Girder 6

© 2015 Promixis, LLC

Girder 6

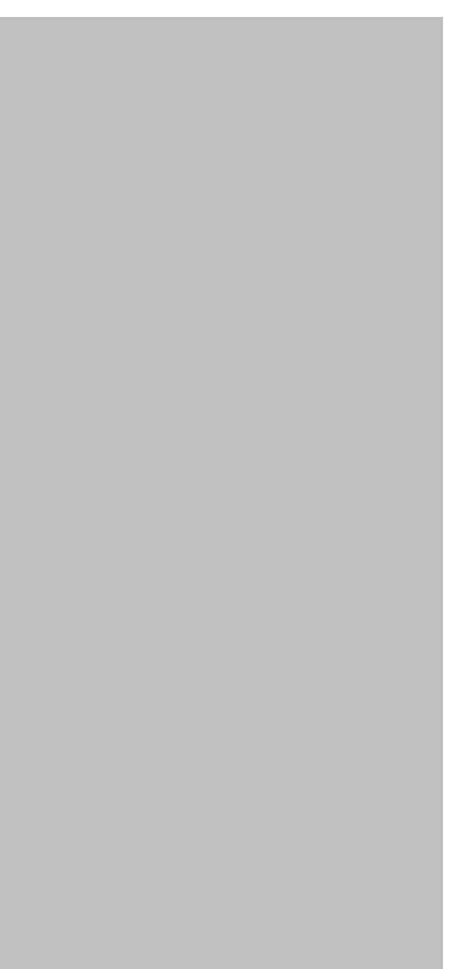
© 2015 Promixis, LLC

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: April 2015 in Jupiter, FL



Special thanks to:

FSM

Table of Contents

Foreword 0

Part I Introduction 10

Part II Getting Started 11

Part III Changes from Girder 5 12

Part IV Connection Dialog 13

Part V Concepts 16

1 Actions and Events ... 16
2 IR Codes ... 17
3 Device Manager ... 17
4 Sessions ... 19

Part VI Tutorials 20

1 Creating your first action ... 20
2 Hook Events to Actions ... 21
3 Pressing a button on Windows Calculator ... 23
4 IR Codes in an action ... 27
5 IR Codes on web page ... 29
Webpage IR codes using Actions ... 30
Webpage IR codes using the Device Manager ... 31
6 Control mouse from remote ... 35
7 Enabling Plugins ... 35
8 Importing IR codes from the web ... 36
9 Event Forwarding ... 39

Part VII Send events to Girder 41

Part VIII Actions 42

1 Command Capture ... 42
2 Device Manager ... 43
3 Flow Control ... 44
Checkbox is checked ... 44
Explicit Return ... 44
File Exists ... 45
Goto Action ... 45
Stop Processing ... 45
Window Exists ... 45

© 2015 Promixis, LLC

Window is Foreground 45

4 Girder Actions 45

Enable / Disable Node 45

Enable / Disable Plugin 46

OSD 46

Reset All States 50

Reset Node State 50

5 Keyboard Actions 50

6 Monitor Action 53

7 Mouse Actions 53

Targeted Mouse Actions 54

Un-Targeted Mouse Actions 54

8 Network Actions 55

Email Action 55

HTTP Request 55

9 PIR Actions 55

PIR-1 Actions 55

PIR-4 Actions 59

10 PRB-16 Actions 59

11 Scripting Action 60

12 Simple Timer 61

13 Speech Action 61

14 System Actions 62

Play Wav 62

Shutdown, Reboot, Poweroff and Logoff 62

Execute Action 62

Eject / Load Media 63

15 Twilio Actions 63

16 Twitter 63

17 USB-UIRT 63

18 Volume Action 63

19 Window Actions 65

Part IX Conditionals 65

1 Script 65

2 Device Manager 65

3 Time of Day 65

Part X Speech 65

Part XI Girder Command Line Options 66

Part XII NetRemote Command Line Options 67

Part XIII IR Devices 68

© 2015 Promixis, LLC

Part XIV Plugins 69

1 1-Wire ... 69
2 CM11 ... 69
3 Insteon ... 69
4 IR Devices ... 69
5 IRTrans ... 69
6 Nest ... 69
7 RFXCom ... 69
8 Scheduler ... 69
9 SimpleTransport ... 69

