
[Timer]:Arm([timeout])
[Timer]:Cancel()
[Timer]:Destroy()
```

arm: Function or String. Code to run when the timer is armed.

trigger: Function or String. Code to run when the timer times out.

cancel: Function or String. Code to run when the timer is canceled.

recurring: Boolean. If true, the timer is automatically rearmed when it times out. If false, the trigger function will be executed once only.

childstate: Boolean (Optional, default false). If true, the functions are executed in a Lua child state. This means they will be less likely to get blocked or delayed by other Lua code, but extra care is required because the timer functions may interrupt other Lua code at any time.

© 2008 Promixis, LLC

timeout: Number. Interval time in milliseconds.

Example:
```
t = gir.CreateTimer("", "print('now!')", "", true)
t:Arm(5000)
-- Later during cleanup
t:Destroy(); t = nil
```

ParseString

Expands a string which may contain Lua variable names enclosed in square brackets by de-referencing each name to a string value.
`[valuestring] = gir.ParseString([namestring])`
namestring: String with Lua variable names embedded.

valuestring: String with values substituted for names.

Example:
```
print(gir.ParseString("LUA_PATH is : [LUA_PATH]."))
```

GetWindowMatches

Returns a list of window handles matching specified criteria. This is the Lua version of the Window Picker for Actions. The handles can be used with the Windows functions in the win Library.
`[handles], [err] = gir.GetWindowMatches([target])`
target: Table containing any of the following keys. ["Name"] string, the main window title bar text; ["Class"] string, the main window class name; ["Filename"] string, the name (no path) of the executable file which created the window; ["ChildName"] string, the child window text; ["ChildClass"] string, the class name of the child window; ["SubMatch"] boolean true to allow substring matching; ["Topmost"] boolean true to match the foreground task; ["OneMatch"] boolean true to return only a single match; ["OneMatchNum"] number, with OneMatch, returns the n'th matching window; ["MatchHidden"] boolean true to allow matching with invisible windows; ["UseVar"] boolean true to expand lua variable names embedded in match strings. ["u_FileVersionMS"], ["u_FileVersionLS"], ["l_FileVersionMS"], ["l_FileVersionLS"] number, components of the file version of the exe file. ["MatchBy"] number, sum of any constants from gir.MatchBy table: Name; Class; Filename; Version; ChildName; ChildClass. If MatchBy is omitted it is generated automatically from the other fields (i.e. exclude from the table any component that should not be used).

handles: Table containing numbers (the Window handles) indexed 1..n. If nil, err contains an error message.

Example:
```
local m = {}
m.Name = "Girder 5.0"
table.print(gir.GetWindowMatches(m))
```

OSD

deprecated and included for compatibility only. Use the OSD Table.

GetRegistryHive

Returns the string containing the registry hive that should be used to store data in the registry. For example "HKEY_CURRENT_USER"
`[valuestring] = gir.GetRegistryHive()`

© 2008 Promixis, LLC

GetRegistryPath

Returns the registry path under which you should store or read settings
```
[path] = gir.GetRegistryPath()
```

GetApplicationDirectory

Returns the Application Data directory to store files. This typically is "Documents and Settings \<your name>\Promixis\Girder\5\" but can be elsewhere.
```
[path] = gir.GetApplicationDirectory()
```

GetConfigDirectory

Returns the Config directory.
```
[path] = gir.GetConfigDirectory()
```

Reset

Resets the Lua Scripting Console.
```
gir.Reset()
```

Source: Built-in.

12.10 globalcache Library

Requires Girder Pro.

The globalcache library provides an object factory function which returns an object for managing the link to a GlobalCache device. Note that you can safely execute this repeatedly as it will return the same object if the connection is already open.
```
[Gc] = globalcache.New([ipaddress])
```
ipaddress: String containing the IP address of the GC. The default address is "192.168.1.70".

The Gc object provides the following methods for IR and Relay modules. **GetState** can be used with IR modules in input mode as well as with relay modules.
```
[Gc]:SendIR([ccf], [repeat], [address], [module])
[Gc]:StopIR([address], [module])
[Gc]:SetRelay([state], [address], [module])
[state] = [Gc]:GetState([address], [module])
```
ccf: String containing CCF code of the IR signal to transmit.

repeat: Number of times to repeat the IR code. 0 = transmit until stopped.

address: Number. Channel address within module (1..3).

module: Number. Module address (see Global Cache Plugin).

state: TO BE ADDED

The Gc object provides the following methods for sending and receiving on a GC serial port. **ReadData** returns immediately with data already in the buffer up to a maximum count. **ReadDataTo** blocks the calling thread waiting for data up to a timeout or until the specified count of data has been received.
```
[Gc]:WriteData([data], [port])
[data] = [Gc]:ReadData([count], [port])
[data] = [Gc]:ReadDataTo([count], [port], [timeout])
[Gc]:ClearBuffer([port])
```
data: String containing ASCII data to be sent or received.

© 2008 Promixis, LLC

port: Number. The port number (see Global Cache Plugin).

count: Number. The maximum number of ASCII characters to return.

timeout: Number. The maximum time in milliseconds to wait for data.

Example:

```
local gc = globalcache.New("192.168.1.70")
if not gc then print("Failed to connect to GC"); return; end
gc:SendIR("0000 0073 0000 000c 0020 0020 0020 0020 0040 0020" ..
        " 0020 0020 0020 0020 0020 0020 0020 0020 0020 0040" ..
        " 0040 0020 0020 0020 0020 0040 0020 0ca5", 1, 4, 1)
```

Source: Global Cache Plugin

12.11 Girder2Girder Library

Requires Girder Pro.

The Communication Server Plugin supplies the ability to network between Girder instances on different machines via the comserv Library. The **Girder2Girder** Library wraps Girder to Girder communications in higher-level functions which are easier to use and more robust.

An associated Tree Script UI provides Girder To Girder settings tabs in the Automation group of the Settings dialog, the Girder to Girder Actions and the Remote Girder Conditional. The Settings tabs provide diagnostic tools, configuration and the ability to distribute GML files to other Girder machines.

Prerequsites

1. The Communication Server Plugin must be loaded, enabled and configured.
2. Any script that uses this library must load it if it is not already loaded using the script function **require('Girder2Girder')**.

High-level Transactions

The high-level interface provides access to the most common Girder to Girder transactions with a single function call. These functions initiate transactions and then return immediately. The library then completes the transaction in the background. Connection management and failure retry is automatic.

Each function has two forms. The version prefixed "Client" sends the transaction to a named Client. The plain version broadcasts the transaction to all available Clients.

The library will retry any transaction up to four times until it gets a response. Connection tracking is used to monitor which clients are online. If a transaction fails, that connection is marked offline. When a connection is not registered as online, a maximum of only one transaction every two minutes is allowed to it in order to probe if it has come back online. This "call gapping" is not applied to the Ping transactions which may be used at any time to actively probe a client.

```
[n] = g2g.SendEvent([event], [device], [mod], [p1], [p2], [p3], [p4])
[n] = g2g.ClientSendEvent([name], [event], [device], [mod], [p1], [p2], [p3], [p4])
[n] = g2g.SendTable([table], [destvar])
[n] = g2g.ClientSendTable([name], [table], [destvar])
[n] = g2g.RunLua([chunk])
[n] = g2g.ClientRunLua([name], [chunk])
[n] = g2g.PushGML([filepath])
[n] = g2g.ClientPushGML([name], [gmlfilepath])
[n] = g2g.UnloadGML([guid])
```

© 2008 Promixis, LLC

```
[n] = g2g.ClientUnloadGML([name], [guid])
[n] = g2g.TransferFile([filepath])
[n] = g2g.ClientTransferFile([name], [filepath])
[n] = g2g.Ping()
[n] = g2g.ClientPing([name])
```

n: Number. The number of clients for which transactions were queued. 0 means an error occurred.

name: String. The Server Name or the IP address of the remote Girder.

event: String. The eventstring of the remote event to be generated.

device: Number (default is the "Lua: Girder to Girder" Event Device). The device number for the event.

mod: Number (default 0). The event modifier of the event 0 = None; 1 = Down; 2 = Up; 4 = Repeat.

p1..p4: String (optional). Payload strings for the event.

table: Table. The table that will be sent to the remote Girder(s).

destvar: String. The name of the variable which will hold the table on the remote Girder(s).

chunk: String. Lua code to be compiled and executed on the remote Girder(s).

gmlfilepath: String. Path and name of a GML file to be sent to and loaded in the remote Girder (s).

filepath: String. Path and name of a non-GML file to be copied to the Upload directory on the remote Girder(s).

guid: String. GUID of the GML file to be unloaded from the remote Girder(s). Note this can be obtained from a local copy of the GML file using guid = g2g.GetGmlInfo([filename]).

Management Functions

Management is normally fully automatic, so the following functions are only required in unusual circumstances. Connections are opened on demand and cached unless explicitly closed. Client registration is not normally required as long as the same password is used for all clients. The password is initially "girder" to match the standard installation. It may be changed (persistently) using g2g.ApplySettings({defpassword="newpassword"}). If clients have different passwords, or if the clients are not discovered automatically (which can happen in some complex networks), **ClientRegister** can be used to supply the necessary information. **PersistRegistrations** stores the registrations so they are loaded automatically with Girder. **GetGmlInfo** returns identification data from a specified GML file.

```
[online] = g2g.IsClientAvailable([name])
[clients] = g2g.GetClients([all])
[status], [clientname], [ip], [port] = g2g.GetClientDetails([name])
g2g.ClientCloseConnection([name])
g2g.CloseConnections()
g2g.ClientRegister([name], [ip], [port], [password])
g2g.ClientUnregister([name])
g2g.PersistRegistrations()
g2g.ApplySettings([settings])
[settings] = g2g.GetSettings()
[guid], [ver], [intname] = g2g.GetGmlInfo([filepath])
```

name: String. The Server Name or the IP address of the remote Girder.

© 2008 Promixis, LLC

online: Boolean. True if the named client is available for transactions.

all: Boolean. True to return all clients known about, not just those online.

clients: Table, indexed from 1. Each entry is a string, a Server Name.

status: String. "no connection", "connected" or "connection lost".

clientname: String. Name of client.

ip: String. IP address of client.

port: Number. IP Port used by client.

password: String. The password set in the remote Girder comserv settings.

settings: Table. **ApplySettings** can include any of the following keys, **GetSettings** returns a copy of all the keys. **defpassword** (string) password used for clients that are not pre-registered; **enabled** (boolean) true to enable Girder2Girder; **diag** (boolean) true for diagnostic messages; **congap** (number) maximum time in seconds for a connection to become available; **recgap** (number) interval in seconds between retry attempts; **ackgap** (number) maximum time in seconds for a transaction to be acknowledged; **interval** (number) milliseconds between retry and timeout checks.

guid: String or nil. Returns a unique string identifying a GML file. **nil** means that the file could not be found. An empty string indicates that the file was found, but was not a valid GML file. The string is set when the file is created in Girder and it does not change if the file is copied or modified.

ver: Number. The file version number can be interrogated on a remote Girder and it can be updated in Girder using the File Properties editor.

intname: String. The Internal Name of the file. This is the name shown in the file node in Girder. By default it is the file name without the path, but it can be changed to any string in Girder.

deprecated Management Functions

```
g2g.SetPassword([password])
-- g2g.ApplySettings({defpassword=password})
g2g.ClientSetPassword([name], [password])
-- g2g.ClientRegister(name, nil, nil, password)
```

Low-level Transactions

The high-level interface is built using a low-level generic queuing system which may be used directly for more control of transactions, including assembling chains of dependent transactions, acting on response messages and processing failures.

```
[n] = g2g.Enqueue([wp])
```

This function places a **workpackage** on the queue and returns the number of clients for which transactions were queued.

The Workpackage (wp) is a table with the following fields.

Compulsory fields for the wp when passed to Enqueue.

dofunc: Function. This is the function used to initiate the transaction. It receives the wp and returns true to continue processing, false on failure. A **dofunc** may require additional parameters which must be passed as extra fields in the wp. If **address** is specified (see below) the field **con** will be present in the wp and will be an available comserv connection object. In this case, field **seq** should be set to the comserv transaction sequence number, and ackcid to the expected acknowledgement CID, unless this is the standard value of 2 (see comserv

© 2008 Promixis, LLC

Library) before returning **true**.

Optional fields for the wp when passed to Enqueue.

address: String or boolean. If a string, it may be a Server Name or an IP address of a client. If boolean **true**, the wp will be duplicated for each available client (broadcast). If boolean **false** or, in a root wp, **nil**, no comserv connection will be supplied to dofunc.

retries: Number (default = 4). The number of retries that will be made to establish a connection and get a confirmation.

force: Boolean (default = false). If true, a broadcast expands to all known clients, not just those believed online, and call-gapping for clients believed off-line is suppressed causing the transaction to be always attempted immediately.

delay: Number. If provided this is a number of seconds to delay the transaction by. The value is limited to 300 seconds and the accuracy is -0, +2secs. The delay is between the establishment of a connection (if required) and the initiation of the transaction. This facility is most useful for chained transactions when a gap must be left between transactions.

ackfunc: Function. This optional function if supplied is called when a response to the transaction initiated in **dofunc** is received. The keys **ackcid**, **ackp1**, **ackp2** and **ackp3** will be set to the values returned in the callback. The function should return **true** for success or **false** for failure. **false** will put the wp into retry, but this can be suppressed by setting **retries** = -1 to give immediate failure.

failfunc: Function. This optional function if supplied is called when a transaction cannot be completed or the retries limit is reached. The additional field **error** gives a message describing the failure.

Transaction Tree optional fields (see below).

dispfunc: Function. This optional function if supplied is called when a parent wp is about to enqueue a child wp. It is passed the parent wp and the child wp. The child wp will be enqueued if **dispfunc** does not exist or if it returns **true**.

next: Table. If supplied, must be a wp to be enqueued when and if the current wp completes successfully.

also: Table. If supplied, must be a wp to be enqueued at the same time as this one. This is useful in a **next** wp so that two or more transactions may be launched on success of the original transaction.

Transaction initiation phase (dofunc) fields.

con: comserv Connection object. Unless **address** was **false**, this key will contain a valid object ready for use when the wp is passed to **dofunc**.

seq: Number. The **dofunc** should set this to the sequence number from the comserv transaction. If this is **nil** after a **true** return from **dofunc**, the wp will complete with immediate success.

ackcid: Number. The dofunc should set this to the expected acknowledgement code, unless this is the default value of 2.

Transaction response phase (ackfunc) fields.

ackp1..ackp2: Various types. The **p1** and **p2** response fields from the comserv callback.

Transaction error phase (failfunc) fields.

© 2008 Promixis, LLC

error: String. Error message.

The functions used for **dofunc**, **ackfunc** and **failfunc** in the high-level transactions are available for used in workpackages. All of these functions take a table (the wp) as input parameter and return a boolean.

g2g.fn.TriggerLocalEvent: Triggers an event in the local Girder. Uses additional fields **eventstring** String, required. **device** Number, optional default = 10012 which is registered as lua: Girder to Girder. **keymod** Number, optional default = 0. **p1**..**p4** String, optional.

g2g.fn.TriggerRemoteEvent: Triggers an event in a remote Girder. Uses additional fields **eventstring** String, required. **device** Number, optional default = 10012 which is registered as lua: Girder to Girder. **keymod** Number, optional default = 0. **p1**..**p4** String, optional.

g2g.fn.TriggerPing: Triggers a test transaction to a remote Girder.

g2g.fn.RunLuaRemote: Initiates a RunLua transaction in a remote Girder. Uses additional field **code** String, required. This is the Lua code in text form.

g2g.fn.PushGML: Initiates a PushGML transaction in a remote Girder. Uses additional field **GML** String, required. This is the path and filename of the GML file to be sent to and loaded in the remote Girder.

g2g.fn.UnloadGML: Initiates an UnloadGML transaction in a remote Girder. Uses additional field **GUID** String, required. This is the GUID of the GML file to be unloaded from the remote Girder.

g2g.fn.CheckGML: Initiates an IsFileLoaded transaction in a remote Girder. Uses additional filed **GUID** String, required. This is the GUID of the GML file. The **ackfunc** in response to this transaction will have the version number of the file in **ackp1** or the pseudo-version 4294967295 if the file is not loaded.

g2g.fn.TransferFile: Initiates a TransferFile transaction in a remote Girder. Uses additional field **filename** String, required. This is the path and filename of the file to be sent to the Uploads directory on the remote Girder.

g2g.fn.TriggerStatusEvent: For use in **dofunc** or **failfunc**. Triggers an event in the local Girder. If the key **status** is present, it is used as the eventstring. If not, the eventstring is "Ack" or "Fail" concatenated with the Client Name. P1 is the Client Name and P2 is the error string if present.

g2g.fn.CheckRunLuaResponse: For use in **ackfunc**. Analyses the response from a RunLua transaction and reports any error to the Interactive Lua Console. Suppresses further retries on an error response (no point if there is a Lua error).

Example 1

The following examples print a message to the Interactive Lua Console of another Girder machine. Substitute your own Server Name for 'cruncher'.

```
require('Girder2Girder')
-- High-level transaction
g2g.ClientRunLua('cruncher', [[print("Message from Girder to Girder")]])
-- Low-level transaction
local wp = {}
wp.address = 'cruncher'
wp.dofunc = g2g.fn.RunLuaRemote
wp.ackfunc = g2g.fn.CheckRunLuaResponse
wp.code = [[print("Message from Girder to Girder")]]
g2g.Enqueue(wp)
```

Transaction Trees

© 2008 Promixis, LLC

This feature allows complex trees of workpackages with interdependencies to be managed automatically in the background. The initial wp passed to Enqueue may refer to another wp using the key **next**. The first wp is termed the parent, the second the child. The child wp may refer to a sibling using the **also** key and that sibling may refer to another in the same way and so on indefinitely. Thus a parent may have any number of children. The children may themselves have **next** links becoming parents in their turn. The basic rule is that when and if a parent wp completes successfully, it will enqueue all of its children. If it fails, the children will be discarded.

Child dispatch can be made conditional by including a **dispfunc** in the child wp. The child will only be enqueued if **dispfunc** returns **true**. **dispfunc** is passed its own wp as usual, but a second parameter is the parent wp. This allows **dispfunc** to take a decision based on information gathered during parent execution and also enables it to pass information through from the parent to the child.

If not specified in a child wp, **address** and **failfunc** keys are copied from the parent wp. This has particular significance for **address** because the copy happens after broadcast expansion. If a parent is a broadcast (**address** = true), it (and all its descendants) will be copied for each available client and the actual client address substituted in the root wp of each copy. If the children have no **address**, they will inherit the actual client **address** from the parent. To make a child actually broadcast, **address** should be explicitly set to **true** (take care as this can expand to an enormous number of transactions!). If a child wp does not require a connection, then **address** should be explicitly set **false**.

Example 2

```
require('Girder2Girder')
local tf = function(wp)
 wp.seq = wp.con:IsFileLoaded(wp.guid)
 return true
end
local wpp = {}
local wp1 = {}
local wp2 = {}
wpp.dofunc = tf
wpp.ackfunc = function(wp) wp.loaded = (wp.ackp1 == 0) return true end
wpp.guid = "{411E5888-283F-4CBC-9612-5B6E1E18730D}"
wp1.dispfunc = function(wp, wpp) return wpp.loaded end
wp1.dofunc = g2g.fn.RunLuaRemote
wp1.ackfunc = g2g.fn.CheckRunLuaResponse
wp1.code = [[print("TODO - GML Loaded")]]
wp2.dispfunc = function(wp, wpp) return not wpp.loaded end
wp2.dofunc = g2g.fn.RunLuaRemote
wp2.ackfunc = g2g.fn.CheckRunLuaResponse
wp2.code = [[print("TODO - GML Not Loaded")]]
wpp.next = wp1
wp1.also = wp2
g2g.Enqueue(wpp)
```

The first transaction determines if the demonstration GML shipped with Girder is loaded in each remote Girder. It then runs one of two chunks of Lua code depending on the result (the library function g2g.fn.CheckGML could be used in place of **tf**).

Source: %GIRDER%luascript\Girder2Girder.lua.

12.12 io Library

Standard files

© 2008 Promixis, LLC

These standard File objects are pre-opened.
```
[File] = io.stdin
[File] = io.stdout
[File] = io.stderr
```
General file access

The open method can be used to open any file and interact with it through a File object.
```
[File] = io.open([filename], [mode])
[...] = [File]:read([format1] ...)
[res], [err] = [File]:write([text] ...)
[pos] = [File]:seek ([ref], [offset])
for line in [File]:lines() do
[File]:flush ()
[File]:close ()
```
filename: (string) Path and name of file to be opened.

mode: (string) "r" - read mode (default); "w" - write mode; "a" - append mode; "r+" - update mode, previous data preserved; "w+" - update mode, previous data erased; "a+" - append update mode (only allows writing at end of file). Plus "b" may be added to specify binary mode.

format: (string or number) If a string, one return value per format. "*n" reads a number; "*a" reads the remainder of the file into a string; "*l" reads the next line into a string. If a number, reads up to that number of characters to a string.

text: (string) The text to write to the file.

ref: (string) "set", "cur" or "end", default is "cur". offset is from beginning ("set") the current position or the end of the file.

offset: (number) Position in file relative to ref. Default is 0 so seek() just gets the current position.

pos: (number) Position in file in bytes from the start of the file. Beginning of file is pos = 0.

Line iterator
```
for line in io.lines([filename]) do
```
filename: (string) Path and filename for file to be iterated.

Default files

This represents an alternative system to General file access. **input** opens the default input file, **output** opens the default output file. **read** is equivalent to io.input():read(), **write** is equivalent to io.output():write(), **flush** is equivalent to io.output():flush(). **close** is equivalent to io.output ():close() by default.
```
[File] = io.input([file])
[File] = io.output([file])
[...] = io.read([format1] ...)
[res], [err] = io.write([text] ...)
io.flush()
io.close([file])
```
file: (string or userdata) Filename or previously opened file handle.

Temporary files

Returns a handle for a temporary file, opened in update mode. Automatically removed when the program ends.
```
[File] = io.tmpfile()
```
Checking file handles

© 2008 Promixis, LLC

```
[type] = io.type([File])
```
type: (string) "file" if an open file handle, "closed file" if a closed file handle, and **nil** if not a file handle.

Source: Lua I/O and OS Facilities Library

12.13 keyboard Library

The keyboard table supplies three functions for suppressing events from specific keyboard keys. The settings are stored in the registry, so do not need to be reestablished on startup.
```
keyboard.AddBlockedKey([key])
keyboard.RemoveBlockedKey([key])
keyboard.ClearBlockedKeys()
```
key: String. The eventstring of the key to be blocked (Q = "51000000"). Use the Logger to get the eventstring of the key you want to block.

Source: Keyboard Plugin

12.14 luacom, luacomE Library

LuaCOM is a Lua library for interfacing with and for creating Microsoft COM Automation objects. luacom is the standard version, luacomE the extended version.

Full documentation is available here - LuaCOM User Manual.
(http://www.tecgraf.puc-rio.br/~rcerq/luacom/pub/1.2/luacom-browseable/luacom.html)

Source: Built-in.

12.15 lxp Library (XML Parser)

This table represents the LuaExpat XML parser, an implementation of the SAX parser API for Lua.

Full documentation for the parser is here - LuaExpat Manual.
(http://www.keplerproject.org/luaexpat/manual.html)

Source: Built-in + %GIRDER%\luascript\cwlom.lua.

12.16 math Library

The math table is a standard Lua library. Girder adds some functions to the standard set.

Trigonometric functions

Except for input to **math.rad**, and result of **math.deg**, all angles are radians.
```
[v] = math.pi
[v] = math.cos([a])   [a] = math.acos([v])
[v] = math.sin([a])   [a] = math.asin([v])
[v] = math.tan([a])   [a] = math.atan([v])
[a] = math.rad([a])   [a] = math.deg([a])
[a] = math.atan2([a1], [a2]) -- atan(a1 / a2)
```

© 2008 Promixis, LLC

Numerical functions

mantissa is a floating point number. **exponent** is a negative or positive integer power of 10.

```
[v] = math.abs([v])
[mantissa], [exponent] = math.frexp([v])
[v] = math.ldexp([mantissa], [exponent])
[v] = math.mod([v1], [v2]) --remainder(v1/v2)
[v] = math.pow([v1], [v2]) --v1^v2 also global __pow
[v] = math.log([v])   [v] = math.log10([v])
[v] = math.exp([v])   [v] = math.sqrt([v])
[v] = math.floor([v]) --largest integer < v
[v] = math.ceil([v])  --smallest integer > v
```

List functions

Selects the maximum or minimum element from a list of elements.

```
[v] = math.max([v1] ...)
[v] = math.min([v1] ...)
```

Random number generator

Generates pseudo-random numbers in specified ranges. **math.randomseed** is not normally needed, but the random numbers produced will be the same sequence if the same seed is set twice.

```
math.randomseed([seed])
math.random()            -- Range 0 to 1
math.random([max])       -- Range 1 to max
math.random([min], [max]) -- Range min to max
```

Bitwise operations

These are Girder additions.

```
[c] = math.band([a], [b])    -- Bitwise And
[c] = math.bor([a], [b])     -- Bitwise Inclusive Or
[c] = math.bxor([a], [b])    -- Bitwise Exclusive Or
[c] = math.bnot([a])         -- Bitwise Not (Inversion)
[c] = math.bshiftl([a], [l]) -- Shift bits in [a] left by [s] bits.
[c] = math.bshiftr([a], [l]) -- Shift bits in [a] right by [s] bits.
[c] = math.zerobits([a], [b])-- Clear the bits in [a] which are set in [b].
[c] = math.bitextract([a], [l], [m]) -- Extract bits from index [s] to [e].
                            -- index right to left, starting at 1.
[l] = math.bitcount([a])     -- index of most significant bit in [a].
```

a, b, c: Number - 32-bit unsigned integer.

l, m: Number - Bit index or shift count 1 to 32.

Conversions

These are Girder additions.

```
[hexstring] = math.formatbytes([string]) -- ASCII string to printable hex string.
[char] = math.decimaltobyte([dec]) -- Decimal number to ASCII character.
[dec] = math.bytetodecimal([char]  -- ASCII character to decimal number.
[dec] = math.hextodecimal([hex]) -- Hex string to decimal number.
[hex] = math.decimaltohex([dec]) -- Decimal number to hex string.
[hex] = math.bytetohex([char]) -- ASCII character to hex string.
[char] = math.hextobyte([hex]) -- Hex string to ASCII character.
[bin] = math.decimaltobinary([dec], [l]) -- Decimal to binary string, length l.
```

hexstring: String. Pairs of hexadecimal digits separated by spaces.

string: String. Interpreted as a byte array.

© 2008 Promixis, LLC

char: String. Single character string representing a byte 0..255.

dec: Number. Positive integer.

hex: String. Containing 0..9, A..F, representing a hexadecimal number.

bin: String. Containing 0..1, representing a binary number.

CRC Calculation

```
[crc number] = math.process_crc_16( [string source], [optional initial crc value] )
[crc number] = math.process_crc_sick( [string source], [optional initial crc value ] )
[crc number] = math.process_crc_ccitt( [string source], [optional initial crc value] )
[crc number] = math.process_crc_kermit( [string source], [optional initial crc value] )
[crc number] = math.process_crc_dnp( [string source], [optional initial crc value] )
[crc number] = math.process_crc_32( [string source], [optional initial crc value] )
```

All functions above return the CRC requested, you can pass an optional initial CRC value to override the default. These functions are useful if you have the full buffer available in a string, if not you can process your data byte for byte with the following functions.

[crc number] = math.init_crc_16()
[crc number] = math.init_crc_sick()
[crc number] = math.init_crc_ccitt()
[crc number] = math.init_crc_kermit()
[crc number] = math.init_crc_dnp()
[crc number] = math.init_crc_32()

[crc number] = math.update_crc_16([crc number], [byte])
[crc number] = math.update_crc_sick([crc number], [byte], [previous byte])
[crc number] = math.update_crc_ccitt([crc number], [byte])
[crc number] = math.update_crc_kermit([crc number], [byte])
[crc number] = math.update_crc_dnp([crc number], [byte])
[crc number] = math.update_crc_32([crc number], [byte])

[crc number] = math.finish_crc_16([crc number])
[crc number] = math.finish_crc_sick([crc number])
[crc number] = math.finish_crc_ccitt([crc number])
[crc number] = math.finish_crc_kermit([crc number])
[crc number] = math.finish_crc_dnp([crc number])
[crc number] = math.finish_crc_32([crc number])

Here is an example on how to use these function to calculate the 'CRC-16 (SICK)'.

```
function process_crc_sick ( s, init )
   if not init then
       init = math.init_crc_sick()
   end

   local last = 0
   for i=1, string.len(s) do
       local b = string.byte(s,i)
       init = math.update_crc_sick(init,b,last)
       last = b
   end
   return math.finish_crc_sick(init)
end
```

© 2008 Promixis, LLC

Source: Lua Mathematical Function Library & Girder built-in extensions.

12.17 mixer Library

Provides Lua functions for working with the computers audio mixer facilities. See Volume Picker for a description of mixer addressing and a tool for exploring resources available on a particular machine.

SetMasterVolume, SetMasterMute, SetWaveVolume, SetWaveMute

These functions allow setting of the Master volume and mute controls, and the wave (digital playback) volume and mute directly without having to discover the appropriate addresses.
```
[res] = mixer.SetMasterVolume([volume])
[res] = mixer.SetMasterMute(mixer.Mute)
[res] = mixer.SetMasterMute(mixer.Unmute)
[res] = mixer.SetWaveVolume([volume])
[res] = mixer.SetWaveMute(mixer.Mute)
[res] = mixer.SetWaveMute(mixer.Unmute)
```
volume: Number (0..100). Volume level as a percentage of full volume.

GetNumberWaveOutDevices, GetWaveOutDeviceName, SetDefaultWaveOutDevice SetDefaultWaveOutDeviceName

These functions allow control of which audio device is used for wave audio output from the computer. This will be used by most audio and video file players, speech synthesizers etc.
```
[devcount] = mixer.GetNumberWaveOutDevices()
[name] = mixer.GetWaveOutDeviceName([devno])
[res] = mixer.SetDefaultWaveOutDevice([devno])
[res] = mixer.SetDefaultWaveOutDeviceName([name])
```
devcount: Number. Total number of available wave out devices.

devno: Number (0..). Available devices are numbered 0 up to devcount - 1.

name: String. Name of the device to be made the default.

GetNumberWaveInDevices, GetWaveInDeviceName, SetDefaultWaveInDevice SetDefaultWaveInDeviceName

These functions allow control of which audio device is used for recording audio input to the computer hard drive. This will be used by by most audio recording devices that use the microphone or line-in input.
```
[devcount] = mixer.GetNumberWaveInDevices()
[name] = mixer.GetWaveInDeviceName([devno])
[res] = mixer.SetDefaultWaveInDevice([devno])
[res] = mixer.SetDefaultWaveInDeviceName([name])
```
devcount: Number. Total number of available wave in devices.

devno: Number (0..). Available devices are numbered 0 up to devcount - 1.

name: String. Name of the device to be made the default.

GetDeviceCount, GetDeviceName, ShowDeviceDetails

Get the number and names of the Audio Mixer devices on the system. **ShowDeviceDetails** prints a complete device enumeration to the Interactive Lua Console.
```
[count] = mixer.GetDeviceCount()
[name] = mixer.GetDeviceName([devno])
[res] = mixer.ShowDeviceDetails([devno])
```
count: Number of devices on the system.

© 2008 Promixis, LLC

devno: Number. Device number 0..(count - 1)

name: String. Name of the device.

GetDestinationCount, GetDestinationName

Get the number and names of the destinations provided by a particular device.
[count] = mixer.GetDestinationCount([devno])
[name] = mixer.GetDestinationName([devno], [destno])
count: Number of destinations.

destno: Number. Destination number 0..(count - 1).

name: String. Name of destination.

GetSourceCount, GetSourceName

Get the number and names of the sources provided for a particular destination.
[count] = mixer.GetSourceCount([devno], [destno])
[name] = mixer.GetSourceName([devno], [destno], [srcno])
count: Number of sources.

srcno: Number. Source number -1 or 0..(count - 1). -1 is a pseudo channel which represents the destination controls (i.e. the mixer output).

name: String. Name of source.

GetControlCount, GetControlForSource, GetControlName, GetControlType, GetControlTypeName, GetControlBounds, GetChannelCount

Gets the number of controls provided for a source and information about each source.
[count] = mixer.GetControlCount([devno], [destno], [srcno])
[ctlno] = mixer.GetControlForSource([devno], [destno], [srcno], [type])
[name] = mixer.GetControlName([devno], [destno], [srcno], [ctlno])
[type] = mixer.GetControlType([devno], [destno], [srcno], [ctlno])
[typnam] = mixer.GetControlTypeName([type])
[min],[max],[type] = mixer.GetControlBounds([devno], [destno], [srcno], [ctlno])
[chans] = mixer.GetChannelCount([devno], [destno], [srcno], [ctlno])
count: Number of controls.

ctlno: Number. The Control Number 0..(count - 1).

type: Number. One of the constants (all prefixed mixer.) ControlTypeBass, ControlTypeEqualizer, ControlTypeFader, ControlTypeMute, ControlTypeMux, ControlTypeOnOff, ControlTypeTreble, ControlTypeVolume, ControlTypeLoudness.

name: String. Name of the control.

typnam: String. Text corresponding to control type number.

min: Number. Minimum value the control can have.

max: Number. Maximum value the control can have.

chans: Number. Number of audio channels the control can separately adjust (2 for stereo).

SetSourceVolume, SetSourceMute, GetChannelValue, SetChannelValue

Gets and sets the control parameters of a control. The **SetSource*** functions determine which

© 2008 Promixis, LLC

control to use automatically and effect all channels uniformly (they do not alter the audio balance). Use **SetChannelValue** to adjust other controls or to alter balance.

```
[res] = mixer.SetSourceVolume([devno], [destno], [srcno], [volume])
[res] = mixer.SetSourceMute([devno], [destno], [srcno], mixer.Mute)
[res] = mixer.SetSourceMute([devno], [destno], [srcno], mixer.Unmute)
[value],[err] = mixer.GetChannelValue([devno], [destno], [srcno], [channo])
[res] = mixer.SetChannelValue([devno], [destno], [srcno], [channo], [value])
```

volume: Number 0..100. Percentage of full volume.

res: nil on error.

channo: Number -1, 0..(channels - 1). The channel number. -1 means adjust all channels uniformly.

value: Number. Raw value of control min..max.

Tip: srcno = -1 specifies the destination (master) controls.

RegisterForVolumeEvents, UnregisterForVolumeEvents

Specifies which audio devices(s) will generate Girder Events.

```
[res] = mixer.RegisterForVolumeEvents([devno])
[res] = mixer.UnregisterForVolumeEvents([devno])
```

Source: Audio Mixer (Sound) Plugin

12.18 monitor Library

In a system with multiple monitors, there is a *virtual desktop* - the smallest rectangle that encloses all of the pixels on all of the active monitors. One monitor is designated the *primary monitor* and its top left pixel always has address 0:0 on the virtual desktop. Thus a secondary monitor above and to the left of the primary will have negative coordinates.

The rectangle of the virtual desktop and of each monitor can be read from the following Number type variables.

```
[vl] = monitor.VirtualDesktopLeft
[vt] = monitor.VirtualDesktopTop
[vw] = monitor.VirtualDesktopWidth
[vh] = monitor.VirtualDesktopHeight
-- monitor numbers are 1 up to monitor.Monitors.Count.
for n = 1, monitor.Monitors.Count do
    [x] = monitor.Monitors[n].PositionX
    [y] = monitor.Monitors[n].PositionY
    [w] = monitor.Monitors[n].Width
    [h] = monitor.Monitors[n].Height
end
```

Tip: The monitor.Monitors table has other information on each monitor which may be useful. Check in Variable Inspector.

Find particular monitors on the virtual desktop

```
[mon] = monitor.GetPrimaryMonitor()
[res] = monitor.IsPrimaryMonitor([mon])
[mon] = monitor.GetNearestMonitor([left], [top, [right], [bottom])
[mon] = monitor.GetNearestMonitor([x], [y])
[mon] = monitor.GetNearestMonitor([hwnd])
```

mon: Number. Primary monitor number, nil on failure.

© 2008 Promixis, LLC

res: nil if mon is NOT the primary monitor.

left: Number. X coordinate of left side of rectangle.

top: Number. Y coordinate of top of rectangle.

right: Number. X coordinate of right side of rectangle.

bottom: Number. Y coordinate of bottom of rectangle.

x: Number. X coordinate of point.

y: Number. Y coordinate of point.

Refresh the monitor table.
`[mons] = monitor.GetDesktopMonitors()`
mons: Number. Number of monitors found.

Monitor display area coordinates

GetMonitorRect gets the screen coordinates of a particular monitor (should be the same as in the Monitors table). GetMonitorWorkAreaRect is the same except that the area excludes any taskbars.
```
[l], [t], [r], [b] = monitors.GetMonitorRect([mon])
[l], [t], [r], [b] = monitors.GetMonitorWorkAreaRect([mon])
```
mon: Number. Monitor number or nil for default monitor.

l, t, r, b: Number. Left, Top, Right, Bottom screen coordinates of area.

Configure monitors
```
[w],[h],[b],[f] = monitor.GetDisplayMode([mon])
[modes] = monitor.GetAllDisplayModes([mon])
[res] = monitor.SetDisplayMode([mon], [w], [h], [b], [f])
[res] = monitor.ChangeDisplaySettingsEx([dev], [f], [h], [w], [b]) -- Testing only.
```
mon: Number. Monitor number or nil for default monitor.

w: Number. Width in pixels.

h: Number. Height in pixels.

b: Number. Bits per pixel.

f: Number. Refresh rate. Refreshes per second.

modes: Table with indexed Table for each available mode. Mode table has keys BitsPerPixel, Width, Height, Frequency.

dev: String. Device name (this can be obtained from monitor.Monitors[mon].DeviceName)

Map monitors onto the virtual desktop
```
[res],[err] = monitor.EnableMonitor([mon], [xpos], [ypos])
[res],[err] = monitor.MoveMonitor([mon], [xpos], [ypos])
[res] = monitor.DisableMonitor([mon])
[res] = monitor.SwapPrimaryMonitor([oldmon], [newmon], [pos])
```
mon: Number. Monitor to act on.

xpos: Number. X coordinate of monitor top-left on virtual desktop.

ypos: Number. Y coordinate of monitor top-left on virtual desktop.

© 2008 Promixis, LLC

oldmon: Number. Current primary monitor.

newmon: Number. New primary monitor.

pos: monitor.LEFT or monitor.RIGHT

Test and control object visibility
```
[res] = monitor.IsOnScreen([hwnd])
[res] = monitor.IsOnScreen([x], [y])
[res] = monitor.IsOnScreen([l], [t], [r], [b])
[res] = monitor.CenterWindowToMonitor([mon], [hwnd])
```
hwnd: Number. Handle of window to be moved to another monitor.

x, y: Number. Screen coordinates of point to test.

l, t, r, b: Number. left, top, right, bottom screen coordinate of rectangle to test.

mon: Number. Monitor to center window on.

Turn monitors on and off
```
monitor.SetMonitorPower([on])
```
on: Boolean. **true** to turn on, **false** to turn off

Display Change Event Handlers

These are called when the Girder event "OnDisplayChange" happens, which should detect any change in resolution or position of a monitor even if caused by another program or user action.
```
monitor.AddDisplayChangeHandler([callback])
monitor.RemoveDisplayChangeHandler([callback])
```
callback: Function. Takes no parameters and returns none.

Source: Multimonitor Extensions Plugin

12.19 mutil Library

GetCDDBID

Reads a CD or DVD and calculates the CDDB ID which can be used to look up information about the content and artists on the internet. See CDDB Developer Information (http://www.gracenote.com/developer/).
```
[cddbid], [tracks], [cddbidquery], [intrf] = mutil.GetCDDBID([path], [intrf])
```
path: String path to drive i.e. "C:\\" or "ASPI[1:1:0]"

intrf: Number. Interface type 2 = ioctl (usual on PC), 1 = spti, 0 = aspi.

cddbid: Number. CDDB Disk ID Number or nil on error.

tracks: Number. Number of tracks on the disk.

cddbidquery: String. Query string for CDDB.

Example:
```
print(mutil.GetCDDBID([[D:\]], 2))
```

GetWAF

Gets the name of the current song being played by the WinAmp media player application.
```
[song] - mutil.GetWAF()
```

© 2008 Promixis, LLC

song: String. Song name.

Source: Lua Misc Function Library Plugin

12.20 NetRemote Library

The Communication Server Plugin automatically allows NetRemote to raise events within Girder. It also provides basic functions for controlling NetRemote from Girder Scripts via the comserv Library. The NetRemote library wraps Girder to NetRemote communications in higher-level functions which are easier to use.

There is a NetRemote Configuration tab in the Settings dialog, Plugins section, which offers some useful diagnostic facilities.

Prerequsites

1. The Communication Server Plugin must be loaded, enabled and configured (check that NetRemote events show up in the Event Log).
2. Any script that uses this library must check that it is loaded using the Lua statement **require('NetRemote')**.

Commands

SetVariable (with a string value), **SetImage**, **SetImageURL**, **SendStemmedTable** and **SendMappedTable** all create and change NetRemote variables. **SetVariable** (with a table value) creates or changes a Lua table in NetRemote. **Jump** changes the page displayed in NetRemote and **RunLua** sends some Lua code to NetRemote and runs it there. This could, if required, send a message back to Girder using the NetRemote Girder Plugin.

```
NetRemote.SetVariable([var], [val], [noskip])
NetRemote.SetImage([var], [filename], [noskip])
NetRemote.SetImageURL([var], [url])
NetRemote.SendStemmedTable([basevar], [table])
NetRemote.SendMappedTable([table], [mapping], [prefix], [suffix])
NetRemote.Jump([pagegroup], [page])
NetRemote.RunLua([luacode])
```

var: String. Name of NetRemote variable (when used with a string val) or a Lua variable (when used with a table val).

val: String or Table. Value for NetRemote variable. With a string value, a NetRemote (not Lua) variable is set. With a table value, a Lua table in NetRemote is created if it does not already exist and synchronized with the Girder table. In this case, a NetRemote variable with the same name but suffixed ".changed" is incremented each time the table is set. This can be used with the NetRemote VariableWatch facility to trigger action when the table is updated.

noskip: Boolean. Normally, if a variable is already in the queue to be updated, a subsequent update will cause the earlier one to be skipped. Setting this parameter true prevents this behavior (for example to do simple animations).

filename: String. Path and filename of image file to download and store in NetRemote image variable.

url: String. Universal Resource Locator of image file to download and store in NetRemote image variable.

table: Table. Table specifying the values for multiple variables in NetRemote. For **SendStemmedTable** the keys must be numeric.

basevar: String. NetRemote variables prefixed basevar followed by an underscore and suffixed with the table key (which must be numeric) will be set to the table value. For example the NetRemote variable names might be myvar_1, myvar_2, myvar_3 etc.

© 2008 Promixis, LLC

mapping: Table of Table. Specifies which keys from the source table to send to NetRemote. The outer table keys match the keys in the source table. If a key is in the mapping table but not the source table, the relevant NetRemote variable is cleared. If a key is in the source table but not in the mapping table, it is ignored. The inner table has any of the following keys: **["label"]** = (String) Name of NetRemote variable (if not specified, the table key is used as the variable name); **["image"]** = (Boolean) If true, the value is taken as an image file name and that is sent; **["url"]** = (Boolean) If true, the value is taken as a URL of an image file on the internet which is downloaded and sent; **["func"]** = (Function) This function supplies the value which will be sent (a String return) based on two input parameters: the value from the source table and the entire source table; **["mapping"]** = (Table) This table becomes the mapping table for the value from the source table which must itself be a table.

prefix: String. If specified, is prefixed before the NetRemote variable name from the mapping table.

suffix: String. If specified, is suffixed after the NetRemote variable name from the mapping table.

pagegroup: String. The name of the group containing the target NetRemote page.

page: String. The name of the target NetRemote page.

luacode: String. The Lua "chunk" in text form which should be compiled and run in NetRemote.

Example:
```
require("NetRemote")
NetRemote.SetVariable("TV.Playing", "ABC One")
```

Example:
```
require("NetRemote")
NetRemote.JumpDevice("TV", "OnNow")
map = {"Channel" = {"label" = "Playing"}, "Prog" = {"label" = "Title"}, "Logo" =
    {"label" = "ChanLogo", "image" = true}}
val = {"Channel" = "ABC One", "Logo" = [[C:\Logos\ABC1.PNG]], "Prog" = "Ellen"}
NetRemote.SendMappedTable(val, map, "TV.")
```

NetRemote Label	Value after code executed
TV.Playing	ABC One
TV.Title	Ellen
TV.ChanLogo	Image from ABC1.PNG

Caching

Girder caches **SetVariable**, **SetImage** and **SetImageURL** parameters and only sends changed values. **SyncState** sends the entire cache contents again to ensure NetRemote is up to date.

```
NetRemote.SyncState()
```

Advanced Usage

The commands detailed above are adequate for most purposes. However they have two shortcomings:

1. If you have more than one NetRemote machine connected, they will send the same commands to all NetRemotes.
2. They are inefficient in complex scenarios where many commands need to be sent to the same NetRemote in sequence.

The library supports two additional forms of each of the commands (including **SyncState**) which address these problems.

© 2008 Promixis, LLC

To solve the first problem and target a specific NetRemote, use the command function prefixed "Client" and insert an additional parameter ahead of the normal ones. This additional parameter is the IP Address, Client Name or Instance GUID (see later) of the target NetRemote.

To solve both the first and second problems, use LookupClient to obtain a Client object and then call the command functions as methods of this object. This avoids the necessity for the library to do LookupClient for every command.

LookupClient
`[Client] = NetRemote.LookupClient([ident])`
`[Client]:Close()`
ident: String. DNS name, IP address or Instance GUID of the NetRemote client.

Example:

Assuming there is just one NetRemote connected and its address is 192.168.34.5, the following three alternative implementations are equivalent.

```
-- Simple (as before):
NetRemote.SetVariable("Var1", "Value1")
NetRemote.SetVariable("Var2", [[C:\Logos\ABC1.PNG]])
-- Client Targeted:
NetRemote.ClientSetVariable("192.168.34.5", "Var1", "Value1")
NetRemote.ClientSetImage("192.168.34.5", "Var2", [[C:\Logos\ABC1.PNG]])
-- Cached Client (note colons for method calls):
local Client = NetRemote.LookupClient("192.168.34.5")
if (Client) then
   Client:SetVariable("Var1", "Value1")
   Client:SetImage("Var2", [[C:\Logos\ABC1.PNG]])
    Client:Close()
end
```

Instance GUID

Normally, there is a maximum of one instance of NetRemote per machine. In this case, the specific NetRemote can be identified from the machine name or its IP address. However, multiple instances of NetRemote can be started on the same machine by specifying the command-line switch "/instance=*n*" where *n* is a unique instance number. NetRemote generates a long string called a GUID which is unique to the machine on which NetRemote is running and to the specified instance number. Girder uses this GUID to identify a specific NetRemote instance running anywhere on the network.

This GUID can be used with **LookupClient** or with any of the **Client...** functions to specify the client to be used. The GUID is the most efficient reference because if the IP address or Machine Name is used Girder has to search through the client table to find the GUID. If there are more than one clients running on the same machine, the GUID is the only way to distinguish between them.

To discover the GUID of a client instance, run the following code in a scripting action.

```
local c
for k, v in pairs(comserv.GetConnections()) do
    if v.Type == "NetRemote" then
        c = comserv.GetConnection(v.Socket)
       print(v.Hostname, v.Port, c:RemoteInstance())
    end
end
```

The GUID is the long hexadecimal string following the final space in the resulting report in the Lua Interactive Console.

© 2008 Promixis, LLC

OnClose

If a client is held open for any length of time there is the danger that the NetRemote goes offline. An OnClose method can be inserted into the Client table and it will be called when the connection is closed.

```
Client = NetRemote.LookupClient("192.168.34.5")
if (Client) then
    Client:OnClose = function() Client = nil end
end
```

GetConnectedClientList, EnumerateClients

```
[callback]([Client], ...)
NetRemote.EnumerateClients([Callback], ...)
[clients] = NetRemote.GetConnectedClientList()
```

Client: Object. One of the clients enumerated.

clients: Table of Strings. Numerically indexed table of strings being the Host Names of the connected NetRemote clients.

Client Tracking Callbacks

RegisterConnectCallback and **RegisterDisconnectCallback** register user supplied functions to be called when a new client appears or disappears.

```
[callback]()
[id] = RegisterConnectCallback([Callback])
[id] = RegisterDisconnectCallback([callback])
DeregisterConnectCallback([id])
DeregisterDisconnectCallback([id])
```

callback: Function. Function taking no parameters which is to be called when an event happens.

id: Number. Reference to callback enabling it to later be deregistered.

WantStatusMessages

```
NetRemote.WantStatusMessages([yesorno])
```

yesorno: Boolean. true to print status messages on the Lua Interactive Console.

Source: %GIRDER%luascript\NetRemote.lua.

12.21 os Library

Time and date

```
[secs] = os.clock()       -- Since program start
[secs] = os.time()        -- Current time
[secs] = os.time([tdtab]) -- From multi-part date/time
[secs] = os.difftime([secs2], [secs1]) -- secs2 - secs1
[ans] = os.date([format], [secs])
```

secs: (number) Number of seconds (since program start or an epoch).

tdtab: (table) Keys are "year" (four digits), "month" (1-12), "day" (1-31), "hour" (0-23), "min" (0-59), "sec" (0-61), "wday" (weekday, Sunday is 1), "yday" (day of the year), and "isdst" (daylight saving flag, a boolean).

format: (string) Starts with "!" for UTC. "*t" returns a tdtab. Else uses C strftime formatters, defaulting to "%c".

© 2008 Promixis, LLC

Set locale
`[name] = os.setlocale([name], [category])`
name: (string) The identifier of the locale to set. Returns **nil** if unavailable.

category: (string, optional) "all" (default), "collate", "ctype", "monetary", "numeric", or "time"

Run and stop processes
```
[value] = os.getenv([name])
[status] = os.execute([command])
os.exit([code])
```
name: (string) Name of a process environment variable.

value: (string) Value of the process environment variable or **nil** if it does not exist.

command: (string) Command to be executed by the OS shell.

status: (number) Status code returned by the OS shell.

code: (number, optional) Status code to be returned when the Girder process exits. Default is the "success" status code.

File system functions

Care is required with os.tmpname since it does not open the file, another process may grab it before it can be opened. Suggest io.tmpfile instead.
```
[res], [err] = os.remove([filename])
[res], [err] = os.rename([filename], [newname])
[filename] = os.tmpname()
```
filename: (string) The name and path of the file. As a return, may be **nil**.

newname: (string) The new name of the file.

Source: Lua I/O and OS Facilities Library

12.22 osd Library

Girder 4 introduces a new On Screen Display (OSD) architecture which enables pop-up windows to be created and rendered on your computer screen. These can show status text and graphics as well as images. Interactive events are also provided enabling the construction of user interfaces such as menus.

The OSD system has three distinct layers. The highest layer is very easy to use, but has limited flexibility. The lowest layer has almost complete flexibility at the cost of being much harder to use. The middle layer is intermediate in complexity and flexibility.

High Level Functions

1. StatusMessage Function provides a text message OSD in a wide variety of different styles.

2. Creating and Specifying Colors describes how to create custom colors and use named colors in all three layers.

Middle Layer Object Classes

3. Generic Style Keys and Methods describes styles and methods common to all the OSD classes.

4. Text OSD Class provides a simple text display OSD.

© 2008 Promixis, LLC

5. DigitalClock OSD Class provides a live digital clock OSD.

6. Image OSD Class is an OSD whose surface is defined by an image file.

7. Menu OSD Class provides a simple OSD menu which generates events.

8. AutoMenu OSD Class a more complex table driven OSD menu.

9. ProgressBar OSD Class provides an OSD with a horizontal progress bar.

10. Creating styles and colors.

Low Level Functions

11. Creating a New OSD Class shows how to create a custom OSD.

12. Low-level OSD Functions used to draw a custom OSD.

Source: Built-in.

12.22.1 StatusMessage Function

Creates and displays an OSD window which displays a string of text. StatusMessage creates a unique OSD which is re-used if called again whilst it is still on screen. In this case, style is ignored. StatusMessage2 always creates a new OSD ignoring any other that may be on screen.

```
[Osd] = osd.StatusMessage([text], [style])
[Osd] = osd.StatusMessage2([text], [style])
[style] = osd.GetStyle([template], [modifier])
```

text: String containing the text to be displayed.

style: Table containing the style settings (see below).

Osd: osd object (discard this return unless you want to manipulate the OSD later).

template: String key in the osd.Styles table - a style template name.

modifier: A table of style keys and values to be changed or added.

Simple Text OSD Using Default Style:

```
osd.StatusMessage("My Status\nSecond Line")
```

Using a Style Template:

```
osd.StatusMessage("My Status Message", osd.GetStyle("StatusDarkBlack"))
```

The available style templates can be viewed in the **Variable Inspector** (**View** menu). Expand the **osd** table and then the **Styles** table. The applicable styles are prefixed "Status".

Using a Modified Style Template:

```
osd.StatusMessage("My Status Message", osd.GetStyle("StatusLightBlue",
   {Position="NEARBOTTOMCENTERW", FontSize=20}))
```

See Text OSD Class for applicable style keys.

12.22.2 Creating and Specifying Colors

Colors are specified as 32-bit unsigned integers stored in Number variables. They can be created using the **MakeARGB** function.

© 2008 Promixis, LLC

```
[color] = osd.MakeARGB([alpha], [red], [green], [blue])
```
alpha: Number 0..255. Alpha channel (transparency) 255 is opaque.

red: Number 0..255. Red color component, 255 is full intensity.

green: Number 0..255. Green color component, 255 is full intensity.

blue: Number 0..255. Blue color component, 255 is full intensity.

Named Colors

A number of named, fully opaque colors are predefined in the **osd.Colors** table.

Examples:
```
local color1 = osd.Colors.Blue
local color2 = osd.MakeARGB(255, 0, 0, 255)
```

12.22.3 Generic Style Keys and Methods

Generic Style Keys

This is a list of the style keys that may be specified in the StatusMessage function or in New methods of all OSD Classes.

For how to specify values for "color" keys see Creating and Specifying Colors.

Monitor: Number (1..). 1 for primary monitor, 2.. for secondary monitors.

Position: String. TOP or CENTERH or NEARBOTTOM or BOTTOM concatenated with LEFT or CENTERW or RIGHT.

X, Y: Number. Pixel position on screen.

UseRegistryForPosition: Boolean. True to get window position from registry.

UpdatePositionInRegistry: Boolean. True to save window position in the registry.

RootKey: String. The root registry key, for example "HKEY_LOCAL_MACHINE".

KeyName: String. The registry key, for example "\SOFTWARE\Promixis\Girder\4\MyWindow".

PositionPrefix: String. Prefix for registry name, suffixed "-X", "-Y".

BGColor1, BGColor2: Number (Color).

BorderColor: Number (Color).

BorderSize: Number. Width of border in pixels.

Transparency: Number (0..255). 255 is fully opaque.

TransparencyColorKey: Number (Color) or nil. If specified, this color will be transparent.

AutoFontSizeWidth: Number or nil. If not nil, width of OSD is set to accommodate this number of characters.

AutoSizeWidth, AutoSizeHeight: Number (percentage). Size to percentage of screen size.

KeepOnDesktop: Boolean. True to ensure OSD remains visible on the desktop.

ShowOnUpdate: Boolean. True to automatically show the OSD on update if not showing.

© 2008 Promixis, LLC

CloseOnDoubleClick: Boolean. true if double-click should close OSD (early).

TopMost: Boolean. true to show over other windows.

Fade: Boolean. true if a fade effect should be used when the OSD closes.

TimeOut: Number - 0 for no timeout else number of milliseconds before auto-close.

Generic Methods

These methods are available for all OSD classes. In some cases **Show** is overridden by a version taking more parameters.

```
[Osd]:Show([forcetotop])
[Osd]:Hide()
[Osd]:ToggleVisible()
```

forcetotop: Boolean. True to force the OSD on top of all other windows.

12.22.4 Text OSD Class

This is the OSD class used by the StatusMessage function.

Example:

```
TextOSD = osd.Classes.Text:New(osd.GetStyle("StatusTransparentGreen", {Text = "Status"}))
TextOSD:Show(true, 1)
```

Style Keys

Text: String. The text to be shown.

Font: String. The name of the font to use.

FontSize: Number. Font size in points.

FontColor: Number (Color).

FontAlternateColor: Number (Color). If color = 2 in Update.

See also Generic Style Keys and Methods.

Methods

```
[Osd]:Show([forcetotop], [color])
[Osd]:Update([text], [color])  -- Replace the text in the OSD
```

forcetotop: Boolean. True to force OSD on top of other windows.

text: String. The text to show in the OSD.

color: Number (1..2). 1 = Use FontColor, 2 = Use FontAlternateColor.

See also Generic Style Keys and Methods.

12.22.5 DigitalClock OSD Class

This is based on the Text template and uses the same style templates. It provides a live digital clock display.

Example:

```
ClockOSD = osd.Classes.DigitalClock:New(osd.GetStyle("StatusTransparentGreen"))
ClockOSD:Show()
ClockOSD:Start()
```

Style Keys

© 2008 Promixis, LLC

For style options, see Text OSD Class.

Methods
```
[Osd]:Start() -- Start the clock.
```
See also Generic Style Keys and Methods.

12.22.6 Image OSD Class

This shows an image from a file in an OSD window.

Example:
```
ImageOSD = osd.Classes.Image:New({Fade=true})
ImageOSD:Show()
ImageOSD:Update([[C:\Documents and Settings\John\My Documents\My Pictures\OperaHouse.jpg]])
```
Style Keys

ImageFilename: String. The path and filename of the file to show.

See also Generic Style Keys and Methods.

Methods
```
[Osd]:Update([filename]) -- Change the file being shown
```
filename: The new path and filename to show.

See also Generic Style Keys and Methods.

12.22.7 Menu OSD Class

This shows an interactive selection menu and returns a selected item either via a callback function or via a Girder Event. The menu can be operated with the up and down arrow and return keys, using the mouse scroll wheel and click or point and click, or it may be linked to a remote control or other Girder event using the methods provided.

Example:
```
function ShowResult(itm, txt) print(itm, txt) end
MenuOSD = osd.Classes.Menu:New(osd.GetStyle("MenuDarkBlue",{Callback = ShowResult}))
MenuOSD:Show(true, "Test Menu", {"Item 1", "Item 2"})
```
Style Keys

Callback: Function taking up to two parameters. The first is the selected item number and the second is the item text.

Font: String. Font name.

FontColor, FontAlternateColor: Number (Color). Color of unselected and selected menu items.

FontTitleColor: Number (Color). Color of title text.

See also Generic Style Keys and Methods.

Methods
```
[Osd]:Show([forcetotop], [title], [items])
[Osd]:Up()  --Same as up-arrow key or scrollwheel up.
[Osd]:Down()  --Same as down-arrow key or scrollwheel down.
[Osd]:SelectCurrent() --Same as Enter/Return key.
[Osd]:Select([itemnumber]) --Same as mouse click on a specified item
```
itemnumber: Number. Index in the item table of the selected item.

callback: Function. Callback function specified in Callback style key.

© 2008 Promixis, LLC

forcetotop: Boolean. True to force the window to the top.

title: String to show as title of the menu.

items: Table of Strings being the itemtexts for the menu.

See also Generic Style Keys and Methods.

Girder Event

A Girder Event is raised only if a Callback function is not specified. It will be on the Girder device number (18) and has the eventstring [title].[itemtext], pld1 = itemtext, pld2 = itemnumber.

12.22.8 AutoMenu OSD Class

This class builds on the Menu OSD Class adding support for sub-menus and extended callback data tables.

The menu is defined by a table. There must be a key "Title" with a string value containing the title which is shown at the top of the menu. The table also contains numerically indexed entries one per menu item. Each entry may be one of the following.

- A "simple item" which is a string. This string becomes the text of the menu item. If an AutoCallback function is provided, the string will be passed to it when the item is clicked. If not, an event is generated (see later).
- An "extended item" which is a table with a "Name" key but NOT a "Title" key. The "Name" key becomes the text of the menu item. The table may contain any further keys required by the callback. This table is passed to the AutoCallback function when the item is clicked.
- A "sub-menu" which is a table with a "Title" key. The "Title" key becomes the text of the menu item and when the item is clicked, the menu is replaced by this sub-menu.

Example:

```
local MyMenu = {
   Title='Toplevel Menu',
   [1]='Simple Item',
   [2]= {
       Name='Extended Item',
       Something=200,
   },
   [3]= {
       Title='Submenu',
       [1]='SubMenu Item 1',
       [2]='SubMenu Item 2',
   },
}
table.insert(MyMenu[3], MyMenu)
local function MyAutoCB(Number, Text, Item)
   if type(Item) == 'string' then
       print("[Simple Item]", Number, Text)
   elseif type(Item) == 'table' then
       if not Item.Title then
          print("[Extended Item]", Number, Text); table.print(Item)
       else
          print("[Sub-menu]", Number, Text)
       end
    end
end
if not AutoMenu then
   AutoMenu = osd.Classes.AutoMenu:New({AutoCallback=MyAutoCB, Fade=true, TimeOut=0})
```

© 2008 Promixis, LLC

```
end
AutoMenu:Show(true, MyMenu)
```
Note that there should generally be one and only one AutoMenu and this is enforced here by defining it as a global and re-using it as required. The table.insert function creates a third item in the submenu containing a reference to the original table. The effect of this is to provide a return link to the main menu.

Style Keys

AutoCallback: Function taking up to three parameters. If not provided, an Event is generated when a menu item is selected. Otherwise this function is called with the item number, the item text and the item.

Font: String. Font name.

FontColor, FontAlternateColor: Number (Color). Color of unselected and selected menu items.

FontTitleColor: Number (Color). Color of title text.

See also Generic Style Keys and Methods.

Methods
```
[autocallback]([itemno], [itemtext], [item])
[Osd] = osd.Classes.AutoMenu:New({AutoCallback = [autocallback]})
[Osd]:Show([forcetotop], [menutable])
[Osd]:Up()  --Same as up-arrow key or scrollwheel up.
[Osd]:Down()  --Same as down-arrow key or scrollwheel down.
[Osd]:SelectCurrent() --Same as Enter/Return key.
[Osd]:Select([itemno]) --Same as mouse click on a specified item.
```
itemno: Number. The index number of the clicked item on the menu.

itemtext: String. The text of the clicked item on the menu.

item: String or Table. The text of the clicked item, or the extended table representing the item.

forcetotop: Boolean. True to force the menu window on top of other windows.

menutable: Table. The table which defines the menu.

See also Generic Style Keys and Methods.

Girder Event

A Girder Event is raised only if an AutoCallback function is not specified. It will be on the Girder device number (18) and has the eventstring [title].[itemtext], pld1 = itemtext, pld2 = itemnumber.

12.22.9 ProgressBar OSD Class

Shows a progress OSD with text percentage and graphical bar.

Example:
```
ProgOSD = osd.Classes.Progress:New(osd.GetStyle("ProgressMiniDarkBlue",
    {Caption = "Progress", Min = 0, Max = 100}))
ProgOSD:Show(true, 20, 1)
```

Style Keys

Caption: String shown before the percentage value in the progress bar.

© 2008 Promixis, LLC

Min: Number the minimum value of progress which will show as 0%.

Max: Number. The maximum value of progress which will show as 100%.

FGColor1, FGColor2, FGColor3, FGColor4: Number (color). 1 and 2 are used for the bar gradient fill when color = 1; 3 and 4 when color = 2.

Font: String. The name of the font.

FontColor, FontAlternateColor: Number (Color). The color of the text.

FontSize: Number. The size of the font

See also Generic Style Keys and Methods.

Methods

```
[Osd]:Show([forcetotop], [pos], [color])
[Osd]:Update([pos], [color])
```

forcetotop: Boolean. True to bring the window to the top.

pos: Number - the current progress value.

color: Number (1..2). Selects one of two preset colors for the progress bar.

See also Generic Style Keys and Methods.

12.22.1 Creating styles and colors

Named colors can be added to the available set and styles can be created for use in osd. GetStyle for existing or custom OSD classes.

```
osd.Colors.LightGray = osd.MakeARGB(255, 200, 200, 200)
osd.Styles.StatusSemiTrans = osd.GetStyle("StatusOrange", {Transparency = 200})
```

12.22.1 Creating a New OSD Class

The preferred method of creating a custom OSD is to create a new class which inherits from osd.Classes.Base, or from another class in the osd.Classes table. This provides a surface on which to draw the visual presentation using the Low-level OSD Functions and an event-handling infrastructure.

The template looks like this.

```
osd.Classes.MyOSD = osd.Classes.Base:New ({
   Initialize = function(self)
      osd.Classes.Base.Initialize(self)
       if not self:ChecknLock () then return false end
       self.Width = 200; self.Height = 40
       self:CalculatePosition()
      self.OSD:Size(self.Width, self.Height)
      self:SetPosition(self.X, self.Y)
       -- Any initialization needed when OSD is created, but not
       -- when it is updated goes here.
       self.mutex:unlock()
   end,
   Show = function(self) -- Add additional parameters as required
       if not self.OSD then self:Initialize () end
       if not self:ChecknLock() then return false end
       self.OSD:WorkOnBuffer(true)
       self.OSD:Clear (osd.Colors.Black)
       -- Draw the OSD using self.OSD with the low-level OSD Functions
```

© 2008 Promixis, LLC

```
        -- Style keys are accessed e.g. self.X
        self.OSD:WorkOnBuffer(false)
        self.OSD:CopyBufferToScreen()
        osd.Classes.Base.Show(self, true)
        self.mutex:unlock()
    end,
    Update = function(self) -- Add additional parameters as required
        if not self:ChecknLock () then return false end
        -- Change any style keys used to draw.
        -- Base calls Show to do the redrawing automatically.
        osd.Classes.Base.Update(self)
        self.mutex:unlock()
    end,
    Quit = function (self)
        if not self:ChecknLock () then return false end
        -- Any tidy-up code when OSD is finished with goes here
        self.mutex:unlock()
        osd.Classes.Base.Quit (self)
    end,
})
```

The new OSD can now be shown using the same techniques as for one of the built in classes.

```
MyOSD = osd.Classes.MyOSD:New(osd.GetStyle("StatusOrange",
   {Position = "NEARBOTTOMCENTERW"}))
MyOSD:Show()
```

This shows an empty black window centered on the screen near the bottom. It will disappear if clicked on or after a default timeout.

The following Base functions can be overridden to alter the behavior of the window.

```
OnLeftButtonDown(x, y, keys)
OnLeftButtonUp()
OnLeftButtonDblClick()
OnRightButtonDown(x, y)
OnMouseMove(x, y)
OnMouseLeave()
```

For example, to override OnRightButtonDown.