Part XV User GUI 69

1 dmModel Object ... 76
2 dmModelAdapter Object ... 77
3 kv ... 77
4 window ... 79
5 girder ... 79

Part XVI Scheduler 80

Part XVII Webserver 82

1 Password protection ... 82
2 Add SSL Protection ... 82
3 addModRewrite ... 86
4 clearModRewrite ... 86
5 escape ... 86
6 getBody ... 87
7 getCGI ... 88
8 getCookies ... 88
9 getFilename ... 89
10 getHeader ... 90
11 getHeaders ... 90
12 getHost ... 91
13 getMethod ... 92
14 getPort ... 92
15 getPort ... 93
16 getURL ... 93
17 print ... 94
18 registerWebSocketServer ... 94
19 setCookie ... 100

© 2015 Promixis, LLC

20 setHeaderEx ... 101
21 setStatus ... 101

Part XVIII Scripting 102

1 bit ... 102
2 class ... 108
3 cm11a ... 109
- getList ... 109
- registerForX10Events ... 110
- X10 Commands ... 111

4 date ... 112
- Date Object ... 112
- newDate ... 115
- newTime ... 116
- now ... 117
- nowUtc ... 118
- Time Object ... 118
- utcOffset ... 120

5 delay ... 121
6 deviceManager ... 122
- areEventsBlocked ... 122
- blockEvents ... 122
- component ... 123
- Component Object ... 124
- components ... 124
- control ... 125
- Control Object ... 126
- controls ... 127
- device ... 127
- Device Object ... 128
- devices ... 129
- Locations ... 129
- requestControlValueChange ... 130

7 gd ... 131
8 gir ... 131
- addEventHandler ... 131
- getversion ... 134
- girderClosing ... 135
- log ... 135
- parseString ... 136
- removeEventHandler ... 136
- settings ... 137
  - Webserver Settings ... 138
  - Proxy Settings ... 139
  - Email Settings ... 139
- shutdownNotifier ... 140
- triggerEvent ... 140

9 hid ... 141
- enumerate ... 141
- open ... 142

HidDevice ... 144
PIR-1 HID Complete Example ... 148
10 json ... 151
11 kv ... 152
12 lfs ... 154
13 lxp ... 155
14 math ... 156
crc ... 156
hexStringToBinary ... 158
binaryToHexString ... 158
formatbytes ... 159
hextodecimal ... 159
decimaltohex ... 160
decimaltobyte ... 160
binarytohex ... 161
15 mime ... 161
16 network ... 161
get ... 161
post ... 163
sendEmail ... 164
Wake On Lan ... 166
17 onewire ... 167
18 os ... 169
newFileSystemWatcher ... 169
FileSystemWatcher Object ... 169
19 pir ... 171
20 plugin ... 173
21 Promixis ... 174
Event ... 174
modifiers ... 175
EventNode ... 175
Modifiers ... 175
IEmailSettings ... 176
ConnectionType ... 176
IProxySettings ... 176
ProxyType ... 176
Transport ... 177
Connection ... 177
Type ... 177
IConnectionCallback ... 177
Status ... 177
IParser ... 178
Type ... 178
ITransactionCallback ... 179
Result ... 179
SerialConnection ... 179
Flow Control ... 179
StopBits ... 180
Parity ... 180
Control ... 180

© 2015 Promixis, LLC

DType 180
Device **181**
Status 181
Log **182**
22 publisher 182
23 raspi 183
export **184**
unexport **184**
direction **185**
write **185**
read **186**
24 scheduler 187
create **187**
getScheduler **187**
getSchedulers **188**
sunrise **189**
sunset **190**
Scheduler Object **190**
25 socket 200
imap **200**
26 speech 206
speak **206**
listVoices **206**
setVolume **207**
listOutputs **208**
setOutput **208**
27 sql 209
28 ssl 209
29 string 210
latin1ToUtf8 ..... **211**
local8BitToUtf8 ..... **211**
ltrim **212**
rtrim **212**
split **213**
trim **213**
30 table 214
copy **214**
isEmpty **214**
print **215**
tostring **215**
31 thread 216
Mutex **217**
Condition **217**
32 timer 217
new **218**
Timer Object **218**
33 transport 219
Connection Object **223**
Serial Connection Object 224
SSL Connection 225