```
   OnRightButtonDown = function(self, x, y)
       if not self:ChecknLock () then return false end
       -- Process the event
       self.mutex:unlock()
   end,
```

12.22.11 Low-level OSD Functions

These functions are more or less direct mappings of Microsoft's GDI+. More information can be obtained from the MSDN web site at http://msdn.microsoft.com/library/.

Object factory functions

```
[Osd], [error] = osd.CreateOSD([topmost])
[Image], [error] = osd.CreateImage([filename])
[Brush] = osd.CreateSolidBrush([color])
[Brush] = osd.CreateLinearGradientBrush([x1], [y1], [x2], [y2], [color1], [color2])
[Brush] = osd.CreateTexturedBrush([Image], [WRAPMODE], [x1], [y1], [x2], [y2])
[Brush] = osd.CreateHatchBrush([HATCHSTYLE], [color1], [color2])
[Pen] = osd.CreatePen([Brush], [width])
[Path] = osd.CreateGraphicsPath([FILLMODE])
[Matrix] = osd.CreateMatrix()
```

Osd object methods

```
[Osd]:Destroy()
```

© 2008 Promixis, LLC

```
[Osd]:Show()
[Osd]:Hide()
[Osd]:SetTranparancy([colorkey], [transparancy])
[Osd]:Position([x], [y])
[Osd]:Size([width], [height])
[Osd]:Topmost([bool])
[Osd]:Line([x1], [y1], [x2], [y2], [Pen], [Path], [Matrix])
[width], [height] = [Osd]:MeasureString([text], [fontname], [fontsize])
[width], [height] = [Osd]:MeasureStringFull([text],[fontname],[fontsize],[boxwidth],
   [boxheight], [STRINGALIGNMENThorizonal], [STRINGALIGNMENTvertical])
-- In the following [Path] defines a clipping region and [Matrix] a transformation
-- Matrix. Each is optional.
[Osd]:DrawString([text], [fontname], [fontsize], [x], [y], [Brush], [Path], [Matrix])
[Osd]:DrawStringFull([text], [fontname], [fontsize], [x], [y],[boxwidth],[boxheight],
  [STRINGALIGNMENThorizonal],[STRINGALIGNMENTvertical],[Brush],[Path],[Matrix])
[Osd]:DrawImage([Image], [x], [y], [Path], [Matrix])
[Osd]:ScaleDrawImage([Image], [x], [y], [width], [height], [Path], [Matrix])
[Osd]:DrawImageFull([Image], [sourcex], [sourcey], [sourcew], [sourceh], [destx],
   [desty], [destw], [desth], [color1], [color2], [Path], [Matrix])
[Osd]:DrawEllipse([x], [y], [width], [height], [Pen], [Path], [Matrix])
[Osd]:DrawRectangle([x], [y], [width], [height], [Pen], [Path], [Matrix])
[Osd]:DrawPie([x], [y], [width], [height], [startangle], [sweepangle], [Pen],
   [Path],[Matrix])
[Osd]:DrawPath([Path], [Pen], [Path], [Matrix])
[Osd]:FillRectangle([x], [y], [width], [height], [Brush], [Path], [Matrix])
[Osd]:FillEllipse([x], [y], [width], [height], [Brush], [Path], [Matrix])
[Osd]:FillPie([x], [y], [width], [height], [startangle], [sweepangle], [Brush],
   [Path],[Matrix])
[Osd]:OnEvent([ON], [eventfunction])
[Osd]:WorkOnBuffer([bool])
[Osd]:CopyBufferToScreen()
[Osd]:SetSmoothingMode([SMOOTHINGMODE])
[Osd]:SetInterpolationMode([INTERPOLATIONMODE])
[Osd]:SetRegion([Path]) -- Makes OSD a shaped Window
[Osd]:Clear([color])
[hwnd] = [Osd]:GetHandle()
```

Prototypes for [eventfunction] in OnEvent

```
OnMouseMove([x], [y])
OnMouseLeave()
OnMouseButton([x], [y], [Key]) -- OnLeftButtonDown, OnRightButtonDown,
   -- OnMiddleButtonDown, OnLeftButtonUp, OnRightButtonUp, OnMiddleButtonUp,
   -- OnLeftButtonDblClick, OnRightButtonDblClick, OnMiddleButtonDblClick
OnPaint()
OnShowWindow(showflag, STATUS)
OnDisplayChange([x], [y], [colordepth])
OnKey([keycode], [keydata]) -- OnKeyDown, OnKeyUp, OnChar
```

Image object methods

```
[w], [h] = [Image]:Size()
[animationflag] = [Image]:Animated()
[Image]:Animate()
[Image]:Pause([bool])
[Image]:Destroy()
```

Pen object methods

```
[Pen]:SetDashStyle([DASHSTYLE])
[Pen]:SetDashCap([DASHCAP])
[Pen]:SetStartCap([LINECAP])
[Pen]:SetEndCap([LINECAP])
```

© 2008 Promixis, LLC

```
[Pen]:Destroy()
```
Brush object method
```
[Brush]:Destroy()
```
Path object methods
```
[Path]:AddRectangle([x], [y], [width], [height])
[Path]:AddEllipse([x], [y], [width], [height])
[Path]:AddArc([x], [y], [width], [height], [startangle], [sweepangle])
[Path]:AddBezier([x1], [y1], [x2], [y2], [x3], [y3], [x4], [y4])
-- "tension" is a positive real number. 0 is equivalent to AddLines or AddPolygon,
-- increasing numbers increases the curve between the points.
[Path]:AddClosedCurve([Points], [tension])
[Path]:AddCurve([Points], [tension])
[Path]:AddLine([x1], [y1], [x2], [y2])
[Path]:AddLines([Points])
[Path]:AddPie([x], [y], [width], [height], [startangle], [sweepangle])
[Path]:AddPolygon([Points], [tension])
[Path]:AddString([text], [fontname], [fontsize], [fontstyle], [x], [y],
   [width], [height], [STRINGALIGNMENThorizonal], [STRINGALIGNMENTvertical])
[Path]:Outline()
[Path]:Reset()
[Path]:StartFigure()
[Path]:CloseFigure()
[Path]:CloseAllFigures()
[Path]:Transform([Matrix])
```
Matrix object methods
```
[Matrix]:Rotate([angle], [order])
[Matrix]:RotateAt([angle], [x], [y], [order])
[m11], [m12], [m21], [m22], [dx], [dy] = [Matrix]:GetElements()
[Matrix]:SetElements([m11], [m12], [m21], [m22], [dx], [dy])
[Matrix]:Multiply([Matrix])
[Matrix]:Scale([x], [y], [order])
[Matrix]:Translate([x], [y], [order])
[Matrix]:Shear([x], [y], [order])
```
Points table format

The Points array is a table of tables. The inner table has two number elements indexed by strings ["x"] and ["y"]. The outer table is indexed by numbers 1..n. This represents an list of pixel coordinate pairs. Example: pts = {{x = 4, y = 2},{x = 5, y = 1}}.

Notes

FillMode.Alternate: Specifies that areas are filled according to the even-odd parity rule. According to this rule, you can determine whether a test point is inside or outside a closed curve as follows: Draw a line from the test point to a point that is distant from the curve. If that line crosses the curve an odd number of times, the test point is inside the curve; otherwise, the test point is outside the curve.

FillMode.Winding: Specifies that areas are filled according to the nonzero winding rule. According to this rule, you can determine whether a test point is inside or outside a closed curve as follows: Draw a line from a test point to a point that is distant from the curve. Count the number of times the curve crosses the test line from left to right, and count the number of times the curve crosses the test line from right to left. If those two numbers are the same, the test point is outside the curve; otherwise, the test point is inside the curve.

12.23 ptable Library

The **ptable** provides an easy way to store persistent settings between Girder sessions. A **ptable** object behaves like an ordinary table except that each time it is created it retains the

© 2008 Promixis, LLC

same keys and values it had the last time it was used. It automatically uses the Windows Registry to store its keys and values.

Creating a ptable

ptable:New creates a **ptable** object. This can be assigned to a local, global or table variable like a real table, but it cannot have an initializer.

```
require('ptable')
local myptable = ptable:New("mysettings")
myptable.setting1 = mytable.setting1 or "string value"
myptable.setting2 = mytable.setting2 or 12.45
if mytable.setting3 == nil then mytable.setting3 = false end
```

The **name** parameter ("mysettings" in the above example) must be unique. If two **ptable** objects are created with the same name, they will use the same registry storage. The example shows how to establish default values of string, number and boolean type.

Reading and Writing values

ptable objects can be read and written using normal table access methods with some restrictions.

- Keys must be of string type and two characters or more in length.
- Values must be of string, number or boolean type. There is also limited support for table values and keys may be set to the value **nil** to delete them from the registry.
- Table values may be read and written in the usual way, but changes to values within these tables will not be automatically persisted. Such values can be re-persisted by assigning **nil** to the **ptable** key then assigning the altered table to it.
- **ptables** cannot be iterated in a generic **for** structure using **pairs** or **ipairs**. The special iterator **ptable:pairs** must be used instead.

Examples

```
require('ptable')
local mytable = ptable:New("mysettings")
local x = mytable.setting1
mytable.setting1 = "Test"
local t = {monday, tuesday, wednesday}
mytable.days = t
print(mytable.days[1])
t = mytable.days
t[4] = "thursday"
mytable.days = nil
mytable.days = t
for i, v in ptable:pairs(mytable) do
    print(i .. " = " .. v)
end
```

Advanced Features

Normally **ptables** persist to the standard Girder registry key for plugins ("SOFTWARE\Promixis\Girder\4\Plugins" in the HKLM hive). However the **New** method has optional parameters to enable any registry location to be used. The **Close** method allows the registry key to be closed after it has been opened by **New** or by a write operation. This is generally advisable if there will be no further writes for some time (it is reopened automatically when needed). The **Remove** method deletes the **ptable** from the registry.

```
[Ptable] = ptable:New([name], [stem], [hive])
[Ptable]:Close()
[Ptable]:Remove()
```

name: String. The name of the registry key which will be used to persist the table. This will be prefixed with a backslash and appended to stem.

stem: String (optional, default = [[SOFTWARE\Promixis\Girder\4\Plugins]]). The location in the registry for the registry key which persists the table.

© 2008 Promixis, LLC

hive: String (optional, default = "HKLM"). The registry hive = "HKLM", "HKCU", "HKCR", "HKU" or "HKCC".

Source: %GIRDER%luascript\ptable.lua.

12.24 scheduler Library

Requires Girder Pro.

Creating a Scheduler

The scheduler library has a function Create which is an object factory returning a scheduler object. The scheduler object generates the specified event repeatedly on a schedule specified by one or more **tasks** (see next section).
`[Sched] = scheduler.Create([eventstring], [devicenumber], [keep])`
eventstring: String. The event string of the event to be generated.

devicenumber: Number. The device number of the event to be generated. Suggest 235, the scheduler plugin device number.

keep: Boolean, Optional default false. Set this true to make the scheduler independent of the Lua object variable (i.e. it will be kept in memory even when Sched is discarded).

Task Creation Methods

You can add multiple tasks of the same or different type to define schedules of arbitrary complexity.

```
[id],[err]=[Sched]:MinuteTask([every],[repeat],[begin],[end])
[id],[err]=[Sched]:HourTask([every],[minute],[repeat],[begin],[end])
[id],[err]=[Sched]:DayTask([every],[hour],[minute],[repeat],[begin],[end])
[id],[err]=[Sched]:DayOfWeekTask([every],[dayofweek],[hour],[minute],
   [repeat],[begin],[end])
[id],[err]=[Sched]:DayOfMonthTask([every],[dayofmonth],[hour],[minute],
   [repeat],[begin],[end])
[id],[err]=[Sched]:SunsetTask([every],[latitude],[longitude],[repeats],
   [begin],[end],[offset])
[id],[err]=[Sched]:SunriseTask([every],[latitude],[longitude],[repeats],
   [begin],[end],[offset])
```

id: Number. Identifier for the task, nil on error.

err: String. Description of error.

every: Number. 1 = every period, 2 = every second period etc.

minute: Number (0..59). Minute of the hour at which the event happens.

hour: Number (1..23). Hour of the day at which the event happens.

dayofweek: Number (0..6, 0 = Sunday). Day of the week on which the event occurs.

dayofmonth: Number (1..32). Day of the month on which the event occurs.

repeat: Number (1..scheduler.INFINITE). Number of times the event occurs.

begin: Table or nil = now. Date and time after which events may occur. Keys ["Day"] (of month), ["Month"], ["Year"], ["Hour"] and ["Minute"].

© 2008 Promixis, LLC

end: Table or nil = never. Date and time after which events may no longer occur. Keys as for begin.

latitude: Number. Degrees and decimals of a degree. Negative for south of equator. You can get this from the settings page using gir.GetLocation().

longitude: Number. Degrees and decimals of a degree. Negative for west of meridian. You can get this from the settings page using gir.GetLocation().

offset: Number or nil = 0. Minutes offset from the sunset or sunrise time. For example -60 schedules an hour before sunset or sunrise.

Manipulating a schedule

Clear removes all tasks from the schedule. **GetTasks** returns a table with an entry for each task being a table containing the parameters of the task. **ListProperties** is deprecated, it returns a similar but less well structured table.

```
[res],[err] = [Sched]:Start()
[res],[err] = [Sched]:Stop()
[running] = [Sched]:IsRunning()
[keep] = [Sched]:KeepBeyondLua()
[eventstring] = [Sched]:GetEventString()
[device] = [Sched]:GetDevice()
[uuid] = [Sched]:GetUUID()
[uuid] = [Sched]:SetUUID([uuid])
[res],[err] = [Sched]:RemoveTask([id])
[res],[err] = [Sched]:Clear()
[tasklist] = [Sched]:GetTasks()
[task] = [Sched]:GetTask([id])
[tasklist],[err] = [Sched]:ListProperties()
[Sched]:Destroy()
```

running: Boolean. true if the schedule has been started, false if it has been stopped.

keep: Boolean. false if the scheduler is bound to a Lua variable, true if it is independent.

eventstring: String. The eventstring that the scheduler generates.

device: Number. The device number on which the eventstring is generated.

uuid: String. This is a string storage in the Scheduler object intended for storing a Universally Unique IDentifier string for persistence control. UUID strings can be generated using win. GetUUID().

id: Number. Task ID as returned by task creation method (this is also the index of the tasklist table).

tasklist: Table of task tables indexed by task id.

task: Table. "Type" field has numeric value 1 = HourTask; 2 = DayTask; 3 = DayOfWeekTask; 4 = DayOfMonthTask; 5 = SunsetTask; 6 = SunriseTask; 7 = MinuteTask. "Active" field is a boolean determining if that task will generate an event in the future. If a task is active, there will be a "NextTrigger" field showing time and date in string form. The table also has a field for each parameter of that task type, named as above, but with the initial letter of each word capitalized.

Example:

```
local sched = scheduler.Create("schedule 1", 235, true)
sched:HourTask(1, 45, scheduler.INFINITE)
sched:HourTask(1, 0)
```

© 2008 Promixis, LLC

```
sched:SunsetTask(1, 51.0, 0.0)
table.print(sched:GetTasks())
sched:Start()
```

Recovering schedules

GetSchedulers returns a list of all scheduler objects in the system. **GetScheduler** recovers a scheduler object specified by ID. **GetID** returns the ID of the scheduler object.

```
[shedlist] = scheduler.GetSchedulers()
[Sched] = scheduler.GetScheduler(id)
[id] = [Sched]:GetID()
```

schedlist: Table. The table is indexed by id and contains sub-tables with keys "EventString" and "Device". The id keys can be used with GetScheduler to recover the scheduler object.

id: Number. ID number identifying the Scheduler object.

Sched: Object. The scheduler object.

SunRise, SunSet

Gets the sunrise and sunset times for a given date and position.

```
[time], [err] = scheduler.SunRise([date], [latitude], [longitude])
[time], [err] = scheduler.SunSet([date], [latitude], [longitude])
```

date: Table or nil = today. Table has keys ["Day"] (of month), ["Month"], ["Year"], all numbers.

latitude: Number. Degrees and decimals of a degree. Negative for south of equator.

longitude: Number. Degrees and decimals of a degree. Negative for west of meridian.

time: Table or nil = no sunrise/sunset (polar regions). Keys are ["Hour"], ["Minute"], ["Second"] plus ["Day"], ["Month"] and ["Year"] echoed from input.

Tip: You can get latitude and longitude from the settings dialog using **gir.GetLocation**.

Scheduler Persistence and Import

The scheduler user interface (see Scheduler in the Home Automation Applications section) automatically saves and restores schedulers created by the user on the **Lua: Scheduler** Event Device. Schedulers are saved in XML files, one scheduler per file in the **schedulers** sub-directory of the Girder installation directory. However the XML format and import method is capable of loading multiple schedulers from a single file, with any Event Device.

To produce an example file to discover the XML format, create a scheduler in the user interface and attach one task of each type. Press "Apply" and then examine the new file in the **schedulers** sub-directory.

When Girder starts (and on a script reset) all files in the **schedulers** sub-directory with the extension **.GSH** are parsed and used to create schedulers if possible. Additionally, any and all **.LUA** files are loaded and executed. If these return a string, Girder attempts to treat it as a filename and load XML contained within. Failing this, Girder attempts to parse the string itself as XML. These features can be used to convert and import schedulers from other applications such as Microsoft Outlook.

When Girder saves a scheduler to XML it assigns the same UUID to the version in memory and in XML. When it loads a scheduler from XML, it overwrites any existing scheduler in memory with the same UUID to avoid duplication. You can use this mechanism when creating XML for import by writing a UUID property in the XML. If you do not write this property, importing the same file twice will result in two copies of the same scheduler in memory.

Two Lua functions are provided for importing and exporting schedulers to XML.

© 2008 Promixis, LLC

```
scheduler.SaveToXML([sched], [file])
scheduler.LoadFromXML([xml])
```
sched: Object or Number. Scheduler object or scheduler ID number.

file: String. Path and filename to store the XML in. This can be omitted in which case the **schedulers** sub directory will be used and the file name will be the UUID of the scheduler (which will be generated if it does not already exist).

xml: String. This can either contain the XML defining the scheduler directly or it can contain the file path and name of the file containing the XML.

Source: Scheduler Plugin

12.25 serial Library

Requires Girder Pro.

Note this functionality is implemented by the transport plugin

The serial Library includes objects and methods for communicating over PC serial ports. It also includes classes for writing Device Drivers. These features are described in the Generic Serial Plugin topic. This section describes the low-level features which support direct access or for use within Device Drivers.

Open port and obtain Serial Object
```
[Port], [err] = serial.Open([comport])
```
comport: Number. The comport number to use.

Settings
```
[res],[err] = [Port]:Baud([baudrate])
[res],[err] = [Port]:Flow([flowtype])
[res],[err] = [Port]:Parms([parity], [stopbits], [databits])
[res],[err] = [Port]:BurstSize([burstsize])
[Port]:ICD([delay])
```
baudrate: Number. Bits per second (for example 9600).

flowtype: String. "N" - No flow control; "H" - Hardware flow control; "S" - Software flow control.

parity: Number. 0 - None; 1 - Odd; 2 - Even; 3 - Mark; 4 - Space.

stopbits: Number. 0 - One; 1 - One and a half; 2 - Two.

databits: Number (5, 6, 7 or 8). Number of bits of each byte transmitted. Most significant bits are ignored (For example, ASCII can be transmitted in 7 bits).

burstsize: Number. Maximum number of bytes transmitted in one go. Default is 256.

delay: Number (milliseconds). Time inserted between bytes. Default is 0.

Reading and Writing

Reading can be polled using the Read method or asynchronous using a Callback.

```
[bytes], [err] = [Port]:Write([data])
[data], [bytes] = [Port]:Read([bytes])
[res] = [Port]:Callback(serial.CB_DISABLE)
[callback]([data], [code])
[res] = [Port]:Callback(serial.CB_FIXEDLENGTH, [bytes], [icto], [callback])
```

© 2008 Promixis, LLC

```
[res] = [Port]:Callback(serial.CB_TERMINATED, [terminator], [timeout], [callback])
[res] = [Port]:Callback(serial.CB_MARKEDLENGTH1,[marker], [trailer], [timeout],
    [callback])
[res], [err] = Event([mask], [timeout])
[res], [err] = CancelEvent()
```

data: String. Data read or to write.

bytes: Number. Maximum bytes to read, or bytes written or read.

terminator: String. Callback is triggered when these bytes have been received.

timeout: Number (milliseconds). Maximum time to wait for callback conditions to be met. Use the serial.INFINITE constant to disable timeout.

marker: String. Single byte that prefixes the count byte.

trailer: Number. Number of bytes that suffix the message after the counted payload.

mask and code: Number. These are bitwise OR combinations of the constants in the following table. The mask determines which of the conditions will generate callbacks. The code determines which of the conditions occurred to cause the callback. Note that more than one condition may have occurred so you need to bitwise AND the code with each enabled flag and test for non-zero.

Mask/Code	Comments
RXCHAR	One or more characters received
TXEMPTY	Transmission complete (transmit buffer empty)
CTS	Clear To Send input changed state
DSR	Data Set Ready input changed state
RLSD	Receive Line Signal Detect (DCD) input changed state
BREAK	Break status of receive line changed
ERR	A reception error occurred (parity, framing etc.)
RING	The ring detection input was asserted.
INCOMPLETERESPONSETIMEOUT	Inter character period exceeded

Line Control

Set and Get the status of the RS232 control signals.

```
[res], [err] = [Port]:SetDTR([on])
[on], [err] = [Port]:GetDTR()
[res], [err] = [Port]:SetRTS([on])
[on], [err] = [Port]:GetRTS()
[on], [err] = [Port]:GetDCD()
[on], [err] = [Port]:GetDSR()
[on], [err] = [Port]:GetRI()
[on], [err] = [Port]:SetBreakBit([op])
[status], [err] = [Port]:Status()
```

on: Boolean. True for signal asserted, false for not asserted.

op: String. "A" - assert; "C" - cancel; "D" - detect.

status: Number. Bitmap OR combination of any of the following: 1 = Receive Buffer overrun; 2 = Receive UART overrun; 4 = Receive Parity Error; 8 = Receive Framing Error; 16 = Receive Break; 256 = Transmission Buffer Full.

© 2008 Promixis, LLC

Reset, Close

```
[res], [err] = [Port]:RxClear()
[res], [err] = [Port]:TxClear()
[res], [err] = [Port]:Close()
```

Utilities

```
[formatted] = serial.formatbytes([string])
serial.printbytes([string])
```

string: String. String presumed to be a mix of ASCII printable and control characters.

formatted: String. Printable string with hex pairs followed by printable input characters.

```
[dec] = serial.hextodecimal([hex])
[hex] = serial.decimaltohex([dec])
[byte] = serial.decimaltobyte([dec])
[dec] = serial.bytetodecimal([byte])
[hex] = serial.bytetohex([bytes])
[byte] = serial.hextobyte([hex])
[hex] = serial.binarytohex([bin])
[bin] = serial.decimaltobinary([dec], [n])
```

dec: Number. Decimal form number.

hex: String. Hexidecimal digits 0..9 A..F or a..f, max 8 digits.

bytes: String. String in which each character may be any byte code.

byte: String. Single character string.

bin: String. Binary string comprising digits 0 and 1.

n: Number, default=8. Number of digits in binary string.

```
[string] = serial.translateescapesequences([excaped], [escapechar])
```

string: String. String with control codes expressed as single bytes.

escaped: String. String with control codes expressed as escapechar followed by two hex digits.

escapechar: String, default='/'. Single character used as escape character.

```
[dec] = serial.bextract([dec],[s],[e])
[bits] = serial.bcount([dec])
[dec] = serial.zerobits([dec], [mask])
```

dec: Number. A positive whole number, 32 bits maximum.

s: Number. Starting bit number 1..32.

e: Number. Ending bit number 1..32 >= s.

bits: Number. Number of bits required to represent the number.

mask: Number. Up to 32 bits, 1 at bit positions to be cleared.

Source: Generic Serial Plugin

© 2008 Promixis, LLC

12.26 socket Library

The LuaSocket library provides low-level Internet Protocol access and protocol support for DNS, FTP, HTTP, LTN12, MIME, SMTP, TCP and UDP along with some helper functions for building URLs. The library is not installed by default. Any script which uses it should have a require statement.

```
require("socket")       -- Includes TCP, UDP and DNS
require("socket.ftp")
require("socket.http")
require("socket.smtp")
```

For detailed documentation, refer to this website:

http://www.cs.princeton.edu/~diego/professional/luasocket/reference.html.

Source: %GIRDER%\luascript\.

12.27 SQL Database

Girder Pro Only

Girder Provides database access through ODBC and Sqlite3 using LuaSQL.

Example

```
require  'luasqlsqlite3';
local env = luasql.sqlite3();
local con = assert( env:connect("my_demo_db"))

res = con:execute"DROP TABLE people"
res = assert (con:execute[[
   CREATE TABLE people(
     name  varchar(50),
     email varchar(50)
  )
]])
-- add a few elements
list = {
   { name="Jose das Couves", email="jose@couves.com", },
   { name="Manoel Joaquim", email="manoel.joaquim@cafundo.com", },
   { name="Maria das Dores", email="maria@dores.com", },
}
for i, p in pairs (list) do
   res = assert (con:execute(string.format([[
     INSERT INTO people
     VALUES ('%s', '%s')]], p.name, p.email)
  ))

end
-- retrieve a cursor
cur = assert (con:execute"SELECT name, email from people")
-- print all rows, the rows will be indexed by field names

row = cur:fetch ({}, "a")
while row do
   print(string.format("Name: %s, E-mail: %s", row.name, row.email))
   -- reusing the table of results
   row = cur:fetch (row, "a")
```

© 2008 Promixis, LLC

```
end
-- close everything

cur:close()
con:close()
env:close()
```

12.28 string Library

The string table is a standard Lua library which is documented in the Lua manuals. Girder adds the some functions to the standard set.

Indexing strings

The first character in a Lua string is at index 1 (not at 0). The last character is at index string.len(s). Negative indices may be used to index the string from the end, -1 being the last character and -string.len(s) being the first.

byte returns the code of the character at an index. Lua can store any byte including zero in a string which may thus be used as an efficient byte array. **sub** returns a substring from a string specified by a starting and ending index.

```
[length] = string.len([string])
[code] = string.byte([string], [index])
[string] = string.sub([string], [start], [end])
```

length: (number) Length of the string in bytes.

code: (number) Byte code at a particular index.

start: (number) Index of first character to be extracted.

end: (number, optional) Index of last character. Defaults to -1, the last character in the string.

Creating strings

format creates a string by substituting any number of Lua values into a format string using format specifiers. **char** forms a string from a list of bytes. **rep** forms a string by repeating a source string (often a single character) a given number of times. **dump** creates a string containing the byte codes of a Lua function.

```
[string] = string.format([format], [...])
[string] = string.char([byte] ...)
[string] = string.rep([string], [copies])
[string] = string.dump([function])
```

format (string): A string consisting of literal characters, escape sequences (see Lexical Conventions), and format specifiers (see below). Remaining parameters are formatted and inserted in place of the format specifiers (in order).

byte: (number) An integer 0-255 forms a single character in the string.

copies: (number) The number of copies of the source string forming the output string.

function: (function) The Lua function to be dumped, which must not have upvalues.

Format Specifiers

The general form of a format token is **%s** where *s* is a letter shown in the table below. Other possibilities are **%2s** where the number is the minimum number of characters (padded with spaces as necessary). **%02d** the zero character is fixed and the number is the minimum number of characters, padded with leading zeros as necessary. **%2.4d** the second number is a precision specifier which specifies the maximum number of characters (strings are truncated, numbers are reduced in precision). **%%** inserts a single **%** in the output.

© 2008 Promixis, LLC

Spec	Meaning	Spec	Meaning
%c	Number as ASCII character	%d	Number as Decimal
%E	Number in Exponent notation	%e	Number in Exponent notation
%f	Number as floating point	%g	Number in most compact form
%G	Number in most compact form	%i	Number as signed integer
%o	Number in Octal	%u	Number as unsigned integer
%X	Number in Hexadecimal	%x	Number in Hexadecimal
%q	String in quote marks	%s	String without quote marks

When there is an upper-case and lower-case version, letters in the output (hexadecimal digits or exponent markers) are shown in that case.

Case conversion

Returns a copy of a string with all upper case letters changed to lower case, or lower case to upper case, according to the current locale.

```
[string] = string.lower([string])
[string] = string.upper([string])
```

String to table conversion

Girder extension. Splits a string at a given delimiter, returning a table of component strings.
`[table] = string.Split([inputstring], [delimiter])`
inputstring: string containing sections separated by delimiters.

delimiter: single character string identifying the delimiter.

Example:

```
local t = string.Split("test,1,2", ",")
for a, b in ipairs(t) do print(a,b) end
```

Pattern matching

find returns the position of the first substring matching the pattern and any capture strings. **gfind** is an iterator which produces either the captures (up to nine) in each match or (if there are no captures specified in the pattern) the full matching string. **gsub** replaces matches in a string with new strings.

```
[start], [end], [cap ...] = string.find([string], [pattern], [start], [plain])
for [cap ...] in string.gfind([string], [pattern]) do
[string], [subs] = string.gsub([string], [pattern], [replace], [max])
```

string: (string) Original or processed string. As a return, may be nil on error.

pattern: (string) Pattern to be matched (see below).

start: (number) Index of first character matched or to be considered. Optional in **find** and defaults to 1.

end: (number) Index of last character in match.

plain: (number, optional) If 1, turns off pattern tokens and interprets pattern literally.

© 2008 Promixis, LLC

[cap ...]: (string) Up to nine parameters being the capture strings. In **gfind**, will be a single string with the entire match if no captures are specified.

replace: (string, function) If a string, it replaces each occurrence of the matching string. It may contain instances of the special token "%*n*" where *n* is a capture number 1 to 9. If a function, that function is called on each match with up to nine capture strings. It should return a replacement string or **nil** to replace with the empty string.

max: (number, optional) Specifies the maximum number of replacements to make (default is no limit).

Patterns

A pattern is a sequence of ordinary characters and "magic sequences". A magic sequence begins with a character from the set [$()%.[]*+-?].

Character match tokens.

A character matches itself except under the influence of a "magic character". Use **%x** to match a magic character literally (for example, **%$**. A pattern cannot contain embedded zeros (use **%z**).

Character classes match any one character in the class.

Pat	Matches	Pat	Matches
%a	letters	%s	space characters
%c	control characters	%u	upper case letters
%d	digits	%w	alphanumeric characters
%l	lower case letters	%x	hexadecimal digits
%p	punctuation characters	%z	character with representation 0

For all single letter classes (%a, %c, etc.), the corresponding upper-case letter matches any character **not** in the class. The definitions of letter, space, etc. depend on the current locale.

A dot (**.**) is a wildcard, which matches any single character.

A class may be explicitly defined by enclosing characters, character classes or character ranges separated by dash (**-**) in square brackets. The compliment set may be defined by including **^** after the opening bracket. For example, "[%a%.]" matches any letter or a full stop; "[AB]" matches any character except A or B.

The **^** magic character matches an imaginary character before the start of the string and the **$** character matches an imaginary character after the end of the string. These may be thought of as *anchoring* a pattern to the start or end of the string.

Repetitions.

Any character match token may be followed by a magic character specifying repetition.

***** 0 or more repetitions, longest possible sequence.

+ 1 or more repetitions, longest possible sequence.

- 0 or more repetitions, shortest possible sequence.

? 0 or 1 occurrence.

© 2008 Promixis, LLC

The sequence "**%b***xy*" matches a balanced sequence of characters starting with character *x* and ending with character *y*. For example "%b()" matches the whole of "(x+y*(z/p)*i)" NOT "(x +y*(z/p)" because of the requirement to be balanced.

Captures.

Sub-patterns enclosed in parentheses describe captures. When a match succeeds, the substrings of the subject string that match captures are stored (captured). Captures are numbered according to their left parentheses, starting from 1. The empty capture "()" captures the current string position (a number).

These captures may be used to match later characters in the string being analyzed. The magic sequence **%***n* matches a substring equal to the n-th captured string, for n between 1 and 9.

Source: Lua String Manipulation Library & Built-in Girder Extensions.

12.29 table Library

The table Library is standard Lua with some Girder extensions.

Table as list or array

A table can simulate a list or array using consecutive integer keys from 1 up. **getn** determines the size of a list using the following methods.

1. Look for a field keyed "n" with a numeric value.
2. Look for an internal size field which is set by **setn**.
3. As a last resort, find the largest contiguous integer key.

setn records a size internally and also updates field keyed "n" if it is present.

```
[size] = table.getn([table])
table.setn([table], [size])
```

insert inserts an element at an index position in a list, increasing the keys of other elements as necessary to make space, or at the end of the list. **remove** deletes an element closing up the keys of other elements as necessary, or the last element. Both adjust the table size with **setn**.

```
table.insert([table], [pos], [value])
table.insert([table], [value])
table.remove([table], [pos])
table.remove([table])
```

table: Table. The table to be operated on.

pos: Number. The index of the element after insertion or before removal.

value: The value of the element to be inserted.

A list table may be sorted by the value of the elements. This changes the numeric keys of the elements to reflect the sort order of their values.

```
table.sort([table], [comp])
```

comp: (function, optional) If not specified, the operator **<** is used to compare values, including any defined metamethods. If specified, must take two values and return true when the first is less than the second.

A list table may be processed in order using **foreachi** which executes a given function on each element.

```
[ret] = table.foreachi([table], [func])
```

func: (function} Receives the key and the value of an element. Return **nil** to continue iterating or any other value to terminate and return this as ret.

ret: **nil** if the iteration completes, otherwise the value returned by func.

© 2008 Promixis, LLC

A list may be converted to a string with an optional separator.
`[string] = table.concat([table], [sep], [start], [end])`
sep: (string, optional) String to concatenate between elements. Default is empty.

start: (number, optional) Index of first element to use (default = 1).

end: (number, optional) Index of last element to use (default = -1).

General table iteration

Any table may be processed in undefined order using **foreach** which executes a given function on each element.
`[ret] = table.foreach([table], [func])`
func: (function} Receives the key and the value of an element. Return **nil** to continue iterating or any other value to terminate and return this as ret.

ret: **nil** if the iteration completes, otherwise the value returned by func.

Table copy

(Girder extension) Normally when you assign one table variable to another this results in both variables pointing to the same underlying table. This function makes a true copy (you can then modify the new table without effecting the original). The copy is a "deep copy" meaning that any tables within the source table are also copied.
`[newtable] = table.copy([oldtable])`
oldtable: (table) Table to be copied.

newtable: (table) Copy of oldtable.

Tostring and print

Girder extension. **tostring** provides a string containing the initializer for a table in "pretty printed" form. **print** is a shortcut for producing this string and then printing it to the Interactive Lua Console.
`[string] = table.tostring([table], [indent], [spacing])`
`table.print([table], [indent], [spacing])`
table: Table to be formatted.

indent: String to be added in front of indented lines (nested tables). **nil** for "\t".

spacing: String to be added between key/value pairs. **nil** for "".

string: String containing the formatted initializer.

Comparison

Girder extension. Returns true if two tables have identical keys and values.
`[equal] = table.equal([table1], [table2])`
table1, table2: tables to be tested.

equal: Boolean true if tables are equal, else false.

Merging

Girder extension. Merges the contents of one table into another. Does NOT merge the contents of nested tables.
`[destination] = table.joinbyindex([destination], [source], [overwrite])`
destination: Table into which source should be merged.

© 2008 Promixis, LLC

source: Table whose contents should be merged into source.

overwrite: Boolean, true if source should overwrite destination when a key is present in both.

Searching

Girder extension. Finds a given value in a table and returns its key.
`[key] = table.findvalue([table], [value])`
table: Table to search in.

value: Value to search for (any type supporting equality test).

key: The (first) key of the specified value or nil if not found.

IsEmpty

Girder extension.
`[res] = table.IsEmpty([table])`
table: Table. Table to test.

res: Boolean. True if the table has no elements.

Source: Lua Table Manipulation Library & %GIRDER%\luascript\startup\Additions.lua.

12.30 thread Library

The functions in this table provide support for preemptive multitasking of Lua scripts. Further details of this implementation can be found here: LuaThread Manual. (http://www.cs.princeton.edu/~diego/professional/luathread/home.html).

Thread object

The **newthread** function creates a thread object and runs the supplied function on it using the given parameters. **newthread** returns immediately leaving the function running in parallel on the new thread. When the function exits, the thread ceases to exist. If you need to resynchronize the creating thread later use **waitforthread** which blocks the calling thread until the new thread completes. You can test if the new thread is still running without blocking using **isthreadrunning**. **setthreadpriority** is used to tune how much runtime your thread gets and you can read the priority using **getthreadpriority**.

```
[Thread] = thread.newthread([function], [parameters])
[ended] = [Thread]:waitforthread([timeout])
[running] = [Thread]:isthreadrunning()
[THREAD_PRIORITY] = [Thread]:getthreadpriority()
[res] = [Thread]:setthreadpriority([THREAD_PRIORITY])
```

function: Function to be run on the thread. This may have any number and type of parameters. Any return values will be ignored.

parameters: Table of the parameters to [function] keyed by numeric indexes based at 1. Note that these are unpacked and the function is called normally, not with a list parameter.

timeout: Number. The time in milliseconds to wait for the thread to terminate, or the constant thread. INFINITE for no timeout.

ended: Boolean. True if the thread ended. False if a timeout occurred.

running: Boolean. True if the thread is still running. False if it has terminated.

THREAD_PRIORITY: Number. Priority of the thread. Use one of the constants prefixed thread.

© 2008 Promixis, LLC

THREAD_PRIORITY_ - HIGHEST; TIME_CRITICAL; ABOVE_NORMAL; NORMAL; BELOW_NORMAL; IDLE; LOWEST.

Example:

```
function DoWork(name, interval)
   repeat print(name); win.Sleep(interval) until gir.IsLuaExiting()
end
thread.newthread(DoWork, {"Thread1", 5000})
thread.newthread(DoWork, {"Thread2", 8000})
-- Reset Lua Scripting Engine to stop these threads.
```

Mutex object

A **mutex** guards shared resources (usually shared variables) from simultaneous access by more than one thread. The **mutex** is owned by one and only one thread at a time. The **lock** method is used by a thread to apply for the **mutex**. If the **mutex** is unowned, **lock** returns immediately, otherwise it blocks the thread until the **mutex** becomes available. In either case, when **lock** returns, the thread owns the **mutex**. **unlock** relinquishes ownership of the **mutex** allowing another thread to acquire it. For efficiency, threads should minimize the time spent holding the **mutex**, if necessary by making private copies of variables. It is good practice to keep related shared variables in a table and to store the protecting **mutex** in the same table.

```
[Mutex] = thread.newmutex()
[Mutex]:lock()
[Mutex]:unlock()
```

Partial Example:

```
share = {
mutex = thread.newmutex()
-- Shared variables
}
-- A thread accesses the shared variables:
share.mutex:lock()
-- Access the shared variables
share.mutex:unlock()
```

Cond object

A condition object (**cond**) provides a way for a boss thread to signal one or more worker threads that it's time to wake up and do some work. The worker thread calls the **wait** method on the **cond** when it has no more work to do. This blocks the worker thread. The boss thread calls the **signal** method or the **broadcast** method to unblock the worker. A boss may control several workers in which case **signal** wakes one up at random while **broadcast** wakes them all up. The implementation of **cond** requires a **mutex** as well and a thread must hold the **mutex** before calling any of the **cond** methods. While the worker is blocked on a **wait**, the **mutex** is automatically released, but when the worker comes out of **wait** it will again own the **mutex** and should unlock it as soon as possible.

```
[Cond] = thread.newcond()
[Cond]:wait([Mutex])
[Cond]:signal()
[Cond]:broadcast()
```

Partial Example:

```
workpak = {
    cond = thread.newcond(),
    mutex = thread.newmutex()
    -- Shared variables
}
-- A worker thread waits for work:
workpak.mutex:lock()
workpak.cond:wait(workpak.mutex)
-- Read shared variables
```

© 2008 Promixis, LLC

```
workpak.mutex:unlock()
-- A boss thread hands out work:
workpak.mutex:lock()
-- Write shared variables
workpak.cond:signal() -- Work for any worker
-- workpak.cond:broadcast() -- Work for all workers
workpak.mutex:unlock()
```

Source: Built-in.

12.31 transport Library

GIRDER PRO

Below are the low level functions, if you are looking for the classes go here.

The transport library is an abstraction layer for device communication. It handles RS232, HID and TCP/IP communications. If some equipment is available in both serial and tcp connected versions you can use this library to your advantage and only write it once, simply changing one line of code. Note that there are several higher level classes available that help you write this code.

The transport library works asynchronously. That means when you write something to the device the function returns before the data has arrived and reading data from the device is handled to callbacks . Callbacks come in the form of 'parsers' they provide a way to parse whole chunks of data before it is handed to the script.

To create a transport call New, it has a few different overloaded parameter lists depending on what you are trying to do.

```
[transport] = transport.New( [transport_id] )
```

transport_id is a constant that can be found in transport.constants.transport it determines the actual path the data will take, serial, tcpip or HID.

transport.constants.transport.GIP
transport.constants.transport.GIPLISTEN
transport.constants.transport.SERIAL
transport.constants.transport.USBHID

Generic Transport functions

These functions apply to all transports:

```
[transport]:Close()
[transport]:RxClear()
[transport]:TxClear()
[transport]:SetICTOValue( [time] )
[transport]:Write( [data] )
[transport]:Read( ) -- only use this is you REALLY know what you are doing.
[transport]:Callback ( [parser_id] , ... )
```

The Callback function setups a callback for incoming data. For example some devices always terminate the incoming data with a byte 0x13, in that case you can use the terminated parser and it will only call into your function if a the terminator was received passing the complete data in one call. Other reasons for calling the callback are a timeout happend, or the connection is closed or opened and similar events. The available parsers are:

transport.constants.parser.BUFFER

© 2008 Promixis, LLC

All data is put into the buffer and you are responsible for calling Read(), the callback is never called for data
`[transport]:Callback ( transport.constants.parser.BUFFER )`

transport.constants.parser.DISABLE
`[transport]:Callback ( transport.constants.parser.DISABLE )`
All incoming data is discarded.

transport.constants.parser.FIXEDLENGTH
`[transport]:Callback ( transport.constants.parser.FIXEDLENGTH, [length], [timeout], [callback] )`
All incoming data is split into [length] sized chunks. Unless the timeout happens.

transport.constants.parser.MARKEDLENGTH1
`[transport]:Callback ( transport.constants.parser.MARKEDLENGTH1, [byte marker], [trailer length],`
Incoming data is expected to have a start byte [byte marker] and the next byte gives the length of the data minus the [trailer length].

transport.constants.parser.STREAM
`[transport]:Callback ( transport.constants.parser.STREAM,[callback] )`
Simply passes the data that is received along to the callback

transport.constants.parser.TERMINATED
`[transport]:Callback ( transport.constants.parser.`***TERMINATED***`, [terminator], [timeout], [callback`
Calls the callback once it finds the exact match of [terminator] which can be a multi byte string.

Serial Transport functions

```
num, err = [transport]:Open( [comport] )
[transport]:Baud([baud])
[transport]:Flow([flowtype])
[transport]:Parms([parity],[stopbits],[databits])
[transport]:ICD([inter character delay])
[transport]:BurstSize([size])
[transport]:SetDTR([bool])
[transport]:SetRTS([bool])
[transport]:SetBreakBit([bool])
[transport]:GetDCD()
[transport]:GetDSR()
[transport]:GetDTR()
[transport]:GetRI()
[transport]:GetRTS()
[transport]:Status()
```

baudrate: Number. Bits per second (for example 9600).
flowtype: String. "N" - No flow control; "H" - Hardware flow control; "S" - Software flow control.
parity: Number. 0 - None; 1 - Odd; 2 - Even; 3 - Mark; 4 - Space.
stopbits: Number. 0 - One; 1 - One and a half; 2 - Two.
databits: Number (5, 6, 7 or 8). Number of bits of each byte transmitted. Most significant bits are ignored (For example, ASCII can be transmitted in 7 bits).
burstsize: Number. Maximum number of bytes transmitted in one go. Default is 256.
delay: Number (milliseconds). Time inserted between bytes. Default is 0.

TCP/IP Transport functions

```
num, err = [transport]:Open( [url], [port number] )
```

Listen TCP/IP Transport functions

```
num, err = [transport]:Open( [hostname], [port number] )
```

This function starts a listening TCP server on hostname (can be nil) and port number. See telnet example below for usage.

© 2008 Promixis, LLC

HID Transport functions

```
num, err = [transport]:Open( [vendor id], [product id], [manufacturer id], [serial num], [device na
[transport]:GetSerialNumber()
[transport]:GetReadReportLength()
[transport]:GetWriteReportLength()
[transport]:StripReportID([bool])
[transport]:PrependReportID([bool],  [num  id])
[transport]:UseSetReport([bool])
```

HTTP Example

Here is an little bit of example code using the TCP/IP transport to retrieve the Promixis.com homepage.

```
local  c  =  transport.New(transport.constants.transport.GIP)
c:Callback(transport.constants.parser.TERMINATED,  '\r\n',  1000,  function(p1,p2)

       if  (  p2  ==  transport.constants.event.CONNECTIONCLOSED  )  then
              print("Connection  Closed")
              c:Close()
              c = nil
              return
       end

       if  (  p2  ==  transport.constants.event.CONNECTIONESTABLISHED  )  then
              if  (  p1 == 0  ) then
                     print("New  Connection")
              else
                     print("New  Connection  Failed:  ",  p1)
                     c:Close()
                     c = nil
                     return
              end

              return
       end

       print(p1)
end)

c:Open('www.promixis.com',80)
c:Write('GET  /  HTTP/1.0\r\nHost: www.promixis.com\r\n\r\n')
```

Telnet Example

This example demonstrates how to make a dump telnet server

```
function  mycbserver(p1,p2)

       if  (  p2  ==  transport.constants.event.CONNECTIONCLOSED  )  then
              print("Connection  Closed")
              return
       end

       if  (  p2  ==  transport.constants.event.NEWCONNECTION  )  then

              print("New  Incoming  Connection")
```

© 2008 Promixis, LLC

```
                p1:Callback(transport.constants.parser.TERMINATED,'\n',  3000,  function  (cp1,cp2)

                        if  (  cp2  ==  transport.constants.event.CONNECTIONCLOSED  )  then
                                print("Client  Connection  Closed")
                                p1:Close()
                                clients[p1]=nil
                                return
                        end

                        if  (  cp2  ==  transport.constants.event.CONNECTIONESTABLISHED  )  then

                                if ( cp1 == 0 ) then
                                        print("New  Client  Connection")
                                        p1:Write("welcome  to  our  echo  telnet  server!\r\n")
                                        clients[p1]=true
                                else
                                        print("New  Client  Connection  Failed:  ")
                                        p1:Close()
                                        clients[p1]=nil
                                end

                                return
                        end
                        print(cp1)
                        p1:Write(cp1  ..  '\r\n')

                end)

                return
        end

end

if ( clients ) then
        for c,v in pairs(clients ) do
                c:Close()
        end
end

if ( c5 ) then
        c5:Close()
end
clients = {}
c5  =  transport.New(transport.constants.transport.GIPLISTEN)
if  not  c5:Open(nil,  12345)  then
        print('Could  not  start  listening')
        c5:Close();
        c5 = nil;
end
c5:Callback(transport.constants.parser.TERMINATED,  '\r\n',  1000,  mycbserver)
print('Listening  for  connections  on  port  12345,  use  telnet  localhost  12345  from  the  commandline  to
```

12.31.1 Transport Classes

Transport Class Documentation

transport.Base
- Events
- Methods

© 2008 Promixis, LLC

transport.TransactionBased
- Events
- Methods

transport.Event
- Methods

transport.Core
- Methods
- Events

12.31.1.1 Transport Event Handlers

All transport based classes support a list of Event Handlers specified below.

```
OnInitialized = function(self)
end,
```

Called after a class is initialized.

```
OnInitializeFail = function(self)
end,
```

Initialization failed.

```
OnConnecting = function(self,...)
end,
```

Attempting to connect

```
OnConnected = function(self)
end,
```

Connection was established

```
OnFirstConnected = function(self)
end,
```

Called if this is the first time this instance of the transport object connected.

```
OnReconnected = function(self)
end,
```

Called if the instance of the transport object reconnected.

```
OnConnectFail = function(self,reason)
end,
```

Called if the connection failed.

```
OnNewConnection = function(self)
end,
```

© 2008 Promixis, LLC

Called when a low-level transport object is created.

```
OnConnectionCheck = function (self)
end,
```

Return true if this connection is still alive, return false if the connection should be reattempted. This is only called in conjunction with the connection monitor.

```
OnDisconnecting = function(self)
end,
```

The connection is about to disconnect.

```
OnDisconnected = function(self)
end,
```

The connection is disconnected.

```
OnError = function(self, count)
end,
```

Called if the connection had too many errors.
The following return codes are supported:
ResponseCodes.Ok, continue with the next item in the send queue.
ResponseCodes.Wait, wait for more data to come in, do not send the next item from the queue.
ResponseCodes.Retry, resend last command.
ResponseCodes.Reset, Reset the connection (close/open)
ResponseCodes.Error, an error has occurred, send the next item.

```
OnReceiveData = function(self, Event)
end,
```

Called if data is received. In the transaction based classes it is best not to override this unless you know what you are doing.

The following return codes are supported:
ResponseCodes.Ok, continue with the next item in the send queue.
ResponseCodes.Wait, wait for more data to come in, do not send the next item from the queue.
ResponseCodes.Retry, resend last command.
ResponseCodes.Reset, Reset the connection (close/open)
ResponseCodes.Error, an error has occurred, send the next item.

```
OnReceiveIncompleteData = function(self,Event)
end,
```

Called if the Timeout expired for the Parser. In the transaction based classes it is best not to override this unless you know what you are doing.

```
OnNoResponse = function(self)
end,
```

© 2008 Promixis, LLC

Called if the NoResponseTimeout expires. In the transaction based classes it is best not to override this unless you know what you are doing.

```
OnReceiveEvent = function(self,Event)
end,
```

Called on NON data events, for example serial line changes.

```
OnStatus = function (self)
end,
```

Called when the status is changed for the connection

```
OnEvent = function (self,Event)
end,
```

OnEvent is called before any processing takes place on an event from the lowlevel transport object. Handle with extreme care.

The following return codes are supported:
ResponseCodes.Ok, continue with the next item in the send queue.
ResponseCodes.Wait, wait for more data to come in, do not send the next item from the queue.
ResponseCodes.Retry, resend last command.
ResponseCodes.Reset, Reset the connection (close/open)
ResponseCodes.Error, an error has occurred, send the next item.
nil, normal processing

```
OnSend = function(self, command)
end,
```

Called right before the data is queued in the transport system for sending. Note that it might not actually be sent out to the device yet.

```
OnBuildCommand = function(self, data)
    return data
end,
```

If this class needs to perform special build steps on the data do it here.

```
OnQueued = function(self, position,command)
end,
```

Called right after a new command is queued.

```
OnQueueFull = function(self)
end,
```

Called when the Queue if full.

```
OnQueueEmpty = function(self)
end,
```

Called when the Queue is empty.

© 2008 Promixis, LLC

12.31.1.2 Transport Methods

All transport object support the following methods

```
Obj:Close()

Obj:Log(...)

Obj:Subscribe(...)

Obj:Unsubscribe(...)

Obj:Send(data)

<string>  =  Obj:GetName()

<string>  =  Obj:GetTransportDescription()

<string>  =  Obj:GetTransportStatusDescription()

<string>  =  Obj:GetTransportStatus()

<table>  =  Obj:GetStatus()

<string>  =  Obj:GetID()
```

12.32 usbuirt Library

This command transmits an IR code using a USB-UIRT PC device.
`usbuirt.TransmitIR([ircode], [repeats], [zone], [waittime])`
ircode: String. The code to transmit in UIRT form or Pronto CCF form.

repeats: Number. The number of times the transmission should be repeated.

zone: Number (optional). The USB-UIRT device to transmit from if more than one.

waittime: Number (optional). The time in milliseconds to wait for IR quiet before transmitting.

Source: USB-UIRT driver Plugin.

12.33 Voice Library

Requires Girder Pro.

The Voice library provides functions for text to speech conversion. The library must be enabled from the Voice tab in Plugins Settings.
`Voice:Speak([text], [async])`
`Voice:Volume([volume])`
`Voice:Enable([enable])`
text: String. The text to be spoken.

async: Boolean. True to return immediately before the speach is finished.

volume: Number. 0..100. Volume of speach audio output.

enable: Boolean. True to enable, false to disable.

© 2008 Promixis, LLC

Example:
```
Voice:Speak(date:now().Hour)
Voice:Speak(date:now().Minute)
```

Source: %GIRDER%luascript\Voice.lua (loaded by Settings Page).

12.34 win Library

Provides Lua bindings for the most important Windows API functions.

WARNING: THE FUNCTIONS IN THIS SECTION ARE VERY LOW LEVEL WRAPPERS TO THE WINDOWS API. USE AT YOUR OWN RISK!

1. Obtaining Window Handles functions provide many ways of obtaining the *handle* (unique run-time identifier) of any current window. These handles are required to use the functions in the next section.
2. Manipulating Windows functions require a window handle and do something to the specified window.
3. Drive Functions are functions relating to disk drives.
4. Time and Date Functions.
5. Process Functions enable discovery and manipulation of application processes.
6. Menu Functions allow discovery and triggering of menus in applications.
7. File and Directory Functions.
8. FolderWatcher raises events or runs Lua functions whenever specified changes happen in the filesystem.
9. Screen Saver Functions
10. Mouse and Pointer Functions
11. Capture and Clipboard Functions allow capture of images of elements on the screen and interacting with the system clipboard.
12. Network Functions.
13. Miscellaneous Functions.
14. Registry objects for reading and writing the system registry.

Source: Built-in.

12.34.1 Obtaining Window Handles

Note: An alternative method of finding window handles is to use **GetWindowMatches** from the gir Library.

FindWindow, FindWindowEx

Obtain the window handle of a specific window.
```
[handle],[err] = win.FindWindow([classname],[title])
[handle],[err] = win.FindWindowEx([parenthandle],[starthandle],[classname],[title])
```
handle: Number. The window handle or nil if not found.

classname: String. The name of the class of the window sought (or nil for any).

title: String. The text in the titlebar of the window sought (or nil for any).

parenthandle: Number. The handle of the parent window of the window sought.

starthandle: Number. Start looking for the next child after this one (or nil for first).

FindProcessWindow

Obtain the window handle of the main window of a specified process.
```
[handle] = win.FindProcessWindow([processid])
```

© 2008 Promixis, LLC

handle: Number. The window handle or nil if not found.

processid: Number. The identifier of the process owning the window sought.

Example:
```
print(win.FindProcessWindow(win.FindProcess("girder.exe")))
```

GetForegroundWindow

Obtain the handle of the window with the focus.
`[handle] = win.GetForegroundWindow()`
handle: Number. The window handle of the window with the focus.

GetDesktopWindow

Obtain the handle of the desktop window which represents the entire screen.
`[handle] = win.GetDesktopWindow()`
handle: Number. The window handle of the desktop window.

GetWindow, GetNextWindow

Obtain the handle of a window having a specified relationship to another window.
`[handle] = win.GetWindow([handle], [GW])`
`[handle] = win.GetNextWindow([handle], [pos])`
handle: Number: Handle of old or new window, nil if there are no more windows.

pos: Number. win.NEXT or win.PREVIOUS, referring to z-order.

GW: Number. One of the constants below, prefixed win.GW_.

- CHILD: The retrieved handle identifies the child window at the top of the Z order, if the specified window is a parent window; otherwise, the retrieved handle is nil. The function examines only child windows of the specified window. It does not examine descendant windows.
- ENABLEDPOPUP: (Windows 2000/XP) The retrieved handle identifies the enabled popup window owned by the specified window (the search uses the first such window found using GW_HWNDNEXT) otherwise, if there are no enabled popup windows, the retrieved handle is that of the specified window.
- HWNDFIRST: The retrieved handle identifies the window of the same type that is highest in the Z order. If the specified window is a topmost window, the handle identifies the topmost window that is highest in the Z order. If the specified window is a top-level window, the handle identifies the top-level window that is highest in the Z order. If the specified window is a child window, the handle identifies the sibling window that is highest in the Z order.
- HWNDLAST: The retrieved handle identifies the window of the same type that is lowest in the Z order. If the specified window is a topmost window, the handle identifies the topmost window that is lowest in the Z order. If the specified window is a top-level window, the handle identifies the top-level window that is lowest in the Z order. If the specified window is a child window, the handle identifies the sibling window that is lowest in the Z order.
- HWNDNEXT: The retrieved handle identifies the window below the specified window in the Z order. If the specified window is a topmost window, the handle identifies the topmost window below the specified window. If the specified window is a top-level window, the handle identifies the top-level window below the specified window. If the specified window is a child window, the handle identifies the sibling window below the specified window.
- HWNDPREV: The retrieved handle identifies the window above the specified window in the Z order. If the specified window is a topmost window, the handle identifies the topmost window above the specified window. If the specified window is a top-level window, the handle identifies the top-level window above the specified window. If the specified window is a child window, the handle identifies the sibling window

© 2008 Promixis, LLC

above the specified window.
- OWNER: The retrieved handle identifies the specified window's owner window, if any.

GetParent

Obtain the handle of the parent window to a specified window, or if the window is top-level the owner window if any.
`[handle] = win.GetParent([handle])`
handle: Number. Handle of old or new window, return is nil for an un-owned top level window.

EnumWindows, EnumChildWindows

Invoke a Lua callback function for each top-level window or for each child window of a given window.
```
[ret] = [callback]([handle])
win.EnumWindows([callback])
win.EnumChildWindows([parent], [callback])
```
parent: Number. The handle of the window whose children are to be enumerated.

handle: Number. The handle of each window found.

ret: return 1 to continue the enumeration, else it is terminated.

12.34.2 Manipulating Windows

SetForegroundWindow, ForceForegroundWindow, BringWindowToTop

Moves the identified window to the foreground (the window that is currently active, similar to clicking on its title bar). **SetForegroundWindow** uses the Windows API and is often not effective on Windows 2000 and XP which tries to prevent the foreground window being changed except by the user. **ForceForegroundWindow** uses non-API techniques and often succeeds where **SetForegroundWindow** fails. But use with thought - it is annoying to suddenly find you are typing into the wrong window! **BringWindowToTop** uncovers a partially or completely obscured window but does not necessarily focus it.
```
[res] = win.SetForegroundWindow([handle])
[res] = win.ForceForegroundWindow([handle])
[res] = win.BringWindowToTop([handle])
```
handle: Number. Window handle of the window to change.

ShowWindow, OpenIcon, CloseWindow

ShowWindow changes the state of the specified window according to the command code. OpenIcon restores a minimized window. CloseWindow minimizes a window.
```
[res] = win.ShowWindow([handle], [SW])
[res] = win.OpenIcon([handle])
[res] = win.CloseWindow([handle])
```
handle: Number. Window handle of the window to change.

SW: Number. Command code as below prefixed win.SW_ .
- FORCEMINIMIZE: Windows 2000/XP: Minimizes a window, even if the thread that owns the window is hung. This flag should only be used when minimizing windows from a different thread.
- HIDE: Hides the window and activates another window.
- MAXIMIZE: Maximizes the specified window.
- MINIMIZE: Minimizes the specified window and activates the next top-level window in the Z order.
- RESTORE: Activates and displays the window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when restoring a minimized window.
- SHOW: Activates the window and displays it in its current size and position.

© 2008 Promixis, LLC

- SHOWDEFAULT: Sets the show state based on the SW_ value specified in the STARTUPINFO structure passed to the CreateProcess function by the program that started the application.
- SHOWMAXIMIZED: Activates the window and displays it as a maximized window.
- SHOWMINIMIZED: Activates the window and displays it as a minimized window.
- SHOWMINNOACTIVE: Displays the window as a minimized window. This value is similar to win.SW_SHOWMINIMIZED, except the window is not activated.
- SHOWNA: Displays the window in its current size and position. This value is similar to win.SW_SHOW, except the window is not activated.
- SHOWNOACTIVATE: Displays a window in its most recent size and position. This value is similar to win.SW_SHOWNORMAL, except the window is not activated.
- SHOWNORMAL: Activates and displays a window. If the window is minimized or maximized, the system restores it to its original size and position. An application should specify this flag when displaying the window for the first time.

IsHungAppWindow, IsIconic, IsWindowInView, IsWindowVisible, IsZoomed

Tests for the state of a specified window.

```
[res] = win.IsHungAppWindow([handle]) -- App. owning window is hung
[res] = win.IsIconic([handle])        -- Window is minimized
[res] = win.IsWindowObscured([handle])  -- Any part of window is obscured
[res] = win.IsWindowVisible([handle]) -- Window has visibility state true
[res] = win.IsZoomed([handle])        -- Window is maximized
```

handle: Number. Window handle of window to be tested.

res: Boolean.

KillWindow

Closes the application owning the specified window. Asks nicely first, but if it does not comply, kills it.

```
win.KillWindow([handle], [timeout])
```

handle: Number. Window handle of window to be closed.

timeout: Number. Milliseconds to wait before forcing closure.

MoveWindow, SetWindowPos

Moves and resizes a specified window.

```
[res] = win.MoveWindow([handle], [left], [top], [width], [height], [repaint])
[res] = win.SetWindowPos([handle],[order], [left], [top], [width], [height], [SWP])
```

handle: Number. Window handle of window to change.

left: Number. New position of left side of window.

right: Number. New position of top side of window.

width: Number. New width of window.

height: Number. New height of window.

repaint: Boolean. True to repaint the window.

order: Number. Window handle to set order relative to, or constant as below prefixed win.HWND_.

- BOTTOM: Places the window at the bottom of the Z order. If the hWnd parameter identifies a topmost window, the window loses its topmost status and is placed at the bottom of all other windows.
- NOTOPMOST: Places the window above all non-topmost windows (that is, behind all topmost windows). This flag has no effect if the window is already a non-topmost

© 2008 Promixis, LLC

window.
- TOP: Places the window at the top of the Z order.
- TOPMOST: Places the window above all non-topmost windows. The window maintains its topmost position even when it is deactivated.

SWP: Bitwise OR of flags below, prefixed win.SWP_.
- ASYNCWINDOWPOS: If the calling thread and the thread that owns the window are attached to different input queues, the system posts the request to the thread that owns the window. This prevents the calling thread from blocking its execution while other threads process the request.
- DEFERERASE: Prevents generation of the WM_SYNCPAINT message.
- DRAWFRAME: Draws a frame (defined in the window's class description) around the window.
- FRAMECHANGED: Applies new frame styles set using the SetWindowLong function. Sends a WM_NCCALCSIZE message to the window, even if the window's size is not being changed. If this flag is not specified, WM_NCCALCSIZE is sent only when the window's size is being changed.
- HIDEWINDOW: Hides the window.
- NOACTIVATE: Does not activate the window. If this flag is not set, the window is activated and moved to the top of either the topmost or non-topmost group (depending on the setting of the hWndInsertAfter parameter).
- NOCOPYBITS: Discards the entire contents of the client area. If this flag is not specified, the valid contents of the client area are saved and copied back into the client area after the window is sized or repositioned.
- NOMOVE: Retains the current position (ignores X and Y parameters).
- NOOWNERZORDER: Does not change the owner window's position in the Z order.
- NOREDRAW: Does not redraw changes. If this flag is set, no repainting of any kind occurs. This applies to the client area, the nonclient area (including the title bar and scroll bars), and any part of the parent window uncovered as a result of the window being moved. When this flag is set, the application must explicitly invalidate or redraw any parts of the window and parent window that need redrawing.
- NOREPOSITION: Same as the SWP_NOOWNERZORDER flag.
- NOSENDCHANGING: Prevents the window from receiving the WM_WINDOWPOSCHANGING message.
- NOSIZE: Retains the current size (ignores the cx and cy parameters).
- NOZORDER: Retains the current Z order (ignores the hWndInsertAfter parameter).
- SHOWWINDOW: Displays the window.

GetWindowRect

Gets the coordinates of each edge of the window on the screen. Screen coordinates run right and down from the upper left corner.
`[top], [left], [bottom], [right] = win.GetWindowRect([handle])`
handle: Number. Window handle of the window.

top: Number. Y coordinate of the top edge of the window (nil on failure).

left: Number. X coordinate of the left edge of the window.

bottom: Number. Y coordinate of the bottom edge of the window.

right: Number. X coordinate of the right edge of the window.

GetWindowText

Gets the text of the specified window title bar. Can also get the text contents of some child controls.
`[windowtext] = win.GetWindowText([handle])`
handle: Number. Window handle of window.

windowtext: String with the text, or nil on failure.

© 2008 Promixis, LLC

GetClassName, RealGetWindowClass

```
[classname] = win.GetClassName([handle])
[classname] = win.RealGetWindowClass([handle])
```
handle: Number. Window handle of window.

classname: String. The class name of the window or nil on error.

SetWindowOpaque, SetWindowTransparency, FadeWindow, SetWindowLayeredAttributesOff

Set the transparency/opacity key color and level for a window under Windows 2000 and later. FadeWindow animates a fade-in or fade-out effect.
```
[res] = win.SetWindowOpaque([handle], [red], [green], [blue])
[res] = win.SetWindowTransparency([handle], [trans])
[callback]()
win.FadeWindow([handle], [start], [stop], [speed], [async], [callback])
[res] = win.SetWindowLayeredAttributesOff([handle])
```
handle: Number. Window handle of window.

red: Number (0..255). Red component of color key.

green: Number (0..255). Green component of color key.

blue: Number (0..255). Blue component of color key.

trans: Number (0..255). Transparency level.

start: Number (0..255). Starting transparency level.

end: Number (0..255). Ending transparency level.

speed: Number. Speed of effect in milliseconds (suggest 1000).

async: Boolean or nil. Non-nil processes the effect asynchronously on a thread and then executes the callback function, if provided.

SetWindowPriority

Sets the thread priority of the specified window.
```
[result] = win.SetWindowPriority([handle], [priority])
```
handle: Number. Window handle of window.

priority: win.NORMAL_PRIORITY_CLASS; win.IDLE_PRIORITY_CLASS; win.HIGH_PRIORITY_CLASS; win.REALTIME_PRIORITY_CLASS.

SendMessage, SendMessageGetText, SendMessageTimeout, PostMessage

Send messages to a window. **PostMessage** does not wait for a reply, the others do. **SendMessageGetText** works only with **WM_GETTEXT** and **LB_GETTEXT** messages and returns a text response. **SendMessageTimeout** sends messages to windows in other processes and blocks until a response is received or the timeout expires.
```
[response] = win.SendMessage([handle], [message], [wparam], [lparam])
[response], [text] = win.SendMessageGetText([handle], [message], [wparam])
[res], [response] = win.SendMessageTimeout([handle], [message], [wparam],
    [lparam], [SMTO], [timeout])
[res] = win.PostMessage([handle], [message], [wparam], [lparam])
```
handle: Number. Window handle to send message to or win.HWND_BROADCAST.

© 2008 Promixis, LLC

message: Number. Message code.

wparam: Number. Message parameter.

lparam: Number. Message parameter.

timeout: Number. Duration in milliseconds windows are allowed for a response.

response: nil on failure, otherwise the, message response.

text: Text string returned by message.

SMTO: Number. One of the constants below, prefixed win.SMTO_.

- ABORTIFHUNG: Returns without waiting for the time-out period to elapse if the receiving thread appears to not respond or "hangs."
- BLOCK: Prevents the calling thread from processing any other requests until the function returns.
- NORMAL: The calling thread is not prevented from processing other requests while waiting for the function to return.
- NOTIMEOUTIFNOTHUNG: Microsoft Windows 2000/Windows XP: Does not return when the time-out period elapses if the receiving thread stops responding.

IsChild

Tests it one window is a child or descendant of another.
`[result] = win.IsChild([parenthandle], [childhandle])`
parenthandle: Number. Handle of the possible parent.

childhandle: Number. Handle of the possible child.

result: nil if the relationship does not hold.

GetWindowLong

Returns information about a window.
`[data] = win.GetWindowLong([handle], [index])`
data: Number. 32-bit value at stated index.

handle: Number. Window handle.

index: Number. Zero-based offset of value to be retrieved or one of the following keys - win.GWL_EXSTYLE; win.GWL_STYLE; win.GWL_HINSTANCE; win.GWL_HWNDPARENT; win.GWL_ID.

HighlightWindow

Draws a highlight marker round a window.
`win.HighlightWindow([handle])`
handle: Number. Window handle.

RefreshWindow

Requests window to redraw.
`win.RefreshWindow([handle])`
handle: Number. Window handle.

HideWindowFromTaskbar

Stops the window from showing in the taskbar.
`[res] = win.HideWindowFromTaskbar([hwnd])`
handle: Number. Window handle.

© 2008 Promixis, LLC

12.34.3 Drive Functions

GetDriveType, GetDriveStatus, GetVolumeInformation, GetDiskFreeSpace
```
[DRIVE] = win.GetDriveType([path])
[status] = win.GetDriveStatus([path])
[label] = win.GetVolumeInformation([path])
[size],[free] = win.GetDiskFreeSpace([path])
```
path: String. Root directory of drive ([[C:\]]).

DRIVE: Number. win.DRIVE_UNKNOWN; win.DRIVE_NO_ROOT_DIR; win.DRIVE_REMOVABLE; win.DRIVE_FIXED; win.DRIVE_REMOTE; win.DRIVE_CDROM; win.DRIVE_RAMDISK.

status: Number. win.ERROR_SUCCESS; win.ERROR_PATH_NOT_FOUND; win.ERROR_NOT_READY.

label: String. Volume label.

size: Number. Total size of volume in mb.

free: Number. Free space on volume in mb.

OpenCDTray, CloseCDTray
```
[res] = win.OpenCDTray([drive])
[res] = win.CloseCDTray([drive])
```
drive: String. Drive letter followed by colon ("D:").

12.34.4 Time and Date Functions

GetElapsedSeconds, GetElapsedMilliseconds, GetTickCount, GetLastInputTime
```
[seconds] = win.GetElapsedSeconds([previous])
[millisec] = win.GetElapsedMilliseconds([previous])
[millisec] = win.GetTickCount()
[seconds] = win.GetLastInputTime()
```
previous: Number. Previous function return value or nil.

millisec: Number. Milliseconds since start or previous if supplied.

seconds: Number. Seconds since start or previous if supplied.

GetTimeZone, GetLocalTime, GetSystemTime, GetDateOffset, ToOtherTimeZone

GetLocalTime returns the time in the time zone configured for this computer. **GetSystemTime** the UTC time. **ToOtherTimeZone** converts a UTC time to a named time zone (does not work in Windows 98 or earlier).