© 2015 Promixis, LLC

TCP Connection 225
Transaction Object 225
UDP Connections 226
resolve 228
PIO-1 Example 228
34 twilio 241
callNumber 241
phoneNumbers 242
sendSMS 243
35 Twitter 244
tweet 244
directMessages 245
36 usbuirt 247
37 zwave 248
poll 248

Part XIX Regular Expressions 249

Part XX Z-Wave 251

1 How to add or remove a device 251
2 How to copy the controller 251
3 How to place controller in learn mode 251
4 How to include controller into existing network 251
5 Association Support 252

Index 253

© 2015 Promixis, LLC

1 Introduction

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.

Girder is used in a wide range of applications by a wide range of customers amongst others Girder is used by Intel, Sony, Microsoft, NBC, DOD, Netflix, Echostar, Bose and many more.

For more information visit: http://www.promixis.com

Control your PC using Remote Control Devices

Combined with a PIR-1 from Promixis you can control your PC from almost any remote in your house!

Control your PC from a Web browser

Girder's built-in web server allows you to access to the power of Girder remotely, even from across the globe with a web browser.

Automate your home

Girder has support for several home automation technologies right at your finger tips. No need for expensive hardware to run it. Girder can do it all. Combined with a PIR-1, PIR-4 or a PIO-1 you can control about any consumer electronics.

Simplify business & IT Task Automation

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

Healthcare

Use Girder in healthcare to provide accessibility where no other application or technology can provide for a price that is unbeatable.

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

© 2015 Promixis, LLC

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.

2 Getting Started

To start using Girder first we'll need to start the application. On the start menu you'll find "Girder Local" or "Girder Remote". To understand the difference we'll need to know a bit about the way Girder works.

Girder is made up out of two main parts. The back-end and the front-end. The back-end is where most of the work is done for Girder. This is where all the automation takes place, it's the brain of the two. The front-end is the part that interacts with you. It's the user-interface. These two parts are completely separate and can run on completely different machines. Of course they do not have to run on different machines in fact they can run inside the same process.

To run Girder as one process, meaning the back-end and the front-end run at the same time on the same machine select "Girder Local". This is probably a good place to start. Below are the various methods of running Girder:

Run in one process

This is the "Girder Local" option from the start menu. This means that the back-end and the front-end are both running in the same process. Stopping the front-end will stop the back-end as well. Note that you can set the front-end to run in the tray bar. This is the way Girder 5 ran.

Run in service

This runs the Girder back-end as a service on your machine. It starts up with your computer and runs until the computer is turned off. If not already done you'll have to register the service. To do this you'll need to start "Girder Remote" as administrator by right clicking the start menu and selecting "run as administrator". The click on the "Local Network" tab and find the "Girder Service Settings..." button. There press the "Install Service" button. You can also start it from that dialog.

Note that when running Girder as a service, Girder is using a different set of configuration files from the Girder instance that is running in process. The reason for this is that configuration settings are stored on a per user basis. The service based Girder is running as the service users (Local System).

Front-end connect to service

Connect to it using the "Girder Remote" option on the start menu. This will open the "Local Network" tab on the Girder connection dialog. If the service has been started you should see the computer name listed here. By default the password is "girder".

Front-end connect to remote Girder anywhere

If your remote Girder instance is not showing up in the "Local Network" tab use the

© 2015 Promixis, LLC

"Internet" tab to connect. Here you must enter the IP address, port number and password to connect.

Secure Connections

Girder supports SSL encryption of the communications. To use this simply select "Secure Connection". Special version of Girder can be requested that perform full SSL certificate verification.

What next?

Now that you have Girder up and running the next step is to learn about the basic actions and events concepts in Girder and then to walk through a tutorial.

3 Changes from Girder 5

Girder 6 is a rebuild from ground up. As such it's not compatible with the Girder 5 structure. In fact it's a major overhaul of the Girder architecture. This has allowed Girder to run as a service, be ported to many different Operating Systems, including Android, Mac, Linux and BlackBerry (iOS soon). Not all ports are publicly available but can be obtained upon special request.