```
[mode],[bias],[zone] = win.GetTimeZone()
[datetime] = win.GetLocalTime()
[datetime] = win.GetSystemTime()
[datetime] = win.GetDateOffset([datetimeref], [datetimeoffs])
[datetime] = win.ToOtherTimeZone([datetime], [zonename])
```
mode: String. "Daylight" or "Standard".

bias: Number. Minutes offset from GMT.

zone: String. Description of time zone.

datetime: Table. ["Year"]; ["Month"] (Jan = 1); ["DayOfWeek"] (Sun = 0); ["Day"] (1..31); ["Hour"] (0..23); ["Minute"] (0..59); ["Second"] (0..59); ["Milliseconds"] (0..999).

datetimeref: Table. Key as above, used as reference for offset calculation.

© 2008 Promixis, LLC

datetimeoffs: Table. Keys as above except ["DayOfWeek"] is not used. Values are unlimited and may be negative (you can find 1001 days before the reference with {["Day"] = -1001}).

zonename: String. Time zone name which must exactly match one of the subkeys of HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones in the registry.

ToOleDateTime, FromOleDateTime

Converts between datetime table (see above) and the OLE date and time format used in COM Automation and Visual Basic.
`[oledate] = win.ToOleDateTime([datetime])`
`[datetime] = win.FromOleDateTime([oledate])`
oledate: Number. Integer part is number of days since 31 December 1899. Fractional part is the time (.0 = Just after midnight the previous day, .999 etc. is just before midnight.

datetime: Table. ["Year"]; ["Month"] (Jan = 1); ["DayOfWeek"] (Sun = 0); ["Day"] (1..31); ["Hour"] (0..23); ["Minute"] (0..59); ["Second"] (0..59); ["Milliseconds"] (0..999).

Sleep

Blocks the current thread for a fixed time. Durations over 1000 (1 second) should not be used on the main thread as this will delay all operations. Longer sleeps may be useful in secondary threads.

`win.Sleep([duration])`
duration: Number. Milliseconds to sleep for.

12.34.5 Process Functions

EnumProcesses, FindProcess
`[count] = win.EnumWindows([callback])`
`[ret] = [callback]:([procid], [assocexe])`
`[procid] = win.FindProcess([assocexe])`
count: Number of processes found.

ret: Return 1 from callback to continue enumeration, else nil.

procid: Number. Process ID.

assocexe: String. Associated exe file name.

ShellExecuteEx
`[res] = win.ShellExecuteEx([filename], [args], [workdir], [SW])`
filename: String. Full path and name of file to execute.

args: String. Command line arguments or nil.

workdir: String. Path to working directory or nil.

SW: Number. win.SW_MAXIMIZE; win.SW_MINIMIZE; win.SW_SHOWNORMAL.

TerminateProcess

Last resort process termination. Note it does not notify the application first, so may result in loss of data.
`[res] = win.TerminateProcess([procid])`
procid: Number. Process ID.

© 2008 Promixis, LLC

12.34.6 Menu Functions

GetMenu, GetMenuItemCount, GetSubMenu, GetMenuItemID, GetMenuItemInfo

```
[menu] = win.GetMenu([hwnd])
[count] = win.GetMenuItemCount([menu])
[menu] = win.GetSubMenu([hwnd], [pos])
[menuid] = win.GetMenuItemID([menu], [pos])
[state] = win.GetMenuItemInfo([menu],[pos],1)
[state] = win.GetMenuItemInfo([menu],[menuid],0)
```

hwnd: Number. Handle of window owning the menu.

menu: Number. Menu handle.

count: Number of menu items in the menu.

pos: Number. Position of menu item 0 to count - 1 counting from top.

menuid: Number. ID of menu item.

state: Number. Bitmap - bit1 = GRAYED; bit2 = DISABLED; bit4 = CHECKED; bit5 = POPUP; bit6 = MENUBARBREAK; bit7 = MENUBREAK; bit8 = HILITE.

12.34.7 File and Directory Functions

Files, FindFirstFile, FindNextFile, FindClose

```
[dir] = win.Files([path]) -- Iterator wraps the next three functions
[handle],[fd] = win.FindFirstFile([path])
[fd] = win.FindNextFile([handle])
[res] = win.FindClose([handle])
```

dir: Table of file descriptor tables.

path: String. Path and (possibly wildcarded) filename.

handle: Number. Abstract handle used to link the functions.

fd: Table. File descriptor. Use example below to view structure.

Example:

```
for fa in win.Files ("c:\\*.*") do
   if math.band (fa.FileAttributes, win.FILE_ATTRIBUTE_DIRECTORY) == 0 then
      print (fa.FileName, fa.FileSize, fa.LastWriteTime.Year,
          fa.LastWriteTime.Month, fa.LastWriteTime.Day)
    end
end
```

Example:

```
local h, fd = win.FindFirstFile("*.*")
while (fd ~= nil) do
   table.print(fd)
 fd = win.FindNextFile(h)
end
win.FindClose(h)
```

MakePath, SplitPath

```
[path] = win.MakePath([drive], [directory], [name], [ext])
[drive], [directory], [name], [ext] = win.SplitPath([path])
```

drive: String. Drive letter.

directory: String. Directory path.

© 2008 Promixis, LLC

name: String. File name.

ext: String. File name extension.

path: String. Full file path.

FileExists, PathExists
[res],[err] = win.FileExists([name])
[res],[err] = win.PathExists([path])
name: String. File path and name. May include wildcards.

path: String. Directory path (no trailing backslash).

res: 1 if exists, nil if error or not exists.

SHFileOperation, SHCreateDirectory

File operations using Shell functions. Supports wildcards and recursion.
[res] = win.SHFileOperation([source], [dest], [FO], [FOF])
[res] = win.SHCreateDirectory([path])
source: String. Path and filename (may include wildcards)

dest: String. Path plus filename for rename. nil for delete.

FO: Number. win.FO_COPY; win.FO_DELETE; win.FO_MOVE; win.FO_RENAME.

FOF: Number. FOF_FILESONLY; win.FOF_NORECURSION.

path: String. Directory path to create.

CreateDirectory, RemoveDirectory, DeleteFile, CopyFile

These create or remove only the final section of the directory path. To create an entire path, use **SHCreateDirectory**.
[res],[err] = win.CreateDirectory([path])
[res],[err] = win.RemoveDirectory([path])
[res] = win.DeleteFile([path])
[res] = win.CopyFile([source], [dest], [boolean failifexists])
path: String. Directory path (no trailing backslash).

BrowseForFolder, GetOpenFileName, GetSaveFileName

Dialog boxes to select a directory or a file, or to create a file.
[folder] = win.BrowseForFolder([title], [root], [create])
[filepath] = win.GetOpenFileName([title], [root], [file], [filter], [OFN])
[filepath] = win.GetSaveFileName([title], [root], [file], [filter], [OFN])
title: String. Title of dialog box.

root: String. Position in directory tree to start at.

file: String. Filename to highlight or default, or "".

filter: String. Pairs of filter names and filter wildcards seperated by pipe symbols ("All Files|(*.*)|Text Files|(*.txt)").

OFN: Number. Combination of win.OFN_OVERWRITEPROMPT; win.OFN_PATHMUSTEXIST; win.OFN_FILEMUSTEXIST; win.OFN_CREATEPROMPT; win.OFN_ENABLESIZING.

folder: String. nil or selected folder path.

© 2008 Promixis, LLC

filepath: String. nil or selected file path.

GetDirectory, GetTempPath, GetCurrentDirectory, SetCurrentDirectory
```
[path] = win.GetDirectory([name])
[path] = win.GetTempPath()
[path] = win.GetCurrentDirectory()
[res],[err] = win.SetCurrentDirectory([path])
```
name: String. "MYPICTURES"; "PROGRAMFILES"; "WINDOWS"; "MYDOCUMENTS"; "SYSTEM"; "APPDATA"; "STARTMENU"; "DESKTOP"; "PROFILE"; "GIRDERDIR"; "TEMP".

path: String. The directory path.

GetTempFilename
```
[file] = GetTempFilename([path], [prefix], [unique])
```
path: String. Directory in which to create the file.

prefix: String. Fixed beginning part of filename.

unique: Number. 0 to have system generate unique hexadecimal suffix. Otherwise the value in hexadecimal is used even if it is not unique.

CompareFiles
```
[same],[err] = CompareFiles([filename1], [filename2])
```
filename1: String. Path and name of first file.

filename2: String. Path and name of second file.

same: Boolean. True if both files are the same.

12.34.8 FolderWatcher

CreateFolderWatcher creates an object which continually watches the file system of your computer for specified changes. When it sees a change it either executes a callback script or generates a Girder Event.
```
[callback]()
[Watcher] = win.CreateFolderWatcher
   ([folder], [eventstring], [eventdevice], [conditions], [recurse])
-- Or alternative form:
   ([folder], [callback], [conditions], [recurse])
[Watcher]:print()
[Watcher]:Destroy()
```
folder: String path to folder to be watched.

eventstring: String. Event to be generated.

eventdevice: Number. Device on which to generate event (suggest 18 = Girder).

conditions: Number. Sum of one or more of the following keys - 1 = Filename changed; 2 = Foldername changed; 4 = File or Folder attributes changed; 8 = File size changed; 16 = File write time or date changed; 32 = File access timer or date changed; 64 = File created; 256 = File or Folder security descriptor changed.

recurse: Number. 1 = watch the specified folder and all folders contained within it recursively, 0 = just the specified folder.

NOTE: Currently neither the event nor the callback mechanism return specific information about what actually changed, so this is probably only useful for watching a specific file.

Example:
```
function watchfunc() print("watchfunc") end
```

© 2008 Promixis, LLC

```
watcher = win.CreateFolderWatcher(
   [[C:\Documents and Settings\John\My Documents\]], watchfunc, 383, 1)
watcher:print()
```

12.34.9 Screen Saver Functions

IAMBusyOn, IAmBusyOff, GetScreenSaverEnable, SetScreenSaverEnable, ScreenSaverOn, ScreenSaverOff

IAmBusyOn sets your computer so the screen saver and monitor power saving will not cut in, **IAmBusyOff** restores these functions to normal. **ScreenSaverOn** makes the screen saver cut in immediately, **ScreenSaverOff** dismisses the screen saver. **SetScreenSaverEnable** enables or disables the screen saver **GetScreenSaverEnable** notifies if it is disabled or enabled.

```
win.IAmBusyOn()
win.IAmBusyOff()
[oldstate] = win.SetScreenSaverEnable([newstate])
[enabled] = win.GetScreenSaverEnable()
[res] = win.ScreenSaverOn()
[res] = win.ScreenSaverOff()
```

oldstate: Number 0 = disabled, 1 = enabled.

newstate: Number 0 = disabled, 1 = enabled.

enabled: nil if disabled, 1 if enabled.

12.34.10 Mouse and Pointer Functions

GetMousePosition, MoveMouse, MouseClick, MouseDown, MouseUp

```
[x],[y] = win.GetMousePosition([rel])
win.MoveMouse([x], [y], [speed], [rel])
win.MouseClick([button], [duration])
win.MouseDown([button])
win.MouseUp([button])
```

rel: nil for absolute virtual screen coordinates, 0 for relative to foreground window.

speed: Number. 0 for immediate move, larger for slower movement.

x, y: Number. Coordinates.

duration: Number. Duration of click in milliseconds.

button: Number. win.RightButton; win.MiddleButton; win.LeftButton.

ShowCursor

```
[counter] = win.ShowCursor([bshow])
```

bshow: Number. 1 = Increment counter; 0 = Decrement counter.

counter: Number. Cursor is shown if counter is 0 or greater.

12.34.11 Capture and Clipboard Functions

ScreenCapture

```
[res],[err] = win.ScreenCapture([source], 1, [filename])
[res],[err] = win.ScreenCapture([source], 2)
[res],[err] = win.ScreenCapture([source], 3, [buffer])
```

source: Table/String/Number. Table has coordinates of rectangle {left,top,right,bottom}. String has "PRIMARY" (monitor), "FOREGROUNDWINDOW" or "VIRTUALSCREEN". Number is a window handle.

dest: Number. 1 = file; 2 = clipboard; 3 = buffer.

© 2008 Promixis, LLC

filename: String. Full path and filename of destination file.

buffer: String. Will contain the bytes of the image.

ClipboardGetText, ClipboardSetText, ClipboardPasteWindow
```
[text] = win.ClipboardGetText()
[res] = win.ClipboardSetText([text])
[res] = win.ClipboardPasteWindow([handle])
```
text: String. Text to transfer to/from clipboard.

handle: Number. Window handle of window to paste to.

12.34.12 Network Functions

ListNetworkInterfaces, IsNetworkInterfaceConnected, GetIPInfo
```
[iflist] = win.ListNetworkInterfaces()
[status] = win.IsNetworkInterfaceConnected([ifname])
[ip],[hostname],[ifcount] = win.GetIPInfo([ifnum])
```
iflist: Table. List containing a table per network interface. Each entry has two fields keyed "Name" and "Description".

ifname: String. Name of adapter per the above table.

status: Number. -1 = Disabled; 0 = Disconnected; 1 = Connected.

ifnum: Number. 0 is first interface, 1 is second etc.

ifcount: Number. The number of available interfaces.

ip: String. IP address of network adapter.

hostname: String. DNS name of computer.

Example.
```
local x = win.ListNetworkInterfaces()
for i, t in ipairs(x)
   j = win.IsNetworkInterfaceConnected(t.Name)
    if j < 0 then s = "Disabled"
    elseif j > 0 then s = "Connected"
    else s = "Disconnected end
   print(t.Description, s, win.GetIPInfo(i - 1))
end
```

Ping
```
[pingtime] = win.Ping([address])
```
address: String. IP address or DNS name of remote computer.

pingtime: Number. nil or time to complete round-trip in milliseconds.

URLDownloadToFile, URLDownloadToMemory
```
[ret] = [callback]([written], [expected])
[written],[expected] = URLDownloadToFile([url], [filename], [callback])
[buffer],[written],[expected] = URLDownloadToMemory([url], [callback])
```
url: String. URL of file to be downloaded.

filename: String. Path and filename to download to.

callback: Function or nil. If supplied gets progress messages.

© 2008 Promixis, LLC

written: Number of bytes written (so far).

expected: Number of bytes expected.

buffer: String which will contain the content of the downloaded file.

ret: Callback can return nil to abort download or 1 to continue.

WakeOverLan

Sends a network message to wake up a LAN node. This will switch a PC on if its LAN card is powered and the wakeup on LAN feature is enabled.
`[ret], [err] = win.WakeOverLan([ip], [mac])`
ip: String. IP address to send wake-up message to.

mac: String. MAC address of LAN card which will receive the message in the format "00:01:02:03:04:05" where each digit is hexadecimal.

12.34.13 Miscellaneous Functions

GetCPUUsage, GetMemoryUsage
`[percent] = win.GetCPUUsage()`
`[tpm],[apm],[tpf],[apf],[tvm],[avm] = win.GetMemoryUsage()`
percent: Number. Percentage CPU occupancy in last 100 milliseconds.

tpm: Number. Total Physical Memory kb.

apm: Number. Available Physical Memory kb.

tpf: Number. Total Page File kb.

apf: Number. Available Page File kb.

tvm: Number. Total Virtual Memory kb.

avm: Number. Available Virtual Memory kb.

PlaySound, Beep
`[res] = win.PlaySound([filename], win.SND_FILENAME)`
`[res] = win.PlaySound([alias], win.SND_ALIAS)`
`win.Beep([freq], [timeout])`
filename: Sting. Path to sound file (.WAV).

alias: String. Windows system event alias.

res: nil on error.

freq: Number. Frequency in Hertz.

timeout: Number. Duration on Milliseconds.

GetLastError
`[error] = win.GetLastError()`
error: Number. The number of the last Windows error to occur on the current thread.

MessageBox
`[res] = win.MessageBox([message], [title], [MB])`
message: String. Message to show in box.

© 2008 Promixis, LLC

title: String. Title of window.

MB: Number. One of these - win.MB_YESNO; win.MB_YESNOCANCEL; win.MB_RETRYCANCEL; win.MB_OKCANCEL; win.MB_OK; win.MB_CANCELTRYCONTINUE; win.ABORTRETRYIGNORE. Plus one of these - win.MB_ICONASTERISK; win.MB_ICONERROR; win.MB_ICONEXCLAMATION; win.MB_ICONHAND; win.MB_ICONINFORMATION; win.MB_ICONQUESTION; win.MB_ICONSTOP; win.MB_ICONWARNING.

ConvertImageFormat, SetDesktopBitmap

```
win.ConvertImageFormat([inputfile], [outputfile], [outputmime])
win.SetDesktopBitmap([filename])
```

inputfile: String. Path and name of input file (JPG, GIF, TIF or PNG).

outputfile: String. Path and name of output file.

outputmime: String. One of "image/jpeg"; "image/gif"; "image/tiff"; "image/png".

filename: String. Path and name of BMP file to show on desktop.

HideTaskbar, ShowTaskbar

Hides or shows the Windows desktop taskbar.

```
win.HideTaskbar()
win.ShowTaskbar()
```

12.34.14 Registry

The registry object allows reading, writing and creation of Windows Registry keys and values. See also ptable Library.

WARNING: YOU CAN DAMAGE YOU SYSTEM BY SETTING INCORRECT VALUES, USE CAUTION!

```
[Reg] = win.CreateRegistry([HKEY], [key], [create])
[value], [error] = [Reg]:Read([name])
[error] = [Reg]:Write([name], [value])
[error] = [Reg]:Delete([name])
[error] = [Reg]:DeleteKey([key])
[keys] = [Reg]:ListKeys()
[names] = [Reg]:ListValues()
[Reg]:CloseKey()
[Reg]:Destroy()
```

HKEY: String. "HKLM"/"HKEY_LOCAL_MACHINE"; "HKCU"/"HKEY_CURRENT_USER"; "HKCR"/"HKEY_CLASSES_ROOT"; "HKU"/"HK_USERS" or "HKCC"/"HKEY_CURRENT_CONFIG".

key: String. The path to the root key to be used.

create: Number. 1 to create the key (if necessary) else 0.

name: String. The name of the value.

value: String or Number. The contents of the named value or nil for error.

error: String. The error that occurred or nil.

keys: Table of Strings. All the sub-keys of the root key.

names: Table of Strings. All the value names of the root key.

Example:

```
local reg, err, val
reg, err = win.CreateRegistry("HKLM", [[Software\Promixis\Girder\4\]])
```

© 2008 Promixis, LLC

```
if (reg == nil) then print(err); return end
val, err = reg:ListKeys()
if (val == nil) then print(err); return end
table.print(val)
reg:CloseKey()
```

12.35 xap Library

Creating a Class Device

Every xap device can (must?) have a heartbeat. This function will make the plugin automatically send out an heartbeat every 60 seconds. NB: The Lua support file described below creates a Class Device with the example parameters below and inserts a reference to it as xap.xApGirderDev.
`[ClassDev] = xap.NewDevice([class], [source], [uid])`
class: String. For example "promixis.girder".

source: String. For example "promixis.girder.server".

uid: String. Unique identifier, for example "FF430000".

Class Device Methods

The **Send** method will send out a xap message with the body text msg (meaning the header is generated for you). **Destroy** removes the Class Device.
`[ClassDev]:Send([msg])`
`[ClassDev]:Destroy()`
msg: String. The body text message.

Support Functions

The **Send** function sends out a message on the XAP bus, note you must include the message header as well. **CalcCRC32** calculates the CRC32 checksum from the msg string.
`xap.Send([msg])`
`[crc] = xap.CalcCRC32([msg])`
msg: String. The entire message including the header.

crc: Number. The CRC32 checksum value.

Lua script

There is a Lua support script which is loaded with the plugin. It is **%GIRDER%luascript\xap.lua** and supplies the following additional functions.

RegisterListener registers a function to be called when an incoming message has been parsed. You can register multiple functions to process different types of message.
`[listener]([message])`
`xap.RegisterListener([listener])`
`xap.UnregisterListener([listener])`
listener: Function. A function to be called when a message arrives.

message: Table. The parsed message (see example below).

```
   ["Name"] = "xap-header",
   ["Header"] = {
       ["uid"] = "FF776107",
      ["source"] = "ACME.Lighting.apartment:BedsideLamp",
       ["hop"] = "1",
       ["class"] = "xAPBSC.event",
       ["v"] = "12",
```

© 2008 Promixis, LLC

```
    },
    ["Body"] = {
        [1] = {
            ["Data"] = {
                ["state"] = "ON",
                ["level"] = "64/255",
            },
            ["Name"] = "input.state.1",
        },
        [1] = {
            ["Data"] = {
                ["state"] = "ON",
                ["level"] = "63/255",
            },
            ["Name"] = "input.state.2",
        },
    },
```

An example implementation of a listener for xAPBSC.event and xAPBSC.info is in the lua file.

Source: xAP Automation Plugin

12.36 zip Library

Requires Girder Pro.

Facilities for working with zip compressed archives from Lua. Archives may be created in files or in memory, archive directories may be scanned and items may be extracted to files or to memory.

Open and Close

```
[Zip] = zip.Open([filename], [mode], zip.FILE)
[Zip] = zip.Open([buffer], [mode], zip.MEMORY)
[Zip]:Close()
```

mode: Number. zip.OPEN or zip.CREATE

filename: String. Filename and path. If creating, path must exist already.

buffer: String. Buffer which contains/will contain the zipped data.

Examination

```
[items] = [Zip]:Count()
[archname], [FILE_ATTRIBUTE], [compsize], [size] = [Zip]:Get([item])
[item], [err], [msg] = [Zip]:Find([archname])
```

items: Number of entries in the zip directory.

item: Number. References an item in the directory 1..items.

archname: String. Filename and relative path in archive.

FILE_ATTRIBUTE: Number. Bitwise OR of any of the following - win.FILE_ATTRIBUTE_READONLY; win.FILE_ATTRIBUTE_HIDDEN; win.FILE_ATTRIBUTE_SYSTEM; win.FILE_ATTRIBUTE_DIRECTORY; win.FILE_ATTRIBUTE_ARCHIVE; win.FILE_ATTRIBUTE_NORMAL.

compsize: Number. Compressed size in bytes.

size: Number. Uncompressed size in bytes.

© 2008 Promixis, LLC

Decompression
```
[res],[err],[msg] = [Zip]:ExtractFile([item], [filename])
[buffer],[err],[msg] = [Zip]:ExtractBuffer([item])
```
item: Number of item to extract.

filename: String. Path and filename to decompress and extract to.

buffer: String containing the decompressed extracted data.

Compression
```
[res],[err],[msg] = [Zip]:AddFile([archname], [sourcename])
[res],[err],[msg] = [Zip]:AddBuffer([archname], [buffer])
```
sourcename: String. Full path to file to be compressed and added.

archname: String. Name of file in archive including any relative path.

buffer: String. Buffer containing the data to be compressed and added.

Error Handling

The first return parameter is nil on an error and error identification is in the following parameters if provided -

err: Number. Error code number. zip.CORRUPT; zip.EXTRACTFAIL; zip.FILENOTFOUND; zip.FILEOPENFAILED; zip.FILEWRITEFAILED; zip.READFAILED; zip.WRONGMODE.

msg: String. Error message.

Source: Zip Extensions Plugin

© 2008 Promixis, LLC

Part XIII

13 Web Programming Reference

Requires Girder Pro.

The Girder Web Server Plugin provides a fully featured implementation of HTTP/1.1. It uses the same scripting language that Girder uses and in fact anything you can access in Girder from Lua you can access from the scripts inside webpages. In addition, it supports https secure, encrypted transmission and authentication, data compression, caching and virtual servers.

This section provides the information needed to create your own Girder websites. In addition, you will need good references for HTML and Javascript.

13.1 Building a Girder Website

Introduction

A website is simply a directory of files on the server which are made available, via the Web Server, for client browsers to download. The primary files are HTML files, a special format of text file which control the layout of the browser screen. The HTML files may also refer to other files such as images (PNG, JPEG or GIF).

The Website Files

The website directory can be anywhere in the server file system. It is specified in the HTTP Root field on the Web Server configuration page. Using the sample website, if the HTTP Root is specified as **C:\Program Files\Promixis\Girder\httpd** and the Hostname of the server is **Cruncher**, then a file called **index.html** can be accessed from a web browser using the URL **http://Cruncher/index.html**. In fact, **index.html** is the *default* filename, so this URL may be shortened to **http://Cruncher/** (most web browsers will manage with just **Cruncher**).

If you want the website to be visible over the internet as opposed to just on your private network, you will need a properly registered DNS domain name like **promixis.com**. You can then define further sub-domains such as **www.promixis.com** if you wish.

A simple HTML file looks like this.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
    <HEAD>
       <TITLE>Girder 4.0 Webserver</TITLE>
    </HEAD>
    <BODY>
        My Web Page
    </BODY>
</HTML>
```

HTML is a complex and versatile language and there are many good reference books available. There are also numerous software HTML editors on the market.

Complex websites can use hierarchies of sub-directories below the HTTP Root. For example a file in **C:\Program Files\Promixis\Girder\httpd\img\send_event.png** is available on the URL **http://Cruncher/img/send_event.png** (note that HTTP uses forward slashes, not backslashes). For URL references within an HTML file, you can specify just the path relative to

© 2008 Promixis, LLC

the URL of the current file, for example **img/send_event.png**.

For the secure Web Server, you must begin the URL with **https:** instead of **http:** and you must specify the domain or hostname exactly as specified on the Web Server configuration page. If you specify a non-standard port either for http or https, you must include the port number in the URL as well, for example **https://cruncher:1055/index.html**.

Directory level Access Control

In addition to general password protected access provided on the Web Server Configuration page, you can control access to specific directories below the HTTP Root. This is useful, for example, to create an administration system within a website otherwise open to the public. To do this, place a file named **___access.txt** in the directory to be restricted (note that this filename begins with three underscore characters, any file beginning this way will not be visible to clients).

The file should have the format.
```
Text describing the realm.
UserName1:Password1
UserName2:Password2
```
Any number of usernames and passwords may be supplied and the first line of the file is just documentation.

13.2 Server-side Lua Scripting

Introduction

You can embed Lua scripts in an HTML file. They are evaluated before the web page is downloaded to the web browser. You enclose the script in **<% ... %>**. Any return value is emitted into html. Web page files with embedded Lua scripting must have the extension **.lhtml** rather than **.html**.

Example:
```
<% local a = 10; a = a + 5; webserver:print(a) %>
```
When the page containing the above is downloaded, the entire string between and including the angle -brackets is replaced with the string "15".

The webserver Object

Lua scripts running in web pages have access to the full facilities of Girder Lua, but additionally they have access to a special object which has some useful methods.

The print Method

The **print** method works like the Lua global function of the same name except that the result is emitted into the HTML being returned to the client. It is inserted in place of the script text, with multiple print values being inserted in order of execution and before any return value.
```
webserver:print([val]...)
```
val: String, Number or Table or Object with tostring meta-method. The values are converted to strings and concatenated with a single space separating multiple values.

GetURL, GetHost, GetFilename, GetAddress

These methods provide basic information about the request and the file being served in response.
```
[url] = webserver:GetURL()
[host] = webserver:GetHost()
[filename] = webserver:GetFilename()
```

© 2008 Promixis, LLC

```
[address] = webserver:GetAddress()
```
url: String. The URL as requested, without the domain part (for example "/test.html").

host: String. The host or domain name part of the URL requested.

filename: String. The full server filename of the file served.

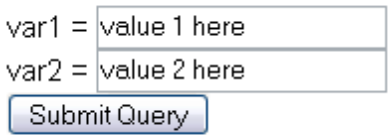
address: String. The IP address of the client machine that sent the request.

GetCGI

GetCGI gets the CGI parameters sent in the Get request URL or the body of a Post request.
```
[cgi] = webserver:GetCGI()
```
cgi: Table containing string keys and values derived from the CGI parameters.

Example

If the current web page contains a form element as follows.
```
<form method="get" action="processform.lhtml">
    var1 = <input type="text" name="var1" value="value 1 here"><br>
    var2 = <input type="text" name="var2" value="value 2 here"><br>
    <input type="submit">
</form>
```
This will show in the browser like.



When the user presses the **submit** button the values of the input field are packaged up and sent to the server which executes **processform.lhtml** and then returns it. Script in this file can then process the form in Girder.
```
<% table.print(webserver:GetCGI()) %>
```
Note that this file is send back and rendered in the browser, so it should also contain some form of HTML acknowledgement. To execute a simple script leaving the current web page intact, you can use Ajax (see next section).

GetHeader, SetHeader, GetBody

HTTP requests and responses include a header part and a body part. The header contains meta-data used by the browser or server which is usually in the form of key:value pairs separated by colon, but may also be a simple line of text with no colon. The body is the contents of the file being served, or for a Post request, CGI information. (Note that it is usually much easier to use **GetCGI** to access the body of a Post request).
```
[header] = webserver:GetHeader()
webserver:SetHeaderEx([name], [value], [replace])
webserver:SetHeader([headerline], [replace])
[body] = webserver:GetBody()
```
header: Table. String keys and values. Keys are the header names.

name: String. The header key.

value: String. The value of the header key.

headerline: String. Line to be added to the response header.

© 2008 Promixis, LLC

replace: Boolean or nil means false. If true a header of the same name is deleted first. Otherwise, duplicate names will result in multiple headers of the same name.

body: String. The full, encoded, body of the request (Post requests only).

GetCookies, SetCookie

Cookies are small files stored on the client computer to retain context information between Web Server requests. If you execute **SetCookie** when serving a web page to a client, then that cookie will be returned to the server with each subsequent request and can be recovered using **GetCookies**.

```
[cookies] = webserver:GetCookies()
webserver:SetCookie([name], [content], [lifetime], [path], [domain], [secure])
```

cookies: Table. String keys and values. Key is the cookie name and the value is the cookie content.

name: String. Name of cookie to set on client machine.

content: String. Contents of cookie to set on client machine.

lifetime: Number or nil for indefinite. Lifetime of cookie in seconds. Set to 0 to delete the cookie.

path: String or nil for current directory. Only requests for pages from this directory and below will see this cookie. "/" means all directories.

domain: String or nil for current domain. URL domain of requests which will receive this cookie. (".foo.com" - all domains in foo.com; "www.foo.com" - only the www domain.) Note that due to widespread abuse, many browsers are set to only accept cookies for the current domain, so this parameter is of limited use.

secure: Boolean or nil for false. If set true, the cookie will only be sent if the connection is secure (https).

SetStatus

SetStatus sets the HTTP status number and text of the response.

```
webserver:SetStatus([status], [reason])
```

status: Number (integer). The HTTP response number (e.g. 200 for OK or 404 for file not found).

reason: String. The HTTP response text.

SetGZIP

SetGZIP sets the compression state for the page. There is a default compression state depending on mime type (see %GIRDER%plugins\webserver\gzip-mime.txt), executing this method can override that default for the current file.

```
webserver:SetGZIP(onoff)
```

onoff: Boolean. true to compress, false (the default) to send uncompressed.

GetSSL
GetSSL returns true if this connection is encrypted and false otherwise.

GetMethod
GetMethod returns a string containing the transaction method, e.g. POST, HEAD or GET.

GetRoot
Returns the virtual webhost root directory.

© 2008 Promixis, LLC

13.3 Client-side Scripting with Ajax

Introduction

So far the Lua scripts we have embedded in web pages have all run within Girder on the server before the page was downloaded to the client browser. This technique is limited because in order to respond visually to user action in the browser, we have to download and replace the entire web page. This inevitably causes noticeable delays and undesirable screen flickering.

Most browsers are capable of executing scripts embedded in web pages locally on the client machine. Unfortunately few, if any, web browsers support Lua so it is necessary to work in a different, supported language when writing scripts to execute in the browser. Javascript (which is not the same as the Java language) is supported by almost all modern browsers and is recommended. Using Javascript, you can dynamically change visual elements on a web page in the browser without involving the server.

Ajax Technology

Ajax is a browser based technology that allows script running client-side to request and receive data from the server outside of the normal download-and-render mechanism of the browser. Client-side scripting can use the response data in any way it chooses. Ajax uses the same HTTP protocol as the browser to request and download files from the server, but does not render the downloaded file in the browser.