GML differences

The GML tree looks very similar between Girder 5 and 6. In fact a lot of the actions are indeed the same. Some things have changed however. Any of the actions that deal with interacting with other Windows are now asynchronous. For example the Wait for Window action no longer blocks the Girder execution. You can safely use larger timeouts here. This however means that if your existing GML relies upon that wait to be synchronous things might not work as they did before.

Some actions are no longer available. If you encounter an action that is missing and you really need it, contact support.

Device Manager

The device manager code has been completely overhauled to be more like the PEAC device manager. Old code will not work.

Transport classes

The transport classes have been overhauled to be compatible with the PEAC classes. The code is similar in structure but you'll need to do a little work.

DUI

Girder 5 had the "DUI" (Dynamic User Interface) facilities to provide scripts with a way to

© 2015 Promixis, LLC

create user interfaces. While it worked well enough for simple things it quickly broke down with more complex things. Girder 6 brings a new script plugin. This plugin allows you to write plugins for Girder 6 without a compiler! Just lua code for the backend (this is the part that interacts with the hardware) and javascript code for the front-end.

Webserver

The webserver code has been redone to allow for websockets to be created. Compatibility functions exists in the namespace.

Plugins

Sadly, none of the old plugins are supported by Girder 6.

Volume Controls

The volume actions have been changed to deal with the new Audio Subsystem of Windows and be more flexible in general.

NetRemote

Girder has the NetRemote replacement built-in. This no longer loads CCF files however.

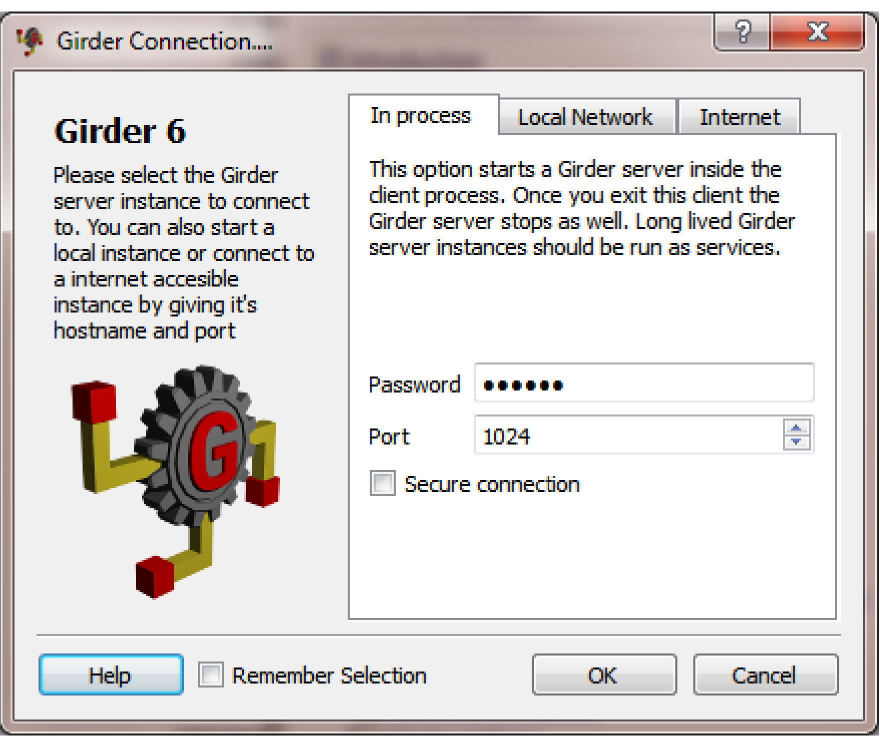
4 Connection Dialog

The connection dialog allows you to control how Girder runs and what instance it connects to.

In process

In process means that the Girder back-end runs simultaneously with the front-end in the same process on your machine. This is the way that Girder 5 worked.

© 2015 Promixis, LLC

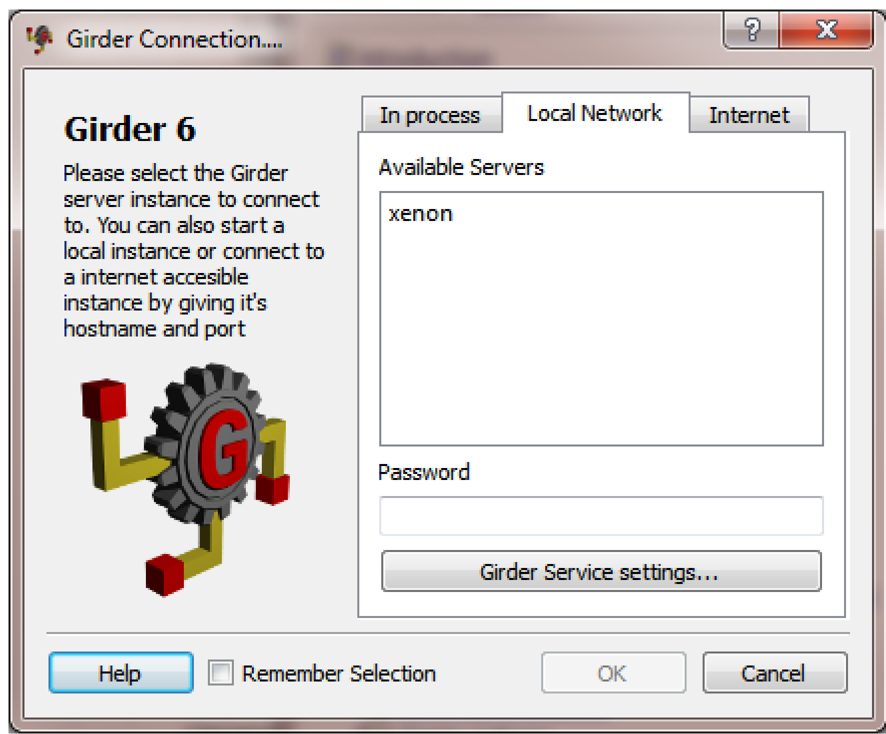


Connection Dialog

Even though the back-end runs in the same process you can still connect to the back-end from a different computer to the back-end. To do that you'll need to use the same port and password as you entered on this dialog. If you wish to encrypt the communication traffic check the "Secure Connection" dialog.

Local Network

Once you started the Girder back-end on your local network you should see it appear in the list here.

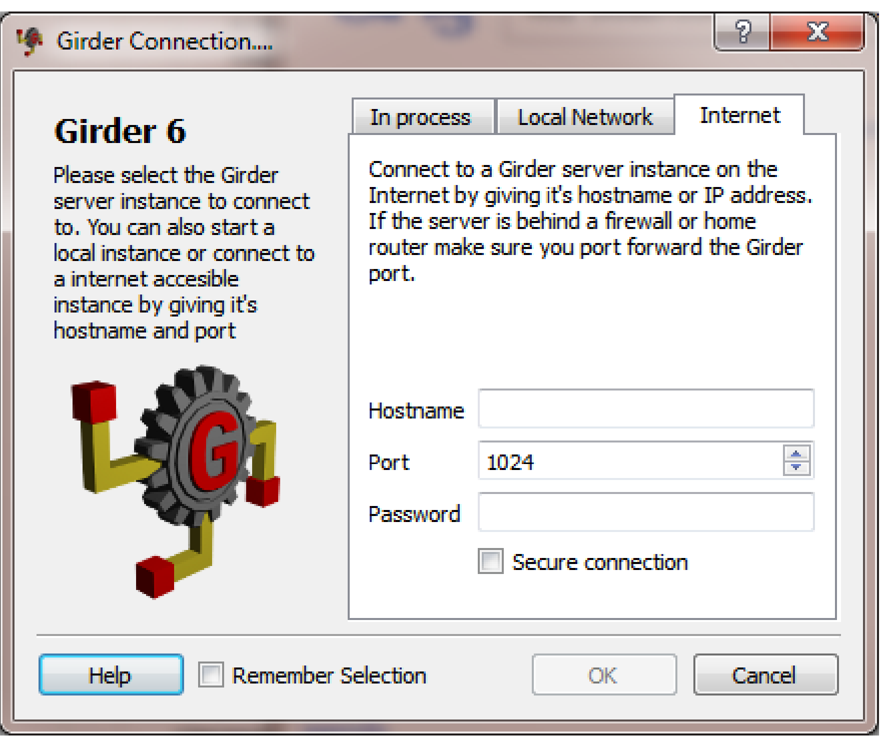


Local Network tab

To connect highlight the server (= Girder back-end) you'd like to connect to and enter it's password.