Ajax is implemented using browser plugins or objects accessed differently depending on the browser. The following Javascript will create an Ajax requester in Internet Explorer or a Mozilla based browser such as FireFox. This is then used to Post a request to the Web Server.

Do yourself a favor and get JQuery, this is a fantastic Javascript library that will make your life in Javascript land so much easier you'll wonder how you ever did without.

```
<HTML>
<HEAD>
<SCRIPT   src="jquery.min.js"   type="text/javascript"></SCRIPT>
<SCRIPT>
function  SendEvent(  Event,  Device  )
{
        $.get("ajax_sendevent.lhtml?event="+escape(Event)+"&device="+escape(Device));
    return false;
}
</SCRIPT>
</HEAD>
<BODY>
<a  href=""  OnClick='return  SendEvent("PREVIOUS",  "4000");'>CLICK  ME!</a>
</BODY>
</HTML>
```

The script on the server can be written in Lua in a **.lhtml** (ajax_sendevent.lhtml) file. Note that despite the name it is not actually necessary for this file to contain any HTML - in this case it just emits the reply data directly and this becomes the body of the response.

```
<%
       webserver:SetGZIP(false)
       webserver:SetHeader("Cache-Control: no-cache, must-revalidate")
       webserver:SetHeader("Expires: Fri, 30 Oct 1998 14:19:41 GMT")
       local table = webserver:GetCGI()
       if ( gir ) then
       gir.TriggerEvent( table['event'], table['device'], 0)
       end
```

© 2008 Promixis, LLC

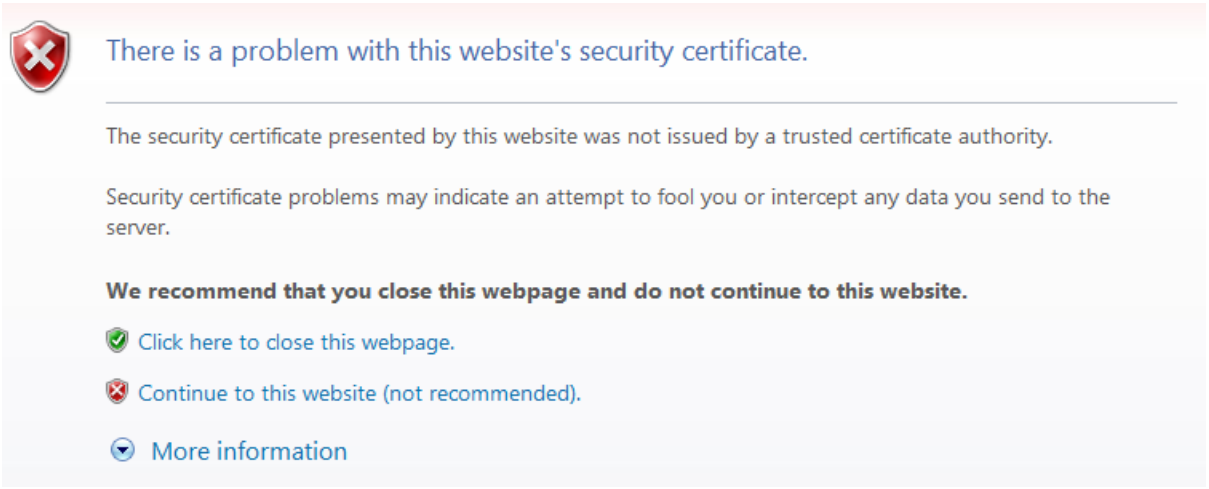
%>

In this example, we used Ajax to send a GET request.

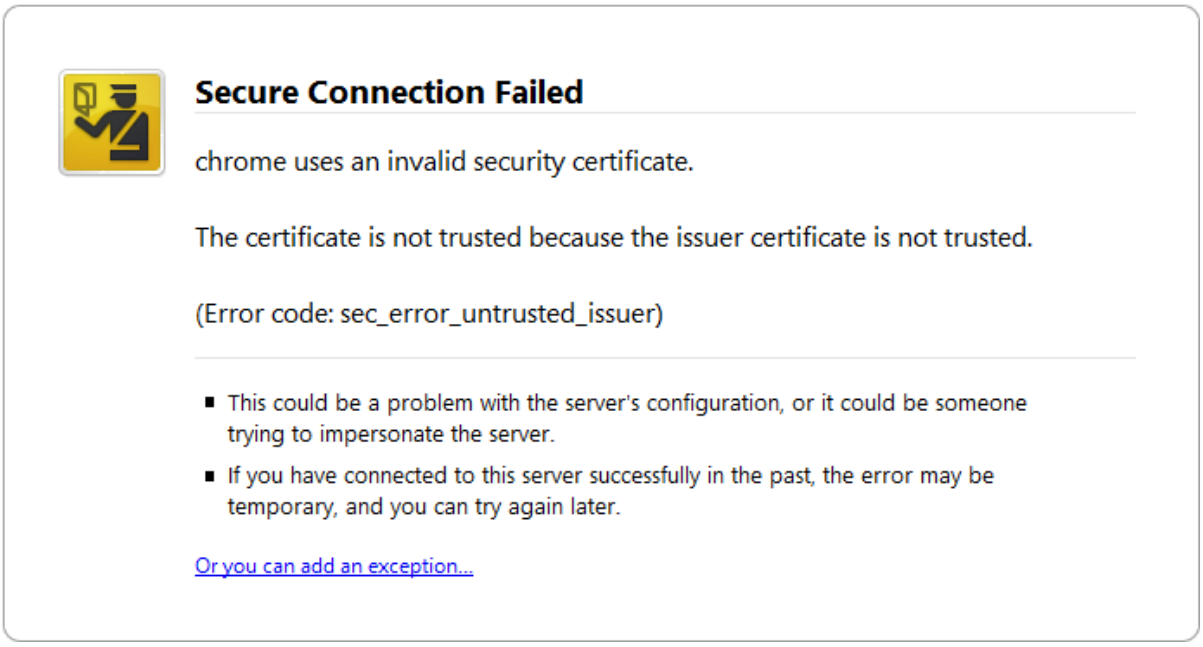
The example above can be found in the Girder\httpd directory called "events.html" and "ajax_sendevent.lhtml".

13.4 Installing SSL Certificate

When using Girder's HTTPS / SSL encrypted webserver you will probably encounter an error message like the one below for IE8



Or Firefox 3



These errors mean that the browser does not have the 'root' certificate that Girder is using to encrypt the pages with. You can import this certificate in the following ways

© 2008 Promixis, LLC

Internet Explorer

Open the Tools menu
Click on Internet Options
Click on the Content Tab
Click on the Certificates Button
Click on Trusted Root Certificates
Click import and follow the wizard, you can find the certificate in the file called "girder.crt". On Windows Vista this can be found in the C:\Users\<YOURNAME>\AppData\Roaming\Promixis\Girder\5\Plugins\Webserver\root directory.

FireFox 3
Open the Tools Menu
Click on Options
Go to the upper Advanced Tab / Icon
Click on the Encryption Tab
Click on View Certificates
Click on the Authorities tab.
Click on the Import button
Find the certificate file called "girder.crt". On Windows Vista this can be found in the C:\Users\<YOURNAME>\AppData\Roaming\Promixis\Girder\5\Plugins\Webserver\root directory.
Check the "Trust this CA to identify websites".

After you have imported the Certificate you will see the "Girder Signing Authority" in your Trusted Root / Authorities list. If you have multiple Girder computers on your network that all run SSL encrypted websites you can use the same root CA for all these servers allowing you to only import one root CA into your browsers. Simply take Girder.crt and PrivateKey.pem and place them in the 'root' directory on the other machines, overwriting both files. After that is done make sure you delete all machine specific files in the directory above the 'root' directory so Girder generates them anew with the correct root CA.

13.5 Advanced Web Server Configuration

These configuration options are for advanced users and can safely be ignored by most users.

All configuration files mention below can live in two places, first Girder will check your user's application data directory.

On Vista that is:

```
C:\Users\<YOUR      NAME>\AppData\Roaming\Promixis\Girder\5\Plugins\Webserver
```

On WinXP / 2000 that is:

```
C:\Documents and Settings\<YOUR NAME>\Application Data\Promixis\Girder\5\Plugins\Webserver
```

Note that this can be configured in Windows to be in a different spot, typically on a corporate network this is mapped to a network drive, so check carefully.

A quick way to find out is to type this into the Interactive Lua console:

```
print(gir.ParseString('%APPDATA%'))
```

Which will then print the location of that directory.

The second location Girder will check for the configuration file is the install directory:

© 2008 Promixis, LLC

```
c:\program    files\promixis\Girder\plugins\webserver
```

This can also be found by typing

```
print(gir.ParseString('%GIRDER%'))
```

into the interactive lua console.

Multiple SSL Certificates

Girder supports the use of multiple SSL certificates, one per IP address to be exact. By default all requests will get the SSL certificate configured in the webserver configuration screen unless an entry for that IP address is found in the SSLHosts.txt file.

This file has the following format:

<IP ADDRESS>=<Host name for SSL Certificate>

for example

192.168.0.42=chrome
127.0.0.1=localhost

Now when you try to access the Girder https://localhost the certificate will match the hostname and the browser should not complain. Note that you will have to script reset to have this take effect. Also read on how to install an SSL certificate into your browser.

Virtual Servers

The Girder Web Server can provide several different websites simultaneously. Each site has its own HTTP Root Directory accessed using a different domain name. The file **hosts.txt** contains multiple lines each of which associates a domain name with an HTTP Root Directory. If the domain of a request is not matched in the hosts file, the HTTP Root specified on the configuration page is used.

The hosts.txt file has a format like this.
`www.foobar.com=C:\websites\foobar`
`manager.foobar.com=C:\websites\manager`
Of course you must also arrange for these names to be registered in DNS, translate to the IP address of the server and be passed by any firewalls.

Mime Types

The file extension to mime type translation is dictated by the file mime.txt it is formatted as follows

.html=text/html
.htm=text/html

GZIP Configuration

GZIP compression speeds up data transfers over the network. To enabled compression for certain mime types use this file the format is as follows:

text/*=true
application/x-javascript=true
audio/*=true

© 2008 Promixis, LLC

© 2008 Promixis, LLC

Part

14 Programming the Girder UI

This section covers adding elements to the Girder User Interface using Lua. You can add Actions, Conditionals and Events which are then available for use in the Girder Tree. You can also add pages to the Setup Dialog, including more complex applications in the Automation section.

With the exception of adding Events, these capabilities depend on the Tree Script Plugin and use the Dynamic User Interface Designer. Many of the features shipped with Girder, particularly the Pro editions, are implemented using the techniques described in this section and additional insight can be gained by examining this code. The XML files in **%GIRDER%plugins\UI** can be opened in the DUI Designer as examples of DUI design. The LUA files in **%GIRDER%plugins \treescript** script the behavior behind the DUI pages. When the Lua system is started (on Girder startup or on script reset) all the Lua files in this directory are loaded and run. The Lua files return the name of the associated DUI XML file which is parsed by Girder and used to create the user interface pages.

The following sections provide details.

- Scripting Actions and Conditionals.
- Scripting Events.
- Scripting Configuration Pages.
- Advanced User Interface Scripting.
- DUI Page Location.

The final section details techniques used to create advanced interaction on DUI pages and is particularly relevant to the creation of Automation applications.

NOTE: You can also program the Girder UI by writing a Plugin using the "C" or "C++" programming languages. This is covered in the separate **Girder API** manual.

14.1 Scripting Actions and Conditionals

Overview

You can add Actions and Conditionals to the Task Panes on the left of the Girder main window (Expert Interface). These can later be used in the Girder Tree in the same way as the built-in Actions or Conditionals. You provide a Lua function to set the default values of the Action parameters. You also layout the first tab on the editor dialog to allow the parameters to be edited. Girder automatically stores these parameters in the Tree.

When the Action is triggered, or when the Conditional needs to be evaluated, Girder executes a Lua function which you provide.

Selecting Action Subtypes

Each Action, Conditional or Configuration page added by TreeScript must have a unique **Action Subtype** within that group (an Action may have the same Subtype as a Conditional and a Configuration page etc.). Check the Promixis forums to obtain and register unique numbers if you intend distribution. For private scripts, just check the Variable Inspector expanding the **treescript** table and then the **Action**, **Conditional** or **Config** sub-table. The index numbers of the next level are the subtypes already in use.

Designing the User Interface

The user interface (Action and Conditional editor pages) is created in the DUI Designer, a

© 2008 Promixis, LLC

separate application in the Girder group on the Start menu. First create a new file and save it to *your-script-name*.**XML** in the **%GIRDER%plugins\UI** directory. Use the Designer **Edit** menu to **Add Action Group** (if the script will add Actions) and **Add Conditional Group** (if the script will add Conditionals).

Add one DUI form per Action or Conditional to the appropriate group. It is also possible to add sub-groups to the groups and then add the forms to those. This group structure will show in the Girder task boxes. To add an Action or Conditional to an existing group (i.e. **Girder**), add that group to the Action Group in the designer using the **Default Groups** sub-menu of the **Edit** menu. Note that groups are matched by the GUID, not the name, so just adding another group and calling it "Girder" will generate two Girder groups in the task box. Set the **Plugin Number** of each form to 243 (the Tree Script Plugin number) and **Action Subtype** to the subtype selected above.

Next arrange the required controls on the DUI forms. Set the **Field** property of the control to an Action parameter (**iValue1-3**; **bValue1-3**; **sValue1-3** or (for numbers only) **lValue1-3**).

Save the changes to the DUI XML file.

Creating the Script File

Create a new text file using any text editor (notepad for example) and save it with the .Lua extension in the **%GIRDER%plugins\treescript** directory.

The lua file should return the name of the DUI XML file discussed above which must be in the **%GIRDER%\plugins\UI** directory.

Before returning, the Lua file should install some functions in the **treescript** global table.

Scripting Actions

The minimal case is installing a single Action with automatic data transfer from its editor page.

```
treescript.Action[2] = {}
treescript.Action[2].OnAction=function(Action, Event)
   print ('Action[2]: OnAction Called', Event.EventString)
   print (Action.sValue1, Action.bValue1)
   return true, "Example Action Triggered"
end
return "example.xml"
```

The **Action Subtype** (2 in this example) must match the value set for the Action Editor form defined in the DUI Designer. The TreeScript Plugin runs this script on startup (and on script reset). This installs the **OnAction** function and returns the name of the DUI file. TreeScript then loads the DUI pages from this file which results in the new action getting installed in the Actions Task Box. When an instance of this Action is put into a GML tree, the Editor Form defined by the DUI file shows to enable the user to set the parameters for the action. When the action is triggered, TreeScript will run the **OnAction** function.

The **OnAction** function receives an **Action** table containing the values the user entered into the Action Editor dialog, and an **Event** table which contains the details of the event which triggered the action. The function returns a boolean, **true** to continue processing this event as normal, **false** to terminate processing; and a string which gets displayed in the Girder status bar.

The **Action** table keys are **iValue1-3**; **bValue1-3**; **sValue1-3** (strings) and **lValue1-3** (number). There are also **FileGUID1-3** (string) which contains the GUID of the GML file containing the linked Action if a **Link Button** control is bound to the corresponding **lValue1-3**.

The Event table keys are **EventString** (string); **Device** (number) and **Modifier** (number).

It is good practice to provide default values for the action parameters independently of the UI code. This means that the user does not need to edit or apply values before using the Action.

© 2008 Promixis, LLC

This can be achieved by installing an **OnDefaults** function.

```
treescript.Action[2].OnDefaults=function(Index, Action)
   print('Action[2]:OnDefaults Called')
   Action.sValue1='this is awesome'
end
```

Scripting Conditionals

Similarly, a single Conditional with automatic data transfer looks like this.

```
treescript.Conditional[2] = {}
treescript.Conditional[2].OnAction=function (Action, Event)
 return true -- Or false
end
return "example.xml"
```

Here the boolean return from OnAction is the result of the Conditional. It should be tightly coded as it will get called often. Again, the index number (2) must match the **Action Subtype** set for the Conditional Editor DUI form.

OnDefaults is also available for Conditionals.

```
treescript.Conditional[2].OnDefaults=function (Index, Action)
   print('Conditional[2]:OnDefaults Called')
   Action.sValue1='this is awesome'
end
```

14.2 Scripting Events

Overview

You can add your own Event Devices and declare Event Strings which can then be selected from the drop-down list on the Event Editor. You can use a script running "in the background" to generate these events. You may also want to add an associated configuration page to configure and enable or disable this event generation (this aspect is covered in the next section).

Declaring Event Devices and Event Strings

You declare an Event Device by adding an entry to the **dui.Devices** table. The entry is keyed on the Device Number and contains a string which will appear in the Event Editor dropdown.

Note that the Device Number must be unique. For a private script, check the table and pick an unused number. If you intend sharing the script, you should apply for a number to be allocated on the Promixis forum.

You declare Event Strings for an Event Device by creating a table keyed with the Device Number in the **dui.Events** table. Then you add a numerically indexed entry to that table for each Event String.

Generating Events

You generate events using the **TriggerEventEx** function in the gir Library. Generally you need a piece of code running continually in the background which checks periodically if something has happened yet and triggers the event at the right time. You can use the facilities of the thread Library to create a background process to do this, or use **CreateTimer** in the gir Library to create a polling routine.

Example

This example generates events when another computer comes on or off the network. It sends a "ping" test message every minute to make the test. You can put this script in a file in the **%**

© 2008 Promixis, LLC

GIRDER%luascript\startup directory or in a Scripting Action triggered by a **Girder Events/ ScriptEnable** event.

```
local target = "thecomputer"  -- Name or IP of computer to test.
local devno = 22200
dui.Devices[devno] = "Pinger"
dui.Events[devno] = {}
table.insert(dui.Events[devno], "Online")
table.insert(dui.Events[devno], "Offline")
local online = nil
local check = function()
   if win.Ping(target) then
       if not online then
          gir.TriggerEventEx("Online", devno, 0)
       end
       online = true
   else
       if online ~= false then
          gir.TriggerEventEx("Offline", devno, 0)
       end
       online = false
   end
end
local t = gir.CreateTimer("", check, "", true, true)
t:Arm(60000)
gir.LogMessage("Pinger", "Started", 3)
```

14.3 Scripting Configuration Pages

Overview

You can add to the Settings (Preferences) dialog, either individual pages (tabs) on the Plug-in Settings group or whole groups of pages accessed from the Automation task box. The former should be used for the configuration of event generation scripts or hardware interfaces and the latter for the creation of additional Home Automation Applications.

Tip: The techniques described in this section are generally sufficient for device configuration pages. The more sophisticated interaction often required for Automation applications is covered in the next section.

Selecting Action Subtypes

Each Configuration page added by TreeScript must have a unique **Action Subtype** within that group. Check the Promixis forums to obtain unique numbers and register them if you intend distribution. For private scripts, just check the Variable Inspector expanding the **treescript** table and then the **Config** sub-table. The index numbers of the next level are the subtypes already in use.

Designing the User Interface

In the DUI Designer, add the "Device Config Group" and/or the "Automation Config Group" to the form tree. For the Automation Config Group, you must add exactly one further Group level below it. You should also assign an Icon to these Groups by right-clicking and selecting an Image on the context menu. The name and icon of the Group at this level becomes the entry in the Automation Task Box and the DUI Forms added to these Groups become the tabs. For the Device Config Group, DUI Forms must be added directly to the Group.

Set the Plugin Number field to 243 and the Action Subtype field to the number selected above for each DUI form you add.

© 2008 Promixis, LLC

Next arrange the required controls on the DUI forms. Set the **Field** property of the control to "Registry Number" or "Registry String" and the Name/Value property to the name of the registry key that will be used to store this setting. Alternately, you can bind the control to the action parameters as for Actions or Conditionals, but that requires more work in the scripting as described below.

Save the changes to the DUI XML file.

Creating the Script File

A simple configuration page just needs an **OnApply** function which will get called when the user presses the apply button on the Config page. (This can be reduced to an empty stub if configuration changes are only applied at startup or script reset, but it must be present or the UI will not show).

```
treescript.Config[1] = {}
treescript.Config[1].OnApply = function (Index, Action, Controls)
   print('Config[1]: OnApply Called')
end
return "example.xml"
```

When **OnApply** is called, Girder will have already saved any changes made to the registry. The **OnApply** function should obtain the settings from the registry and apply them as it would do at startup.

```
local Setup = function()
    -- Get the settings from the registry and apply them
end
treescript.Config[1] = {}
treescript.Config[1].OnApply = function (Index, Action, Controls)
    Setup()
end
Setup()
return "example.xml"
```

In the above pattern, the Setup function is called when the script is installed in order to recover the initial settings from the registry, and also in **OnApply** to change the settings.

It is also possible to use the Action parameters to bind a Config page to its script. In this case the values are not automatically persisted and the script needs to to that explicitly. Another script function is needed to load values into the Action parameters prior to showing the config page.

```
treescript.Config[1] = {}
treescript.Config[1].OnShow = function (Index, Action, Controls)
   print('Config[1]: OnShow Called')
   --Transfer current/default settings from storage to Action table
end
treescript.Config[1].OnApply = function (Index, Action, Controls)
   print('Config[1]: OnApply Called')
   --Persist and implement changes in Action table
end
return "example.xml"
```

In this case, in **OnShow**, the current configuration or the default configuration is loaded into the Action parameters (for example **Action.iValue1**). The configuration page appears showing these values. When the user presses **Apply**, the **OnApply** function gets called to persist and apply any changes.

14.4 Advanced User Interface Scripting

Overview

The techniques described above allow simple Action, Conditional and Configuration pages to be created with minimal scripting. However the pages are limited to simple parameter editing with

© 2008 Promixis, LLC

no interaction between controls and only the simple validation provided by the controls.

With a little more scripting work, it is possible to program controls on the form, take over data transfer between storage and the controls, change the DUI template controlling the page layout and even update the form for external changes while it is showing.

These techniques can be used for all types of DUI form, but they are most useful for Configuration forms and particularly Automation applications.

Designer Capabilities

To use these facilities, the controls on the DUI form should be bound to Lua variables rather than to action parameters or registry keys. Set the **Field** parameter for the control to **Lua** and the **Name/Value** parameter to a unique name. If action needs to be taken when the value of the control is changed by the user, check the **Event** box and enter values in the two numerical boxes to the right of this checkbox.

Lua-bound Controls

The **Controls** parameter contains a sub-table for each of the controls keyed on the Control Lua **Name** set in the DUI designer. The keys in the sub-table depend on the control type. For example, if there is a control of CheckBox type with Lua name "power", its checked state is controlled by and read from the variable **Controls.power.Checked**.

All control types have boolean fields **Visible** and **Enabled**, and the string field **Hint** which is the tooltip text shown when the mouse hovers over the control. The controls have the following additional fields, depending on type.

Control Type	Field	Type	Field	Type	Field	Type
BrowseButton	Caption	string	Filename	string		
CaptureButton						
CheckBox	Caption	string	Checked	boolean		
CodeEditor	Text	string				
ComboBox	Strings	string	ItemIndex	number		
CommandButton	Caption	string				
DirectoryBrowser	AllowFunc	number	Caption	string	Directory	string
EditBox	Text	string				
EditComboBox	Strings	string	Text	string		
GroupBox	Caption	strings				
HTMLLabel	Caption	string				
Image	Filename	string				
Label	Caption	string				
LinkButton	Caption	string	FileGUID	string	ID	number
ListBox	Strings	string	ItemIndex	number		
Memo	Text	string				
RadioGroup	Caption	string	Strings	string	ItemIndex	number
Slider	Min	number	Max	number	Position	number
SpinButton	Min	number	Max	number	Position	number
TargetButton	Caption	string				
VolumeButton	Caption	string	Address	string	Channel	number

© 2008 Promixis, LLC

Strings parameters are lists of strings separated by CR.

For lua-bound controls, **OnShow** and **OnApply** should be used to initialize the controls and to apply any changes made by the user.

```
treescript.Config[1].OnShow = function (Index, Action, Controls)
   print('Config[1]: OnShow Called')
   --Set the initial state of the Controls
end
treescript.Config[1].OnApply = function (Index, Action, Controls)
   print('Config[1]: OnApply Called')
   --Take action according to the changed values of the Controls
end
```

Control Events

If a control has its **Event** tickbox checked in the designer, **OnEvent** gets called when any of the fields of that control change.

```
treescript.Config[1].OnEvent = function (Index, Action, Controls, Control, ID1, ID2)
   print('Config[1]: OnEvent Called', ID1, ID2)
end
```

Control is the table for the specific control raising the event and **Controls** contains all of the controls on the form hosting that control. **ID1** and **ID2** are the two identification numbers set against the event in the DUI designer.

A typical use of Control Events is to hide or reveal some controls based on the state of the control generating the event.

Apply Management

Girder maintains a "changed" flag for each UI page. This flag gets set whenever the state of any of the controls on the page is changed. For Action and Conditional editors, this flag controls whether the **Apply** button is enabled and whether the **OnApply** function gets called when the **OK** or **Apply** button is pressed. For the Configuration pages, the **Apply** button is enabled when the flag for one or more pages is set. When the **Apply** button or the **OK** button is pressed, the **OnApply** function is called for all pages with the changed flag set.

This system has problems with complex pages with lua-bound controls. Often changes are just informational and do not require application and sometimes refreshing a control from Lua will set the flag inappropriately.

There is a function to manually set or reset the changed flag.

```
treescript.SetChanged([index], true)  -- Sets the flag
treescript.SetChanged([index], false) -- Resets the flag
```

index: Number. The index of the page (use the Index parameter of OnChanged).

Best practice is to maintain an internal changed flag as a local variable in the script and then to use **SetChanged** with this flag at the end of the **OnEvent** function. It may be necessary to enable event generation for all controls. A good standard is to use **ID1** = 0 and **ID2** = 0 to indicate an event which requires no action other than changed flag maintenance.

Asynchronous Update

In some cases it may be desirable to update the controls on a form other than in response to user action on the form. For example, a control may display a live clock or the status of something that can change while the form is showing.

The **Index** parameter (**OnShow**, **OnApply**, **OnDefaults**, **OnEvent**, **OnClose**) refers to the global table **dui** which contains the **Controls** tables of all the forms currently open. In this table, the

© 2008 Promixis, LLC

control fields have the full name **dui.Forms[***Index***].Controls.***name***.***field*.

Cache the Index value in **OnShow** and then use it with the **dui** table to manipulate the Lua control tables. Then call **treescript.UpdateDUIPage** to make the visible form reflect the changes made. Set the cached Index to **nil** in **OnClose**.
`treescript.UpdateDUIPage([index])`
index: Number. The index of the page to be updated.

```
treescript.Config[1] = {}
local inx = -1
treescript.Config[1].OnShow = function(Index, Action, Controls)
    inx = Index
end
--This is an example - it would be called from other code or on a timer
--for periodic update.
treescript.Config[1].Update = function(time)
    if inx < 0 then return end
   dui.Forms[inx].Controls.time.Text = time
   treescript.UpdateDUIPage([Index])
end
treescript.Config[1].OnClose = function(Index, Action, Controls)
    inx = -1
end
return "example.xml"
```

Changing the form

OnEvent and **OnShow** functions may optionally return the GUID string of another DUI form, in which case the window will switch to displaying that form in place of the current one.

There is also a library function **treescript.GotoDUIPage** which can change the page in any DUI form that is currently showing.
`treescript.GotoDUIPage([index], [guid])`
index: Number. The index of the page to be updated.

guid: String. The GUID of the form to switch to (this can be copied from the Designer. It is the long string below the **Action Subtype** in the **Form Settings** task box.

A form which is intended to be used as a secondary page in this way (i.e. it should not get its own entry in the Action or Conditional tree or its own Config tab) should be set with **Action Subtype** = -1.

14.5 DUI Page Location

Dynamic User Interface (DUI) pages can show up in 4 different areas of Girder's interface. Depending on how the DUI XML was prepared in the Promixis DUI Designer it will show up in one of the following:

- Plugin Settings Tab.
- Settings Window.
- Component Window.
- Action / Conditional Window.

A few DUI rules:

- DUI files are created with the DUI editor.
- DUI files live in the plugins/ui directory.

© 2008 Promixis, LLC

- A DUI form must have a unique GUID (a string like {6D7EFA72-B273-49ED-88D8-1D8FAE8B72A9}), so if you are copying a DUI from an existing XML make [b]very[/b] sure you do not have this GUID in use somewhere else, as either the other DUI page will no longer show up or yours will not.
- A DUI that is controlled by a plugin must have the plugin number filled out corresponding to the plugin number
- A DUI that is controlled from Lua must use plugin number 243 and for the action subtype a unique ID which is assigned by Promixis, (see our forum).
- If you need to edit the GUID in the DUI Designer hit F12 (Once) to unlock the GUID field.
- For a Lua controlled DUI for to show up the corresponding Treescript script (plugins/ treescript) must be loaded as well.

14.5.1 Plugin Settings Tab

This location is reserved for any settings related to a plugin. To add a page to here open the DUI Designer and select 'Edit->Add Device Config Group'. Right click this new group and select 'Add DUI form'. In the plugin number box enter the plugin number of the plugin this DUI is for.

14.5.2 Settings Window

There are a few different locations on this window that you can display a DUI page. First the 'Main' Drawer.

- 'Add Automation Config Group' or create a group with GUID {E56D7AFE-172F-11DA-8CD6-0800200C9A66}.
- Create a sub group with guid {D5FE8A7B-87B7-4CA1-A7B7-278BAEBECF78} and caption "Main"
- Create a sub group with guid {DF97879D-C913-4C3D-803D-CC34F4AC8061} and caption "General"
- Now create the DUI page of your choice.

To show up an any other drawer pick newly generated GUIDs.

14.5.3 Component Window

In the component manager there are a few predefined drawers.

Component Manager: {B7F07512-9EDB-4AD1-B13A-84AE241DDDEB}
RF Devices: {32A56996-C602-4595-83B4-C8F37804605C}
Schedule/Timed Events: {1F74FDF0-B026-457B-AC5D-911388900880}
Misc: {A6A56996-C602-4595-83B4-C8F37804605C}
Communications: {8A84A0AD-A3C0-4006-AC8C-27FDAC3B67A0}
Security: {22A0A9D8-27EC-4C77-87BC-890B115D659B}
Audio/Video: {A6A56996-C602-4595-83B4-C8F378046123}
HVAC: {A6A56996-C602-4595-83B4-C8F378046123}
Lighting/Appliances: {509D5D80-E747-4B36-B2C9-DB2DB04C251B}

14.5.4 Action / Conditional Window

To create an action form first create an action node. The create the appropriate subgroup, there are a few predifined groups available from 'Edit->Default Groups'.
The conditional forms work the same.

© 2008 Promixis, LLC

Part

15 Plugins Reference

This section documents all of the Plugins shipped with Girder. The Configuration Page is described if present. If the Plugin generates events, these are described. If the Plugin provides Actions, Conditionals or Lua Libraries, the relevant topic is identified in those reference sections.

- Audio Mixer (Sound) Plugin.
- Communication Server Plugin.
- CopyData Plugin.
- DLPort IO Extensions Plugin.
- Device Notify driver Plugin.
- Diamond Key Plugin.
- Generic Internet Plugin.
- Generic Serial Plugin (Girder Pro).
- Generic X10 Remote Plugin.
- Global Cache Plugin.
- IRTrans Plugin.
- Keyboard Plugin.
- Lua Misc Function Library Plugin.
- Monitor APM Extensions Plugin.
- Mouse Control Plugin.
- PowerLinc USB Plugin.
- Scheduler Plugin (Girder Pro).
- SendMessage Plugin.
- SimpleTimer Plugin.
- SlinkE Plugin.
- Streamzap PC Remote Plugin.
- TaskCreate Plugin.
- TaskSwitch Plugin.
- Tree Script Plugin.
- UIR/IRman/IRA/CTInfra/Hollywood+ Plugin.
- USB-UIRT driver Plugin.
- Web Server Plugin (Girder Pro).
- Window Conditionals Plugin.
- X10-CM1X Plugin.
- xAP Automation Plugin
- Zip Extensions Plugin (Girder Pro).

15.1 Audio Mixer (Sound) Plugin

Generates events when the computer's audio mixer settings (which includes the master volume and mute controls) change. Provides Lua functions for changing the settings from script.

Note: Actions in the built-in Volume Group provide an alternative method of changing the settings.

Configuration Page

The Configuration Page is described in the next section.

© 2008 Promixis, LLC

Events Generated

If events are enabled on the configuration page, an event is raised whenever a control changes (whether that change originated from Girder or any other source). The event string is in the form *device*:*destination*:*source*:*control*. The device, destination, source and control are also passed individually in pld1 through pld4. Generally these address numbers are 0..max, but source may be -1 which indicates the master control for that destination.

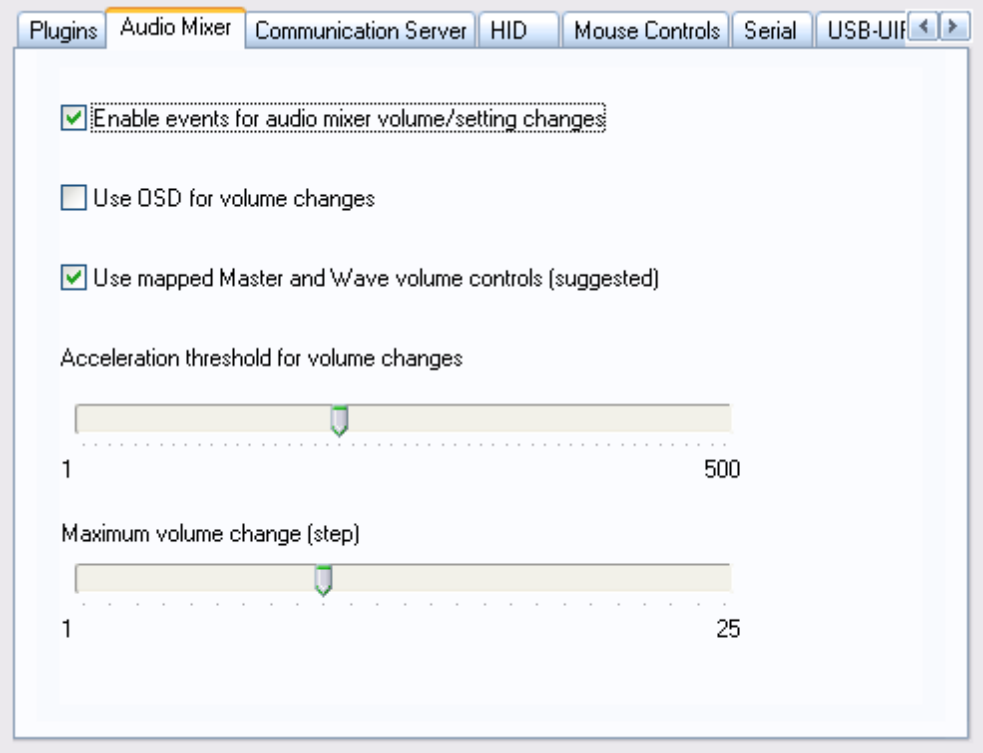
You can decode these event strings by temporarily adding a Change Volume Action to the Girder Tree and calling up the Audio Picker from its Action Editor. The Audio Picker topic also provides a detailed description of the mixer addressing system.