Internet

If the Girder back-end does not show up in your local network list you can still connect to it. You must then specify the hostname, port number and password.



Internet Tab

5 Concepts

5.1 Actions and Events

A big part of Girder's power comes from the Action and Event model of configuration. Actions are things Girder can do. For example it can send an IR code, close a relay, or press a button on an application on your computer. These actions could turn a TV on/off (IR codes), open a door (automatic door opener) or control your PowerPoint presentation.

Events are things that can make an action trigger. For example: The doorbell rings (this is an event) or a motion detector, detects motion. Other events include events fired by the scheduler at certain times.

By themselves actions and events are pretty useless, but place them together and we have an incredibly flexible way of configuring automation. Take the motion sensor for example. Maybe you have a room that has a TV in it that you want to have come on automatically when you walk in.

Inside Girder you'll see an event arrive in the Logger Window. You can previously created an action that sends the power on signal to the TV. Now all you need to do is drag the event from the logger onto the newly created IR action! This teaches Girder that when the motion sensor is triggered it should send out the IR codes.

5.2 IR Codes

Together with a PIR-1 or a PIR-4 Girder can be used to control all kinds of equipment that is operated with a remote control. For this to work with a PIR this remote has to be a IR (infra red) remote. Typically 90% of remotes out there are IR remotes.

If you are not certain your remote is infra red an easy way to test is by being close to the device you are controlling but covering the remote with a towel while pressing the button. If the equipment still responds it's likely to not be a IR remote as they need line of sight to work.

Next we'll need to get a IR (or CCF) code. This is a long list of numbers that represent the timings of the signal that is sent out. Typically you don't need to know any details about these. The PIR-1 can learn IR codes for you. The PIR-4 can only send.

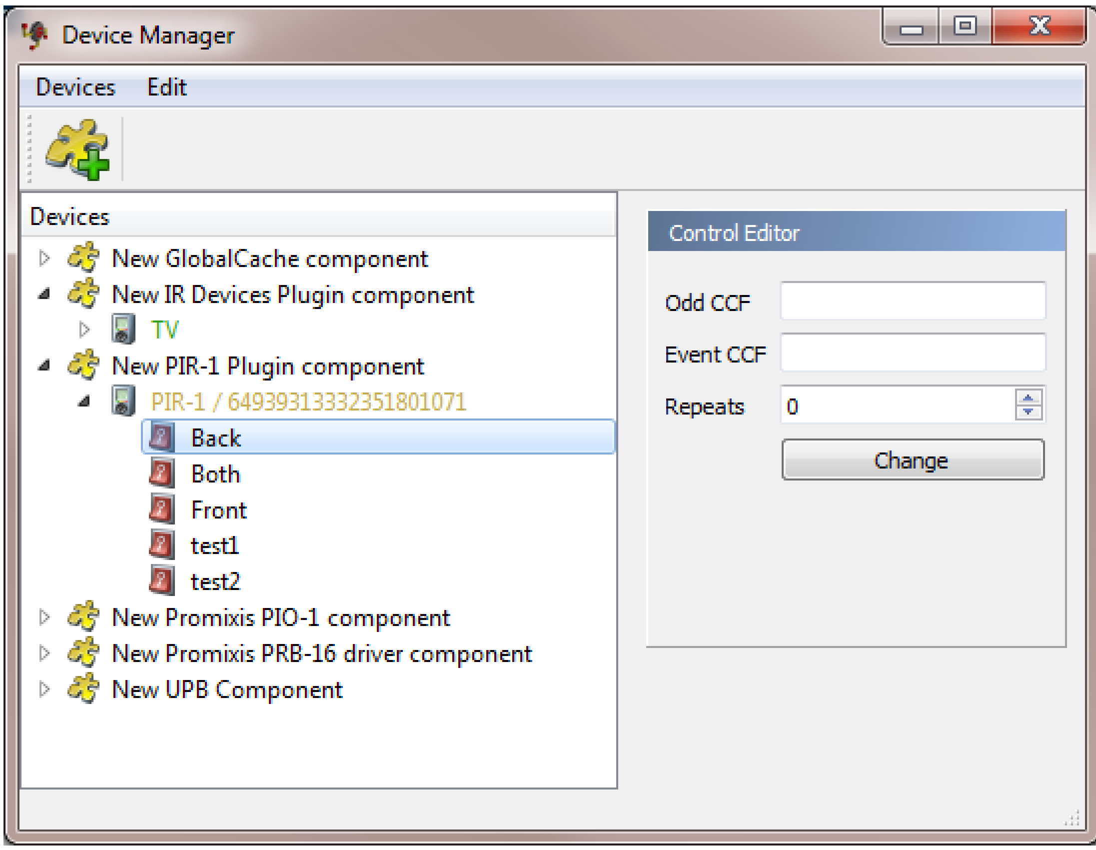
5.3 Device Manager

The Device Manager concept was created to have a digital representation of actions you can do in your surroundings. For example opening an automatic door would require you to push a button on the wall. This would have a button inside Girder's Device Manager as well. Also things like amplifiers or TV's. These all have buttons and sliders (think volume). These all translate to the Device Manager framework inside Girder. Inside Girder these can even be organized into locations. These could match for example the names of rooms in your house.

To get an overview of what you currently have defined open the device manager. View->Device Manager.

The interface

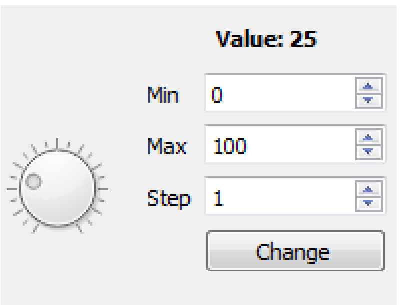
© 2015 Promixis, LLC



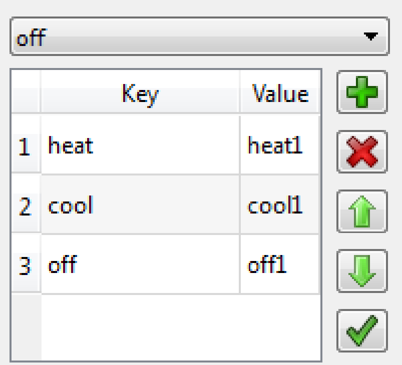
Device Manager

On the Device Manager interface you'll find all device currently defined grouped by component. The Component is the part that holds all it's devices. In the example above the "New PIR-1 Plugin component" is a component. Inside it you'll see the "PIR-1 / 6493931332351801071" device. It's currently not green, an indication it's not available. Last but not least each device will have controls. The controls are the actual parts that you'll interact with. In the example above the control is a "CCF control". This is an advanced control that allows you to send IR codes. Some of the other controls are visible below:

Toggle Button Control



Range Control



List Control

These control types are assigned upon creation of the control.

Referencing Controls from Actions or Lua code

All elements in the Device Manager have unique id's internally. You can use these to reference the Controls. Alternatively you can reference Controls by their Path. The Path consists of three main parts. The location name, the device name and the control name. If the location is nested in another location you'll see all location names listed. These parts are separated by "///" three forward slashes. Make sure you don't use three forward slashes in names if you want to use Path names to reference Controls.

There are a few pro's and con's for using either method.

Id's don't change and are unique. So you can change the name of any part of the Control, Device or Location name and your reference by Id will continue to work as normal. However if you transfer to a new computer and recreate the Device Manager you'll find that the Ids have changed. (You could always copy the database from one computer to another to avoid this!)

Path references are nice since you can transfer these from one computer to another, as long as you kept the names the same things will continue to work.

5.4 Sessions

Girder is split up into a back-end and a front-end. The back-end runs the automation and interacts with the hardware. The front-end takes care of the interaction with the user and or applications running on the user's desktop. Some actions like for example mouse move actions run on the users desktop. Since multiple front-ends can be connected to one back-end its necessary to tell Girder which desktop to target. This is called the session. Typically these are in the form of HOSTNAME|USERNAME and all attached front-end sessions should show up in the session box. If you'd like to have an action run on all hosts with user "RON" set the session to *|RON. If you wan to target all users on hostname BLACKBOX use session BLACKBOX|*

© 2015 Promixis, LLC

6 Tutorials

These tutorials should slowly get you on your way using Girder.