Lua Extensions

Adds the mixer Library.

File: AudioMixer.dll Device: 144.

15.1.1 Audio Mixer Configuration



Enable events: Tick to raise events whenever a control changes.

Use OSD: Tick to flash a display on the screen when a control changes.

Use mapped Master and Wave volume controls: Automatically link Girder Standard Events to the Master and Wave volume controls so Standard Controllers can control these functions.

Acceleration threshold: milliseconds after which the volume increment increases.

© 2008 Promixis, LLC

Maximum volume change: units to change volume by after the acceleration threshold.

15.2 Communication Server Plugin

Provides a two-way binary TCP protocol for communicating between instances of Girder on different machines, between Girder and NetRemote and between Girder and other conforming clients.

Configuration Page

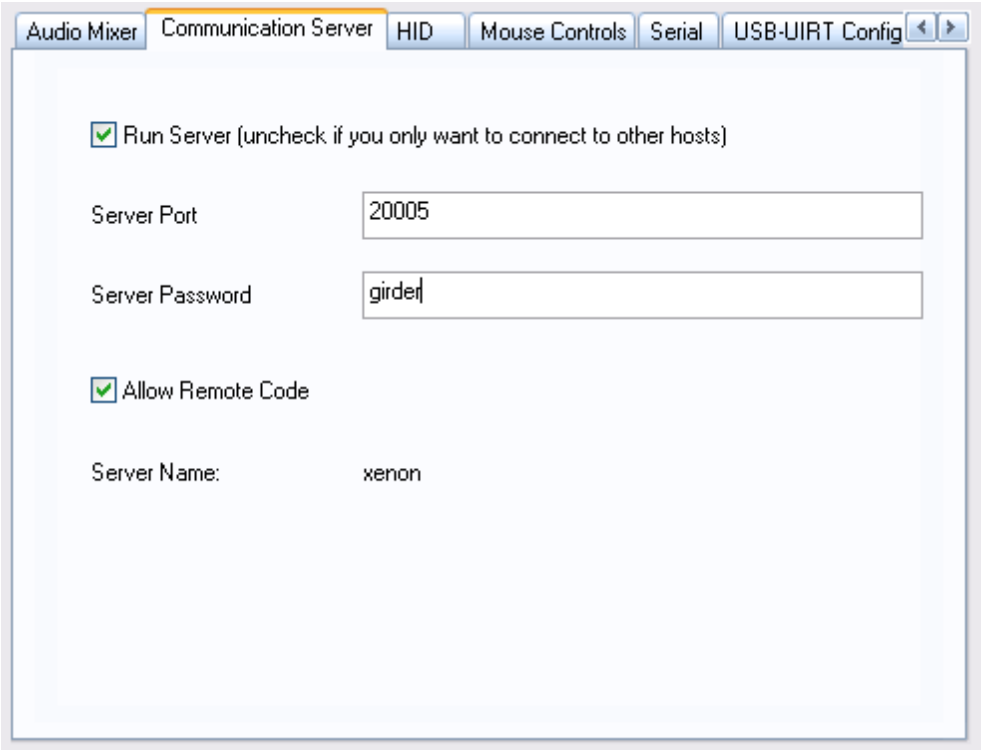
The Configuration Tab is described in the next section.

Lua Extensions

Provides the comserv Library. Supports the Girder2Girder Library and the NetRemote Library.

File: ComSvr.dll Device: 232.

15.2.1 Communications Server Configuration



Run Server: Tick if this Girder should act as a server as well as a client (i.e. Should accept connection requests).

Server Port: The IP port for the server to listen on (usually can be left unchanged, but you may want to change it in the event of a clash or for firewall reasons).

Server Password: You can change this to a hard-to-guess string for added security, but this will complicate setting up NetRemote and Girder to Girder connections. We recommend leaving it at the default "girder" until everything is working and then "locking it down".

© 2008 Promixis, LLC

Allow Remote Code: Tick to allow other Girder instances to send Lua code to this one for execution.

Server Name: The name of this server, provided for information when setting up NetRemote and other Girder machines to network with this one.

15.3 CopyData Plugin

This plugin provides the Copy Data action which can be used to send data to some programs using WN_COPYDATA messages.

Actions Provided

Adds the Copy Data Action.

File: copydata.dll Device: 113.

15.4 DLPort IO Extensions Plugin

This plugin enables Girder to read and write the hardware ports of the PC. These are used to interface with peripherals on the motherboard and on expansion boards at the lowest level and hence are highly technical.

Port IO is dangerous! Do not use unless you know exactly what you are doing!

The plugin requires the PortIO driver which can be downloaded from www.driverlinx.com. The Lua functions documented below will only be available if PortIO is installed.

Lua Extensions

```
[value] = PORTIO_ReadBYTE( [portnumber] )
[value] = PORTIO_ReadWORD( [portnumber] )
[value] = PORTIO_ReadDWORD( [portnumber] )
PORTIO_WriteBYTE( [portnumber], [value] )
PORTIO_WriteWORD( [portnumber], [value] )
PORTIO_WriteDWORD( [portnumber], [value] )
```

File: LPortIO.dll Device: 61.

15.5 Device Notify driver Plugin

Generates events when the media in removable drives changes.

Events Generated

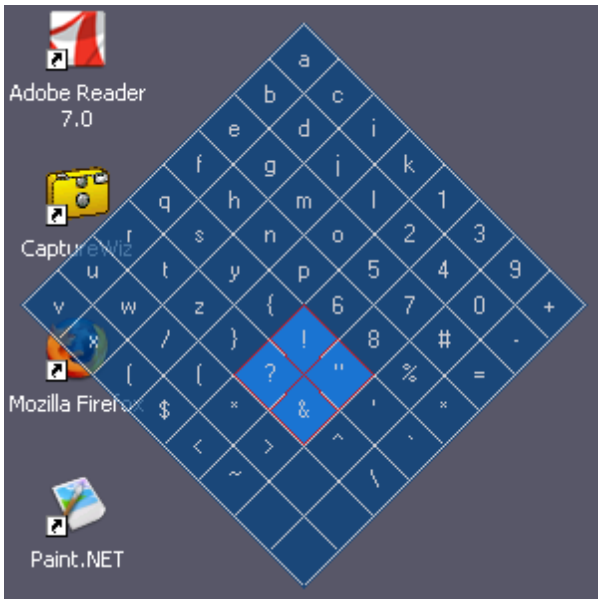
The event string has the form **Drive:*x* arrived** or **Drive:*x* removed** where *x* is the drive letter.

File: devnotify.dll Device: 222.

15.6 Diamond Key Plugin

Provides an OSD keyboard which can be used for text entry from a remote with four arrow keys.

© 2008 Promixis, LLC



Configuration Page

The Configuration Tab is described in the next section.

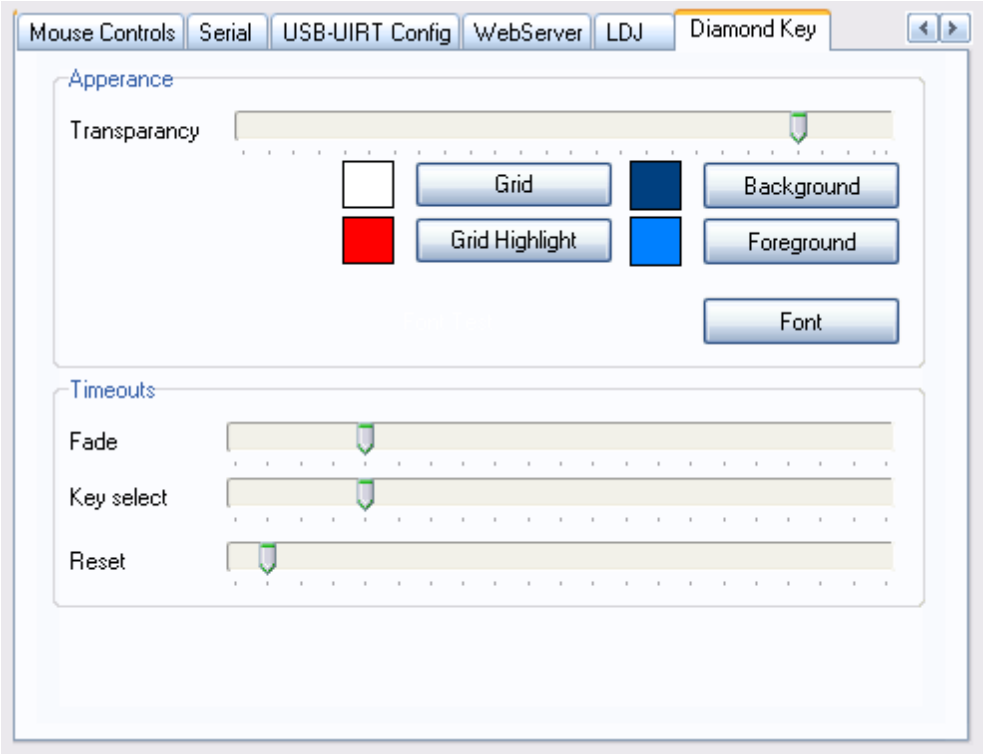
Actions Provided

Diamond Key Entry Action for linking remote buttons to the keyboard.

File: diamondkey.dll Device: 157.

© 2008 Promixis, LLC

15.6.1 Diamond Key Configuration



Appearance: Controls the look of the OSD. Transparency controls the degree to which windows under the OSD (or the background) shows through. If the slider is moved fully to the left, the OSD will be invisible.

Fade: Time with no activity before the OSD fades out.

Key select: Time with key selected before it is actioned.

Reset: Time during key selection with no activity before the selection is reset.

15.7 Generic Internet Plugin

Provides generic communications over Internet Protocol via Lua scripting.

Lua Extensions

Provides the gip Library.

File: gip2.dll Device: 251.

15.8 Generic Serial Plugin

Requires Girder Pro.

Provides generic communications over RS232 serial ports via Lua scripting.

Lua Serial Device Drivers may be written to generate events from specific serial devices and

© 2008 Promixis, LLC

provide functions to command them. These should be placed in the **%GIRDER%\plugins \serial** directory. A number of Device Drivers are shipped with Girder.

- IRMan. Raises events from IRMan, UIR or IRA IR remotes. Should be regarded as a sample since there is a specific Plugin for these devices.
- W800RF32. Raises events from the X10 RF receiver by WGA.
- CallerIDModem. Feeds information to the CallerID Library from most data modems that support this feature.
- PowerLinc Serial. Sends X10 Home Control commands using a PowerLinc II Serial X10 interface.
- NetCallerID. Feeds information to the CallerID Library from a NetCallerID device.
- Panasonic Plasma. Provides commands to control a Panasonic Plasma TH42PHD6UY.
- Command Cube. Generates events from a AOpen Command Cube remote device.

Configuration Page

The Configuration Page is described in the next section.

Events Generated

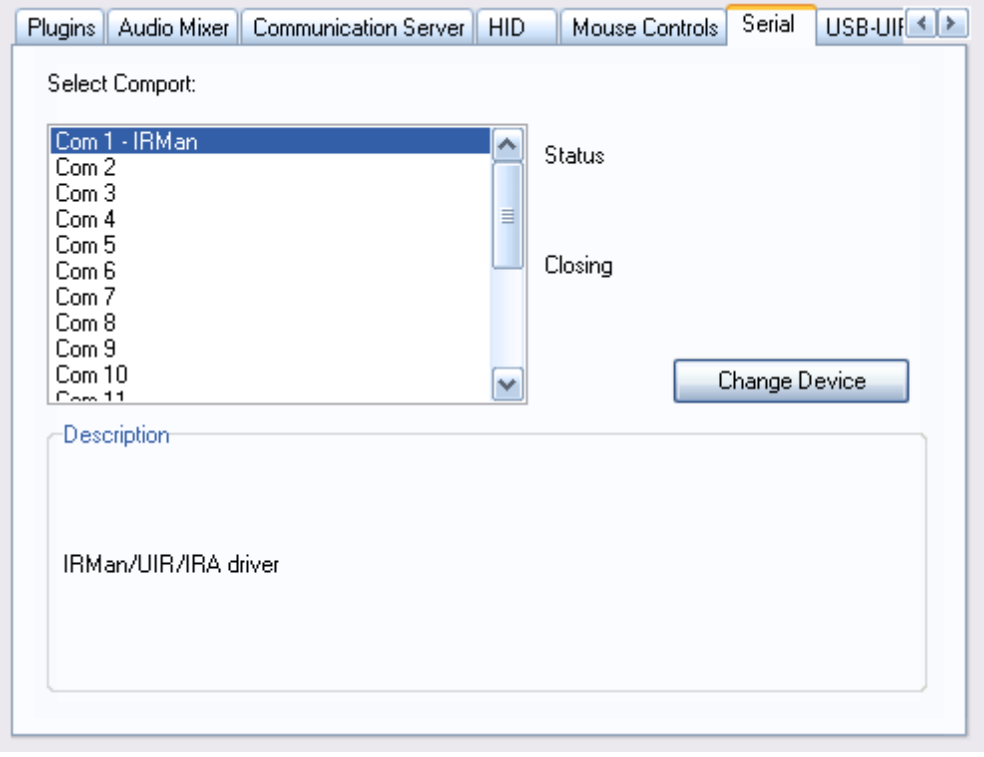
Depends on the Lua Device Driver.

Lua Extensions

Provides the serial Library.

File: serial.dll Device: 242.

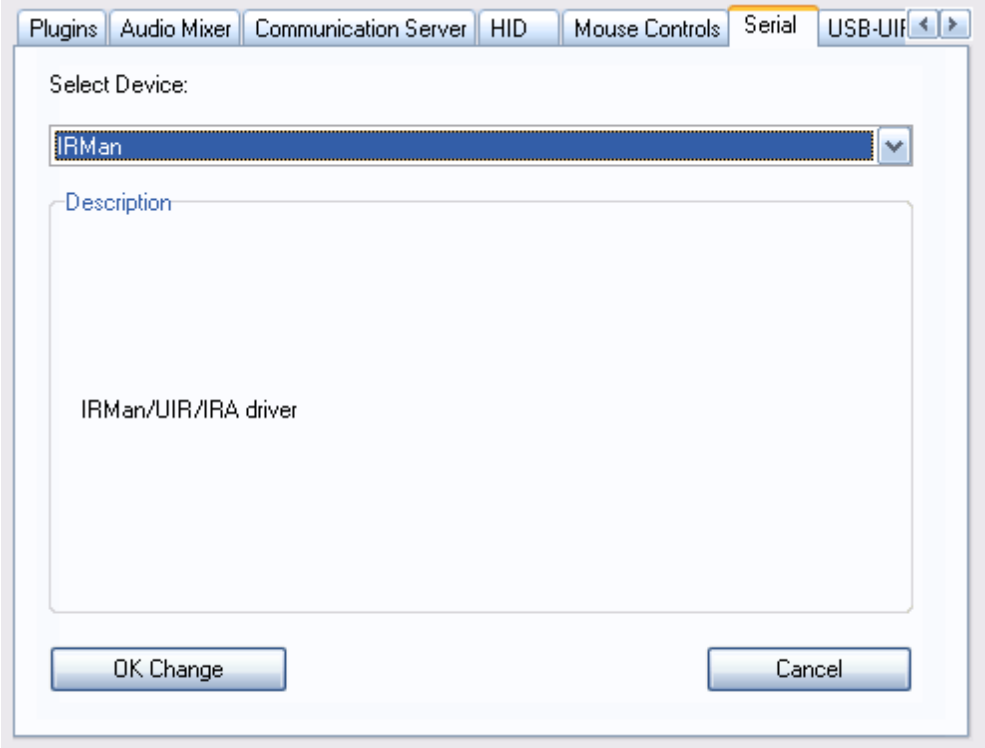
15.8.1 Serial Configuration



Select Comport: Select the serial port to be assigned.

© 2008 Promixis, LLC

Change Device: Press this button to select a sub-device driver for the selected port.



Select Device: Select the serial sub-device driver from the drop-down.

OK Change: Press to accept the sub-device driver and return to page 1.

Cancel: Leave this port unassigned.

15.8.2 Serial Device Drivers

The Serial Class Library is implemented in **%GIRDER%plugins\serial\init.lua** and specific Device Drivers may be placed in other files in this directory. The Plugin loads and executes **init.lua** followed by all other Lua files in this directory on startup. Low-level functions useful for writing serial device drivers are described in the serial Library section.

The class library provides two alternate base classes one of which should be used as the base for creating a new Device Driver.

- Simple. Suited to receive-only devices or protocols without command responses.
- Queued. Suited to devices with command/response protocols.

Device Driver Template

The basic device driver template looks like this (in, say, **%GIRDER%plugins\serial\MyDevice.lua**).

```
local device = serial.Classes.Simple:New({
    Name = "My Serial Device",
    Description = "Description of my serial device.",
})
serial.AddDevice(device)
```

This slightly odd syntax creates a Lua table by making a copy of a base table **Simple** and then adding new fields keyed **Name** and **Description** to it. **AddDevice** adds this table to another table **serial.devices** using the **Name** field as the key.

© 2008 Promixis, LLC

At this point, the new Device Driver will be available for configuration in Serial Configuration.

Communications Settings

At this point, when Girder starts, the specified Comport will be automatically opened with default settings and closed again just before Girder exits. To specify non-default communications settings for the Comport, add some standard fields to the Device Driver object.

BaudRate: Number, default=9600. The baud rate (speed) for the communication in bits per second.

FlowControl: String, default='H'. 'H' - Hardware (RTS/CTS); 'S' - Software (XON/XOFF); 'N' - None.

Parity: Number, default=0. 0 - None; 1 - Odd; 2 - Even; 3 - Mark; 4 - Space.

StopBits: Number, default=0. 0 - One; 1 - One and a Half; 2 - Two

DataBits: Number, default=8. Number of data bits per character. 6, 7 or 8.

IntraCharacterDelay: Number, default=0. Gap in milliseconds between characters.

BurstSize: Number, default=256. Maximum number of characters in a single transmission.

Example:
```
local device = serial.Classes.Simple:New({
    Name = "My Serial Device",
    Description = "Description of my serial device.",
    BaudRate = 19200,
    FlowControl = 'N',
    BurstSize = 1024,
    --Code in following examples should be inserted here.
})
serial.AddDevice(device)
```
Initialization

Immediately after the Comport has been opened, a stub function Initialize gets called. This may be replaced to add initialization functionality.
```
    Initialize = function (self)
        -- Initialization code goes here
        self.Status = "Port Opened"
       return serial.Classes.Queued.Initialize(self)
    end,
```
When replacing stub functions, it is good practice always to call the original version that is being replaced. This ensures that any base class functionality continues to work.

Receiving Messages

In order to receive messages from the device, some form of packet scheme is needed to determine the start and end of a message. Three alternate schemes are supported and one of these is suited to most devices on the market.

- TERMINATED. In this scheme, a fixed sequence of characters (often ASCII CR or CR with LF) is used to terminate a variable length message string.
- FIXEDLENGTH. In this scheme, all message strings have the same number of characters.
- MARKEDLENGTH1. In this scheme, there is a fixed marker character followed by a single character count of the message length, then that number of message characters possibly followed by a fixed number of termination characters.

© 2008 Promixis, LLC

The following Device Driver fields specify the receive message scheme.

CallbackType: Number, default=serial.CB_TERMINATED. serial.CB_TERMINATED, serial.CB_FIXEDLENGTH or serial.CB_MARKEDLENGTH1.

ReceiveTerminator: String, default='\n\r'. The string used to terminate the TERMINATED message.

ReceiveFixedLength: Number, default=8. The length in characters of a FIXEDLENGTH message.

ReceiveStartByte: Number, default=0. The character code of the character used as the fixed marker in MARKEDLENGTH1 messages.

ReceiveTerminatorLength: Number, default=0. The number of characters expected after the counted characters and before the marker for the next message in a MARKEDLENGTH1 message.

IncompleteResponseTimeout: Number, default=300. The maximum milliseconds to wait between characters within a message that has not been terminated yet. The key serial.INFINITE can also be used.

When a complete message has been received or a timeout or other error occurs, a stub function **ReceiveResponse** is called. This should be replaced to process incoming messages.

```
CallbackType = serial.CB_TERMINATED,
ReceiveTerminator = "\r",
ReceiveResponse = function(self, data, code)
    if math.band(code, serial.RXCHAR) > 0 and data then
        -- Process the received message in "data"
    end
   serial.Classes.Simple.ReceiveResponse(self,data,code)
end,
```

A typical process for received data is to parse the data and use it to form a Girder Event using **gir.TriggerEventEx**.

The above approach is usually adequate because most error conditions are handled in the base class **ReceiveResponse**. However, other conditions can be handled. The **code** parameter is a bitmap which contains a logical **or** combination of any of the following symbols.

serial.BREAK: A RS232 Break condition was detected on the line.

serial.CTS: The RS232 CTS input changed state.

serial.DSR: The RS232 DSR input changed state.

serial.ERR: A low-level error such as framing or parity was detected by the receiver hardware.

serial.EVENT1: A specified marker character was received.

serial.EVENT2: A specified marker character was received.

serial.PERR: A parity error was detected in the incoming data.

serial.RING: The RS232 Ring Indicator input changed state.

serial.RLSD: The RS232 Receive Line Signal Detect input changed state.

serial.RX80FULL: The receive buffer passed above a set threshold (nominally 80%).

serial.RXCHAR: A complete message has been received and is in the data parameter.

© 2008 Promixis, LLC

serial.RXFLAG: A specified marker character was received.

serial.TXEMPTY: The transmission buffer is empty (transmission is complete).

serial.INCOMPLETERESPONSETIMEOUT: The gap between characters within a message was longer than the set timeout.

serial.NORESPONSETIMEOUT: The time between message transmission and reception was longer than the set timeout.

Sending Messages

Best practice is to create a specific message sending function for each distinct message type that can be sent to the device. The **Simple** base class provides two primitive send functions that can be called to actually send the assembled message string.
`self:Write([string])`
`self:SendCommand([string])`
Write sends the string exactly as specified. **SendCommand** also honors the following settings keys.

SendStartByte: String, default="". This string is prefixed to messages (packet start characters).

SendTerminator: String, default="". This string is suffixed to messages (packet terminator).

NoResponseTimeout: Number, default=500. If there is no reception for longer than this number of milliseconds after the command is sent then **ReceiveResponse** will be called with NORESPONSETIMEOUT code. Can also be set to serial.INFINITE.

ResponsePending: Boolean. This flag is set by **SendCommand** and reset when a response has been received.

Example:
```
    SendStartByte = "AT"
    SendTerminator = "\r\n"
    CallbackType = serial.CB_TERMINATED,
    ReceiveTerminator = "\r\n",
    GlobalName = "MyDevice",
    DoHayes = function(self, command)
        self:SendCommand(command)
        while self.ResponsePending do
            win.sleep(100)
        end
        return self.Response
    end,
    ReceiveResponse = function(self, data, code)
        if math.band(code, serial.RXCHAR) > 0 and data then
            self.Response = data
        else
            self.Response = nil
        end
       serial.Classes.Simple.ReceiveResponse(self,data,code)
    end,
```
The GlobalName parameter causes the functions to be exported as a global table with the supplied name, so the function can now be called from other scripts.
`local response = MyDevice:DoHayes("?")`
DoHayes sends a Hayes AT command (used by modems) and blocks the caller for up to the default **NoResponseTimeout** of half a second. If a response has been received in this time, it is returned, else (presumably a timeout) **DoHayes** returns nil.

© 2008 Promixis, LLC

NB: If more than one instance of the same device is added to different ports, the first device added will be represented by a table with the specified global name. Subsequent instances will have the specified global name with the com port number suffixed. If in doubt, check the Variable Display.

Queued Sending

If the device driver object is based on **serial.Classes.Queued** rather than **serial.Classes.Simple**, then a more advanced version of **SendCommand** is available. This stores messages and response functions on a queue. Behind the scenes, the command is sent and when a response arrives the response function is called. Then the next command is sent and so on. It is not necessary to provide a **ReceiveResponse** replacement.

```
[Callback] = function([timestamp], [data], [code])
[timestamp], [pos] = self:SendCommand([command], [Callback], [topofq], [force])
```

timestamp: Number. Arbitrary number linking command with callback.

data: String. The received string.

code: Number. Bitmap as for **ReceiveResponse** above.

pos: Number. The position in the queue (the number of commands ahead of this one).

command: String. The command to be sent.

topofq: Boolean, default=false. If true, this command will be inserted at the head of the queue.

force: Boolean, default=false. If true, the command will be sent regardless of status, even if a previous response is still pending.

SendCommand also honors the following settings keys.

SendStartByte: String, default="". This string is prefixed to messages (packet start characters).

SendTerminator: String, default="". This string is suffixed to messages (packet terminator).

NoResponseTimeout: Number, default=500. If there is no reception for longer than this number of milliseconds after the command is sent then the callback will be called with NORESPONSETIMEOUT code. Can also be set to serial.INFINITE.

Example:

```
local device = serial.Classes.Queued:New({
    Name = "My Serial Device",
    Description = "Description of my serial device.",
    BaudRate = 19200,
    FlowControl = 'N',
    SendStartByte = "AT",
    SendTerminator = "\r\n",
    CallbackType = serial.CB_TERMINATED,
    ReceiveTerminator = "\r\n",
    GlobalName = "MyDevice",
    OnResponse = function(ts, data, code)
        if math.band(code, serial.RXCHAR) > 0 and data then
            gir.TriggerEventEx(data, 18, 0)
        else
           gir.LogMessage("MySerialDevice", "DoHayes Failed", 1)
        end
    end,
    DoHayes = function(self, command)
       self:SendCommand(command, self.OnResponse)
```

© 2008 Promixis, LLC

```
    end,
})
serial.AddDevice(device)
```

Management

These functions provide programmatic access to the functions normally carried out by the configuration system, but which are useful for automated configuration etc.

DeviceAssignToPort associates a comport with a Device Driver class. **DeviceLoad** creates the Device driver registered for a given comport. **Delete** deletes a driver and removes its comport assignment.

```
res = serial.DeviceAssignToPort([comport], [Device])
res = serial.DeviceLoad([comport])
res = serial.Delete([comport])
```

comport: Number. The port number.

Device: Table. Device Driver which must be created by the New function of one of the Classes.

Initialize opens the port and runs the Device initialization routine. **Close** closes the port.

```
res = serial.Initialize([comport])
res = serial.Close([comport])
```

comport: Number. The port number.

15.9 Generic X10 Remote Plugin

Generates events from X10 remote controllers like Snapstream FireFly and Niveus.

The Firefly device drivers must be installed. firefly.dll must be placed in the **%GIRDER% \plugins** directory.

Events Generated

Generates a unique eventstring for each key which may be learned. Generates Down, Repeat and Up versions.

File: X10-Remotes.dll Device: 161.

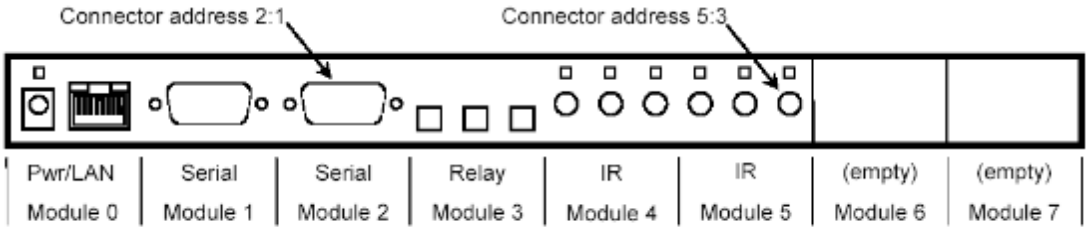
15.10 Global Cache Plugin

Provides Actions and a Lua library for interfacing with the Global Cache IR transmitter and interface device. Note that the Actions require the Tree Script Plugin in addition to this plugin.

Device Addressing

The default IP address for the GC device is 192.168.1.70. If you change this in CG setup you will need to specify the new address in the Action Editor and in the Lua function. Module addresses are mapped as follows.

© 2008 Promixis, LLC



GC-100-12

Actions Provided

Provides the Global Cache Actions group.

Lua Extensions

Provides the globalcache Library

File: globalcache.dll Device: 250.

15.11 IRTrans Plugin

Allows Girder to send IR commands using an IRTrans network IR device and raises Girder Events from commands received from remotes.

See http://www.irtrans.com/ for details of the devices.

Configuration Page

The Configuration Page is described in the next section.

Events Generated

Generates events for IR signals received from remotes. These can be mapped using the Event Mapping Editor.

Actions Provided

Provides the Send IR Code via irserver Action.

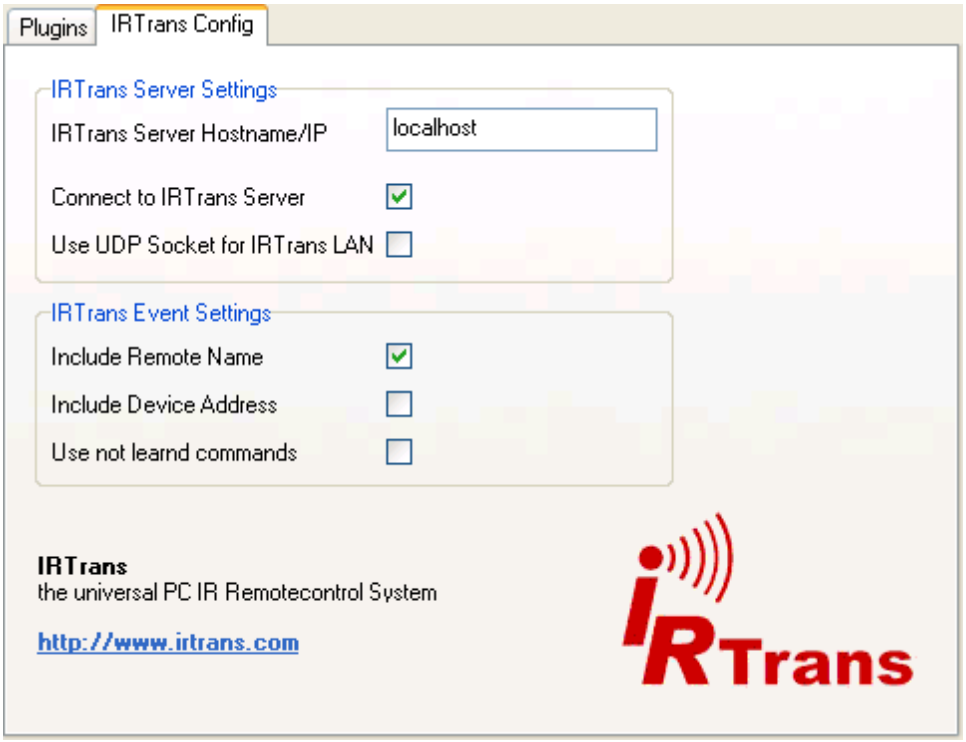
Lua Extensions

Adds the IRTrans.SendIR function.

File: irtrans.dll Device: 83.

© 2008 Promixis, LLC

15.11.1 IRTrans Configuration



15.12 Keyboard Plugin

Raises events when keys are pressed on the computer's keyboard. Also provides a Lua extension for enabling and disabling keyboard keys.

Events Generated

When a key is pressed, a **Down** event is generated with the **eventstring** being a unique hash code representing the key and any modifier keys (Ctrl, Alt, Shift, Left-Windows, Right-Windows) currently pressed. Then an **Up** event is generated with the same **eventstring**. If a key is held down, Repeat events will be generated at the keyboard repeat rate between the **Down** and **Up** events.

For modifier key strokes (e.g. **Alt-F1**) there are three distinct time periods. First, the modifier key is pressed on its own, next the modifier key and the other key are pressed simultaneously, finally the other key has been released, but the modifier key is still pressed. However, because the modified key has a unique **eventstring** you only need to respond to a single event.

Girder also has a keyboard mapping function which insulates Program Definitions from keyboard differences. If an **eventstring** is recognized by the Keyboard Mapping, each Keyboard event will be immediately followed by a Keyboard Mapping event in which the **eventstring** is mnemonic (e.g. **KEY SPACE**).

Lua Extensions

Adds the keyboard Library.

File: keyboard.dll Device: 202.

© 2008 Promixis, LLC

15.13 Lua Misc Function Library Plugin

Adds some miscellaneous Lua functions for accessing CDDB music information databases and Now Playing information on some music player applications.

Lua Extensions

Adds the mutil Library.

File: MiscUtils.dll Device: 236.

15.14 Monitor APM Extensions Plugin

Adds an action to control monitor power saving and some Lua functions to control monitor related functions in a single or multiple monitor environment.

Actions Provided

Adds the Monitor Power Management Action.

Lua Extensions

Adds the monitor Library.

File: multimon.dll Device: 138.

15.15 Mouse Control Plugin

Enables Standard Controllers and NetRemote to control the mouse pointer remotely. Adds Lua extensions for controlling the mouse pointer.

Configuration Page

The Configuration Tab is described in the next section.

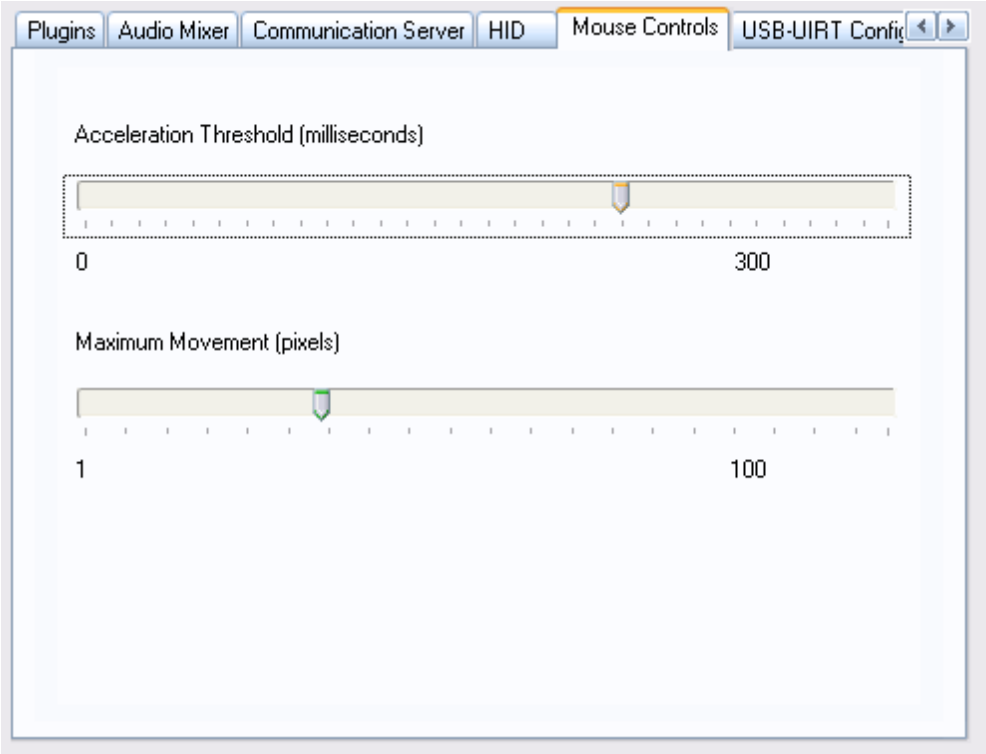
Lua Extensions

Adds the mouse Library.

File: MouseControl.dll Device: 228.

© 2008 Promixis, LLC

15.15.1 Mouse Control Configuration



Acceleration Threshold: Time in milliseconds after which step size increases.

Maximum Movement: Step size in pixels after threshold time.

15.16 PowerLinc USB Plugin

Allows Girder to send and receive X10 Home Automation signals using the USB version of the PowerLinc interface.

Configuration Page

The Configuration Page is described in the next section.

Events Generated

Unique events are generated for each house code/ unit code command received on the powerline interface.

Actions Provided

Provides the SendX10 Action.

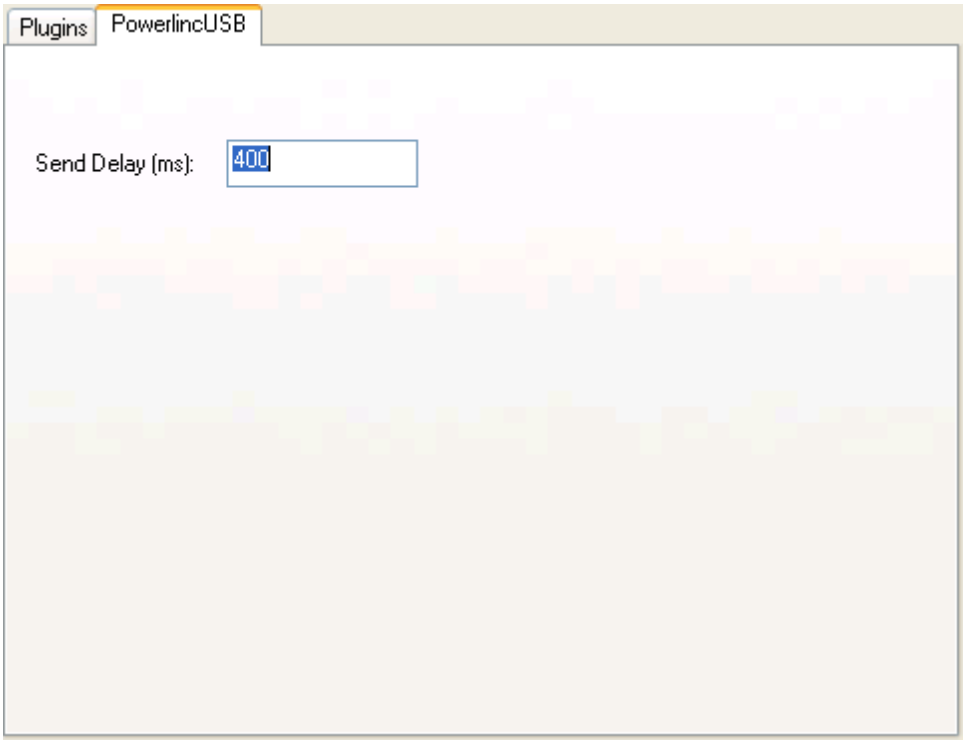
Lua Extensions

Provides the PowerLincU library.

File: powerlinc.dll Device: 153.

© 2008 Promixis, LLC

15.16.1 PowerLinc USB Configuration



Send Delay (ms): Line quiet period before an X10 command is sent.

15.17 Scheduler Plugin

Requires Girder Pro.

Adds actions and Lua functions for scheduling activities at particular times.

Actions Provided

Adds the Scheduler Actions group.

Lua Extensions

Adds the scheduler Library.

File: Scheduler.dll Device: 235.

15.18 SendMessage Plugin

Adds the legacy SendMessage action which may be required by Girder 3 GML files. New designs should use the built-in Command Capture Action instead.

Actions Provided

Adds the SendMessage Action.

© 2008 Promixis, LLC

File: SendMessage.dll Device: 112.

15.19 SimpleTimer Plugin

Adds the SimpleTimer action which generates an event after a time delay or repeatedly at periodic intervals.

Actions Provided

Adds the SimpleTimer Action.

File: simpletimer.dll Device: 122.

15.20 SlinkE Plugin

Interfaces the Slink-e Infrared Equipment Controller. Currently, only the Consumer IR transmission facilities are supported, though support is likely to be extended in future Girder versions.

Actions Provided

Adds the SlinkE Actions.

File: Slinke.dll Device: 254.

15.21 Streamzap PC Remote Plugin

Generates Girder Events from a Streamzap PC remote control device.

http://www.streamzap.com/.

Configuration

The driver can be set to generate "named" events from the Streamzap remote or to generate "raw" events from the Streamzap remote or a third party remote through the Streamzap receiver. Highlight the Plugin on the Plugin tab list and press the "Settings" button to access this dialog.

Events Generated

Generates a unique Girder event for each key on the remote device.

File: streamzap.dll Device: 29.

15.22 TaskCreate Plugin

Generates events when applications start and stop.

Events Generated

Generates events in which the **eventstring** is the name of the executable file followed by **[CREATE]** or **[CLOSE]**.

© 2008 Promixis, LLC

File: taskcreate.dll Device: 212.

15.23 TaskSwitch Plugin

Generates an event whenever the foreground application changes.

Events Generated

The **eventstring** is simply the name of the executable file of the new foreground application.

File: taskswitch.dll Device: 211.

15.24 Tree Script Plugin

Enables additional Actions, Conditionals and Configuration pages to be added to Girder using Lua Scripting and the Dynamic User Interface Designer.

The Programming the Girder UI section provides detailed information on Lua programming for this plugin.

Actions Provided

A large number of standard Actions and Conditionals in Girder are provided using Lua scripting via this plugin.

Lua Extensions

Adds the **treescript** Table. Runs any scripts found in the **%GIRDER%plugins\treescript** directory on Plugin start and on script reset. The **treescript** table stores callback functions which are called automatically by the Plugin in response to defined User Interface events and Action and Conditional triggering. The startup scripts install these functions and return the name of a Dynamic User Interface file which the Plugin inserts into the Girder DUI trees.

File: treescript.dll Device: 243.

15.25 UIR/IRman/IRA/CTInfra/Hollywood+ Plugin

Generates events from IRA, UIR, Evation IRMan, Creative Infra RS232 or Real Magic Hollywood+ remote control devices.

Configuration Page

The Configuration Tab is described in the next section.

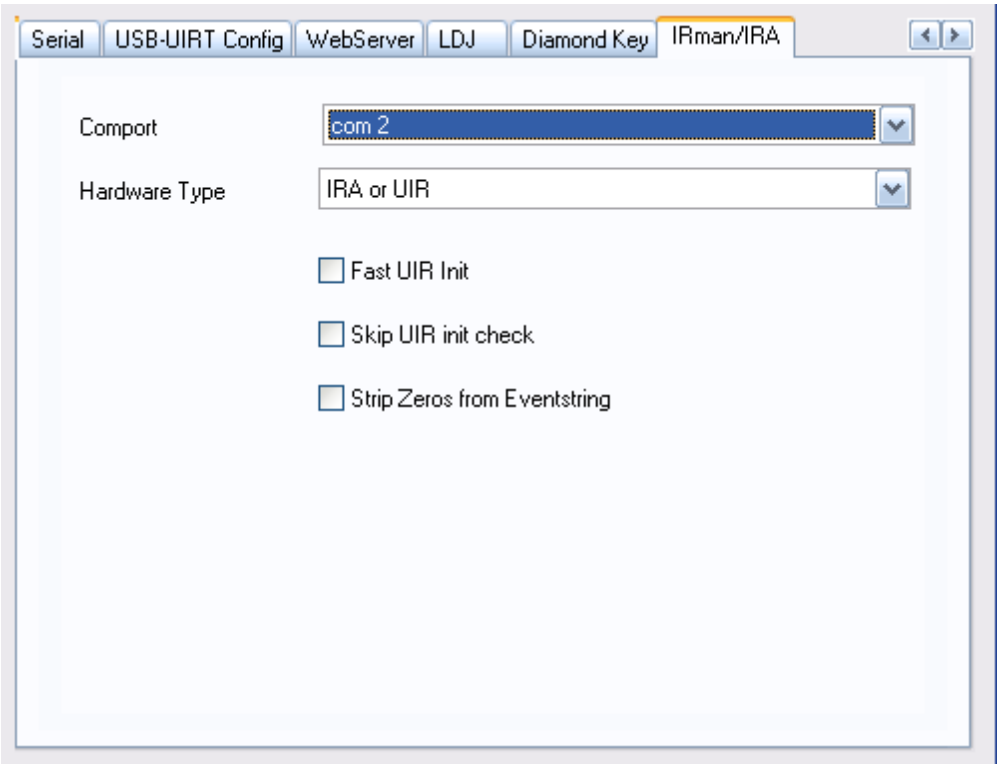
Events Generated

Generates a unique **eventstring** for each button on the controller in **Down**, **Repeat** and **Up** variations.

File: uir.dll Device: 201.

© 2008 Promixis, LLC

15.25.1 UIR Configuration



Comport: Enter the comm port that the device is connected to.

Hardware Type: Select the type of device.

Fast UIR Init: Initialize more rapidly.

Skip UIR init check: Carry on even if device does not respond.

Strip zeros from Eventstring: Remove leading zeros to shorten eventstring.

15.26 USB-UIRT driver Plugin

Transmits and receives CIR codes using a USB-UIRT device.

http://www.usbuirt.com/

Configuration page

The Configuration Tab is described in the next section.

Events Generated

Generates a unique event from each remote button that the receiver is trained to recognize.

Actions Provided

Provides the USB-UIRT Action. This is used for training the device to recognize CIR codes as well as to transmit them.

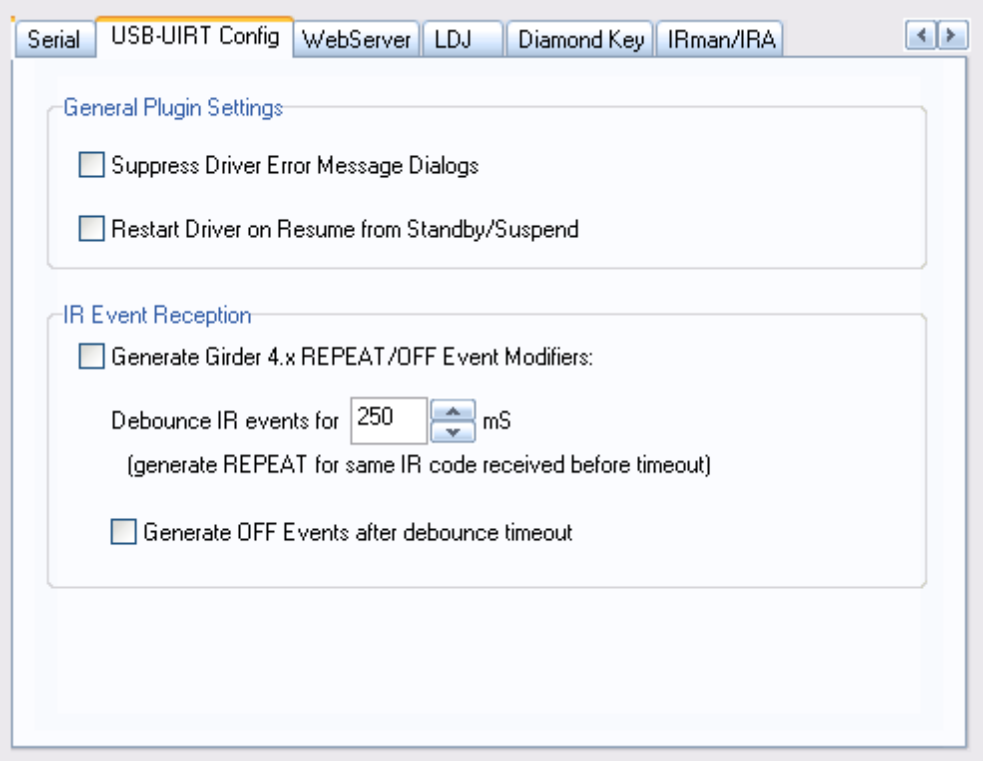
© 2008 Promixis, LLC

Lua Extensions

Adds the usbuirt Library. This provides a function for transmitting CIR codes from Lua.

File: USBUIRT.dll Device: 75.

15.26.1 USB-UIRT Configuration



Suppress Driver Error Message Dialogs: Tick to stop error dialogs appearing.

Restart Driver on Resume from Standby/Suspend: On some machines the receiver may not work after resume. If this is the case, tick this box to automatically reset it.

Generate Girder 4 Event Modifiers: Tick to generate Down, Repeat and Up modifiers for the events.

Debounce IR events: Maximum time between code receptions before an Up event is generated and further codes construed as a new key press.

Generate OFF Events after debounce timeout: Tick to generate an Up event when the debounce period has expired.

15.27 Web Server Plugin

Requires Girder Pro.

The web server plugin allows you to access Girder from a remote machine using a configurable web page interface. Ordinary or secure (http or https) access is available. Once the web server is configured, you can access it from a web browser using the following URLs.

© 2008 Promixis, LLC

From the machine with Girder: http://localhost/.

From another machine on your local network: http://*computername*/.

From the internet (assuming Girder is connected): http://*ipaddress*/.

For secure access, replace **http** with **https** in the above addresses. If you change the port number, add the port after the name or IP address separated with a colon.

A sample website is provided in **%GIRDER%httpd\index.html**. This may be used as is or you can replace it with your own website in another directory. See Web Programming Reference for details of how to program a Girder website.

Configuration Page

The Configuration Page is described in the next section.

Events Generated

An event is generated for each incoming request to the Web Server. The eventstring is the file requested and the payloads are extracted from the URL payload if any. For example, the relative URL "index.html?load1=1&load2='London'" generates an event with eventstring = "/index.html"; pld1 = "load1=1"; pld2 = "load2='London'".

You can also generate a custom event from a form on the web page using the GET method. There may be up to four controls and the event payloads will contain *name*=*value* for each.

Example:

```
<form action="index.html" method="get">
<input type="text" name="field1" value="testing">
<input type="text" name="field2" value="18">
<input type="submit" name="Button" value="Send Event">
</form>
```

When the button is pressed, an event is generated, eventstring = "/index.html"; pld1 = "field1=testing"; pld2 = "field2=18"; pld3 = "Button=Send Event" (assuming the form values have not been changed from the defaults).

Lua Scripting Extensions

See Web Programming Reference for details of how to run Lua scripts from web pages.

SSL Certificate Issues

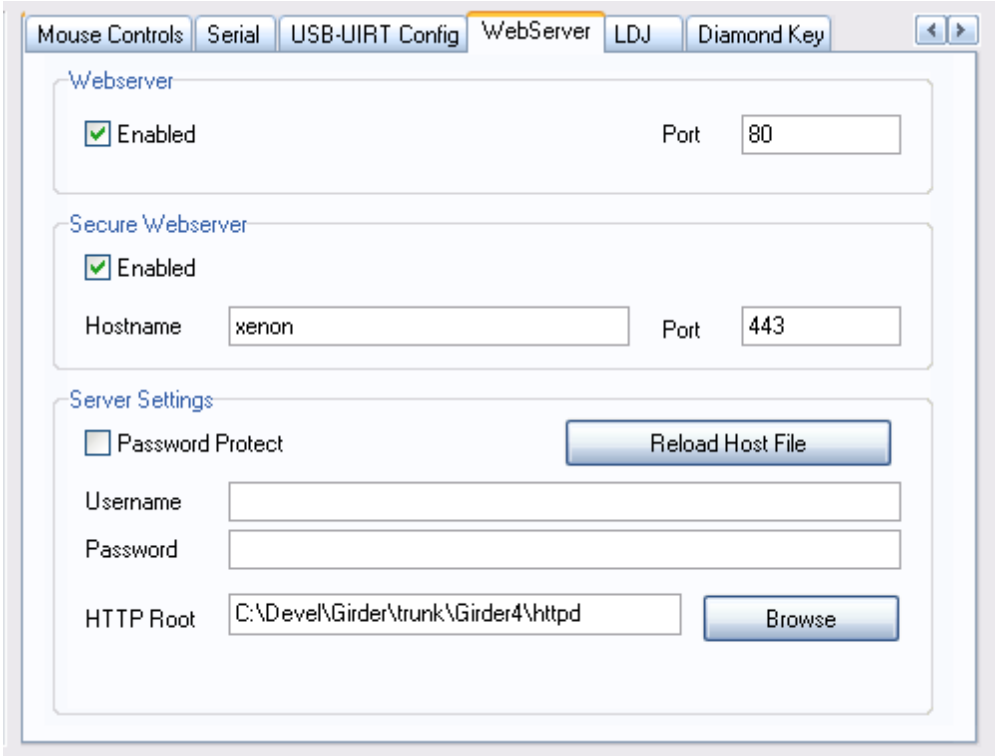
If you are having issues with SSL certificates please read this

---

File: httpd.dll Device: 36.

© 2008 Promixis, LLC

15.27.1 Web Server Configuration



Enabled: Check to enable un-encrypted (Webserver - http) and/or encrypted (Secure Webserver - https) access.

Port: The default port numbers are 80 for http and 443 for https. These can be changed, for example if there is another web server operating on this machine or to provide additional security in conjunction with a firewall. If the port is changed, clients will have to specify it in the URL, for example, **http://localhost:2001/** if the port is changed to 2001.

Hostname: The standard web server will respond to any URL that reaches this machine (for example "localhost", the machine name or a domain registered with DNS to translate to this machine's IP address). For secure access, the hostname must be specified. This will be included in the site certificate and if it does not match the URL requested by the client, the client will refuse to render the page or at least will warn the user.

Password Protect: If this is ticked both secure and standard web servers will cause the client to prompt for a password and user name before returning a page.

Username: The user name for authentication.

Password: The password for authentication.

Reload Host File: For advanced use, different host names may be associated with different directories of web resources on this machine. These associations are made in the file **%GIRDER%plugins\webserver\hosts.txt**. If you change this file, you can reload the changes into the webserver using this button.

HTTP Root: The directory where the publicly accessible web page files are kept (unless overridden for a specific domain by hosts.txt). By default it is **%GIRDER%httpd** which is a subdirectory under the Girder installation directory. This contains a sample website

© 2008 Promixis, LLC

demonstrating the capabilities of the Girder Web Server. You are strongly advised to create your own website in a different directory since updating or reinstalling Girder will overwrite this directory.

You will need to restart Girder or the Web Server Plugin for changes to these settings to take effect.

Adding MIME Types

Advanced users can add to the list of file types supported by the Web Server by editing the file **%GIRDER%plugins\webserver\mime.txt**. Each line in the file associates a file extension with a MIME type, for example ".html=text/html".

Another file, **%GIRDER%plugins\webserver\gzip-mime.txt** specifies file types which will be compressed for download to the client. In this file, "*" may be used to wildcard parts of the MIME type. The default file has a single line "text/*=true" which causes all MIME types beginning "text/" to be compressed by default.

Certificates for Secure Webserver

Secure Transmission depends on a cryptographic security certificate that verifies the identity of your website. Specifically, the user's browser checks that the site certificate matches the domain specified in the URL used to access it. Girder automatically generates a certificate for the **Hostname** supplied in the settings above. The certificate is contained in two files Cert-*name*.pem and Key-*name*.pem in the **%GIRDER%plugins\webserver** directory.

Certificates generated by Girder will not be certified by a recognized certification authority.

This issue will cause most web browsers to raise security warnings. Unless you are prepared to pay for a certificate from an authority, you will have to accept this. Most browsers will prompt you to accept, install or trust such a certificate either for the current session or permanently, but this will need to be done on every computer accessing a given Girder machine. To get an equivalent level of trust to authority certification, you should take a note of the "thumbprint" of the site certificate using a local browser. Then when connecting using a remote browser, ensure that the certificate thumbprint is the same. This proves you are looking at your certificate and not another generated by someone wishing to hijack your site.

Note that security depends on the security of the certificate files. If these are stolen, your Girder website can be spoofed.

15.28 Window Conditionals Plugin

Adds the Window Conditional which enables Girder nodes to be made conditional on the presence of a particular window or on whether a window is the foreground window.

Conditionals Provided

Adds the Window Conditional.

File: WinCon.dll Device: 193.

15.29 X10-CM1X Plugin

Provides actions to controls X-10 home automation devices via CM11, CM12 or CM17A interface devices. Also provides a Lua Library for the same purpose.

Configuration Page

© 2008 Promixis, LLC

The Configuration Tab is described in the next section.

Events Generated

For received X10 messages, generates events of the form *house_device_command_dim%_extval*, e.g. A_3_On; A_2_Dim_50. This string is also saved in Lua variable X10_Command so an Event Node can be used to recognize all events and process them using a Lua script.

Actions Provided

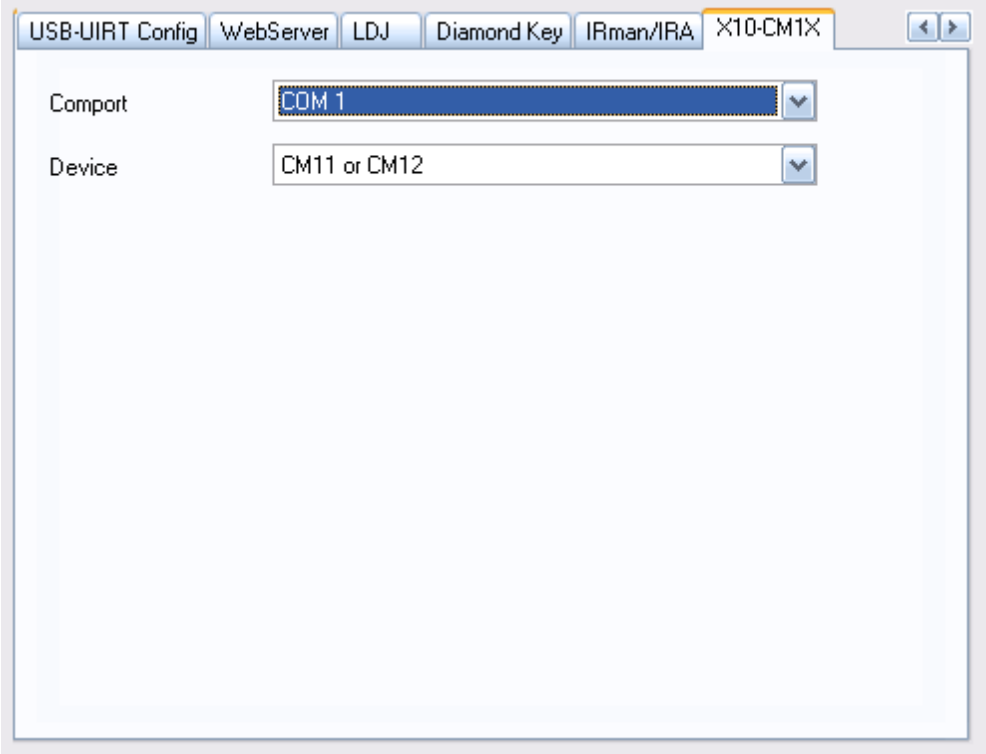
Adds the X10 Controls Actions group.

Lua Extensions

Adds the cm11 Library.

File: cm111217.dll Device: 69.

15.29.1 X10-CM1X Configuration



Comport: Select the comm port to which the device is connected.

Device: Select the device type.

15.30 xAP Automation Plugin

Provides a Lua library for using the xAP Automation open standard.

http://www.xapautomation.org/

© 2008 Promixis, LLC

You will need an xAP Hub application running on the same computer.

http://wiki.xapautomation.org/tiki-index.php?page=xAP%20Hub

Lua Extensions

Adds the xap Library.

File: xap.dll Device: 261.

15.31 Zip Extensions Plugin

Requires Girder Pro.

Adds Lua functions for working with Zip compressed archive files. This uses the Info-ZIP library, Copyright (c) 1990-2003 Info-ZIP.

Lua Extensions

Adds the zip Library.

File: ZipExt.dll Device: 231.

© 2008 Promixis, LLC

Part XVI

16 Upgrading from Girder 4

Upgrading from Girder 4 should be straightforward, just follow these basic rules.

- First and foremost, backup your GML's and any other customizations you made
- Install in a different folder from Girder 4

Some things that you might have to pay close attention to:

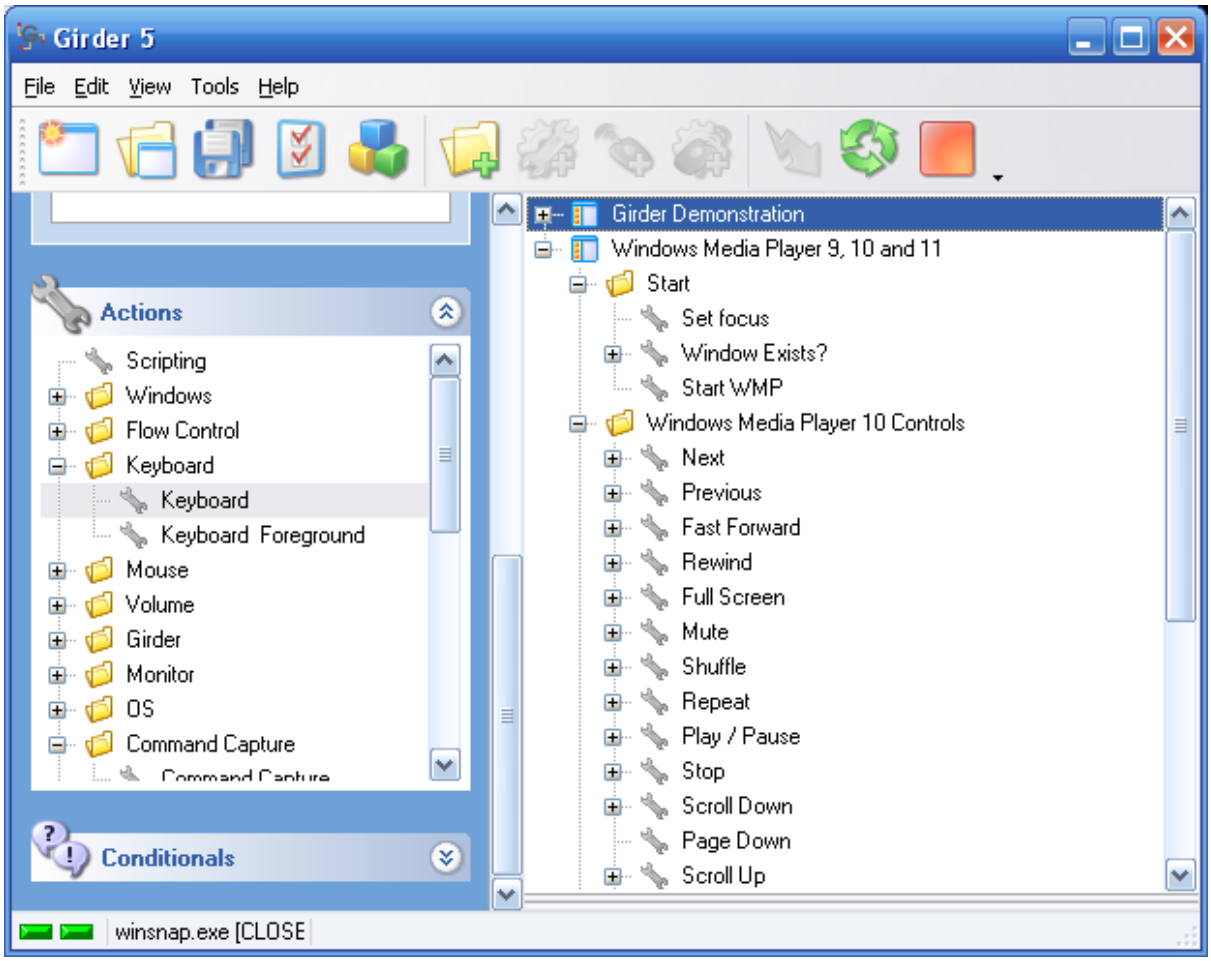
- The serial interface has changed and you might have to upgrade your serial definitions.

© 2008 Promixis, LLC

Part

17 Upgrading from Girder 3

Tip: Users upgrading from previous versions of Girder are recommended to switch immediately to **Expert** mode (using the **View** menu) as this is closer to the Girder 3 interface.



Upgraders will notice that the interface to Girder has radically changed. However, in Expert mode, the familiar Girder tree is there, but moved from the left to the right. The nodes in the tree are similar in function, but some of the terminology has changed.

- Girder can now open multiple GML files simultaneously, so each file appears as a top-level node in the tree.
- There is no distinction between a **Toplevel Group** and an ordinary **Group**. You can add a Group to a File node or to another Group node by right-clicking the parent node or by using the toolbar button.
- **MultiGroups** are now called **Macros**, but function in the same way.
- **Eventstrings** have been renamed simply **Events**. Events have been enhanced with new **Down**, **Repeat** and **Up** modifiers to enable finer control from keyboards and remote controllers.
- **Event Mapping** has been introduced to link controllers to program definitions in a less device dependent way. This allows GML files to be more reusable.
- **Commands** have been renamed **Actions** and are selected from a secondary tree on the task box to the left of the interface. Either highlight the parent node in the main tree and double-click on the required Action or alternatively drag the action onto the parent node in the main

© 2008 Promixis, LLC

tree.
- There is a new node-type **Conditionals** which is selected from an task box in the same way as Actions. A conditional enables or disables its parent node according to the test described in its settings.
- There is a new node-type **Macro Events**. This acts as a container for individual Events (similar to the way a Macro acts as a container for Actions). The Events in the Macro Event must all occur in sequence for the associated actions to be triggered.
- You can now disable any node individually and add documentation to any node. These functions are in the **General** task box top left of the interface.

GML File Compatibility

- Some Girder 3 Plugins will work in Girder 5, some will not work at all, and some will work with reduced functionality. Generally, event generation Plugins will work. Configuration UI pages will generally work, but not in the same way as Girder 5 Plugins – instead you should press the **Settings** button on the **Plugins** tab of the Settings dialog. Commands (Actions) from Girder 3 Plugins will run but cannot be edited. Look for updated Girder 5 Plugins on www.promixis.com.
- For maximum safety, make a backup copy of your GML file before opening it in Girder 5.

Changes in the Lua Language

- The whole tag-method scheme was replaced by metatables.
- Function calls written between parentheses result in exactly one value.
- A function call as the last expression in a list constructor (like **{a,b,f()}**) has all its return values inserted in the list.
- **or** now comes after **and** in precedence order.
- **in**, **false**, and **true** are reserved words.
- The old construction **for k,v in t**, where t is a table, is deprecated (although it is still supported). Use **for k,v in pairs(t)** instead.
- When a literal string of the form **[[…]]** starts with a newline, this newline is ignored.
- Upvalues in the form **%var** are obsolete; use external local variables instead.

Changes in the Lua Libraries

- Most library functions now are defined inside tables. There is a compatibility script (compat.lua) that redefines most of them as global names.
- In the math library, angles are expressed in radians. With the compatibility script (compat.lua), functions still work in degrees.
- The **call** function is deprecated. Use **f(unpack(tab))** instead of **call(f, tab)** for unprotected calls, or the new **pcall** function for protected calls.
- **dofile** does not handle errors, but simply propagates them.
- **dostring** is deprecated. Use **loadstring** instead.
- The **read** option ***w** is obsolete.
- The **format** option **%n$** is obsolete.

© 2008 Promixis, LLC

Part

18 What's New

NOTE: This manual will not necessarily be updated for all minor versions of Girder. If you are using a version later than shown below, check the online forums at www.promixis.com for the release notes.

Most notably Girder 5 introduces the Device Manager and Windows Vista support (including Vista64)

© 2008 Promixis, LLC

Index

- A -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- K -

- L -

- M -

© 2008 Promixis, LLC

- N -

- O -

- P -

- R -

- S -

- T -

- V -

- W -

- X -

© 2008 Promixis, LLC

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

© 2008 Promixis, LLC