6.1 Creating your first action

The tutorial will go through the steps of creating your first action inside Girder.

1. Enabling required plugins

For this tutorial we'll use the keyboard plugin as the event generator. You can substitute any other event source here. So follow the "Enabling Plugins" tutorial to enable the "Keyboard Input" plugin.

Once that is done you should look at the logger tab at the bottom of the main Girder window and press a few keys on the keyboard. You'll notice that events are generated for each keypress.

2. Add a GML

GML's are the main containers that hold the actions and events. So start by click "File -> New". You'll see a new node appear in the Action Tree on the right. Click on this action tree. Inside the action tree you'll find a group called "New Group", click on this.

3. Add the action

Now to actually add the action find the "Scripting" action in the "Available Actions" list on the left. (You might have to click on the wrench symbol with the word "Actions" underneath). Once you've found it double click on the "Scripting" node. You'll notice that the "New Group" has a new node in it. Opening the "New Group" node you'll find the "Scripting" action.

4. Add some code to the scripting node.

Double click on the scripting action in the "Action Tree" in the "New Group" group. A new tab should appear at the bottom of the Girder interface with a big white text box. The title of the tab is something like "Action [/New File/New Group/Scripting]".

In the big box enter

```
print("Hello World")
```

and press "Apply and Test". If you did this right and look at the Lua Console tab it should have printed "Hello World" there.

5. Attach a keyboard event to the action.

© 2015 Promixis, LLC

Now we need to tell Girder when to fire that action. We enabled the keyboard input plugin earlier and we'll use it's event to trigger. Let's use the escape key. On the left hand side of the main Girder window there are three Icons arranged vertically. "Actions", "Conditionals" and "Events". Click on events. Press the escape key. Now open the "Keyboard Input" folder in the "Events" window. You should see a long list of events. Find the "Escape [On]" event. If you cannot find the "Keyboard Input" folder, make sure you did step 1 correctly.

Now for the tricky part. Drag and drop "Escape [On]" from the Events tree to the "Action Tree" right on top of the "Scripting" action we created earlier.

6. Test!

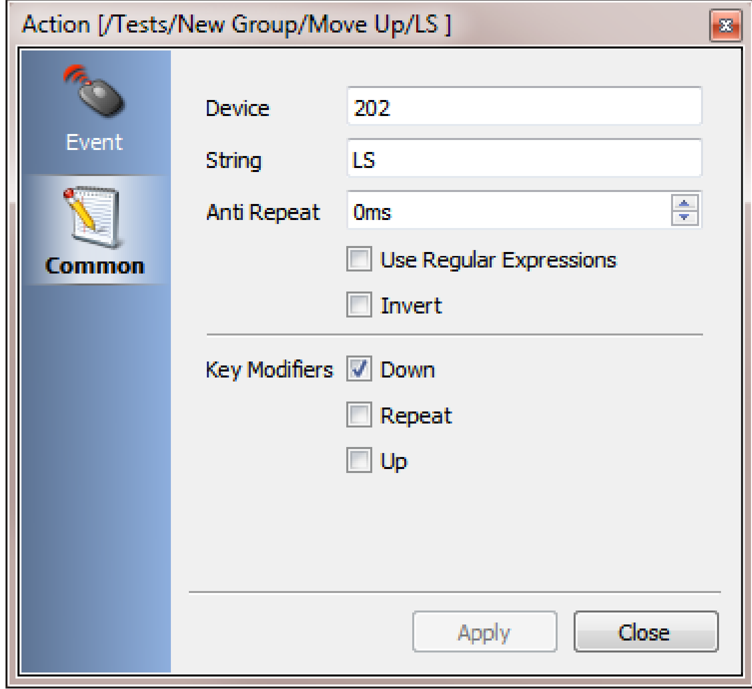
That should be all. Now open the Lua Console tab again at the bottom of the main Girder window and press the escape button. You should see "Hello World" appear every time you press escape.

6.2 Hook Events to Actions

There are a few different ways to hook events to actions. We'll describe 3 ways of doing it.

Manually

This is the hard way, which some people prefer, no judgment on our part there! Find the action you wish to add the event to. Click on the action. Next click "Edit->Add Event". Now fill in the various fields manually. Making sure you get the modifiers set correctly.

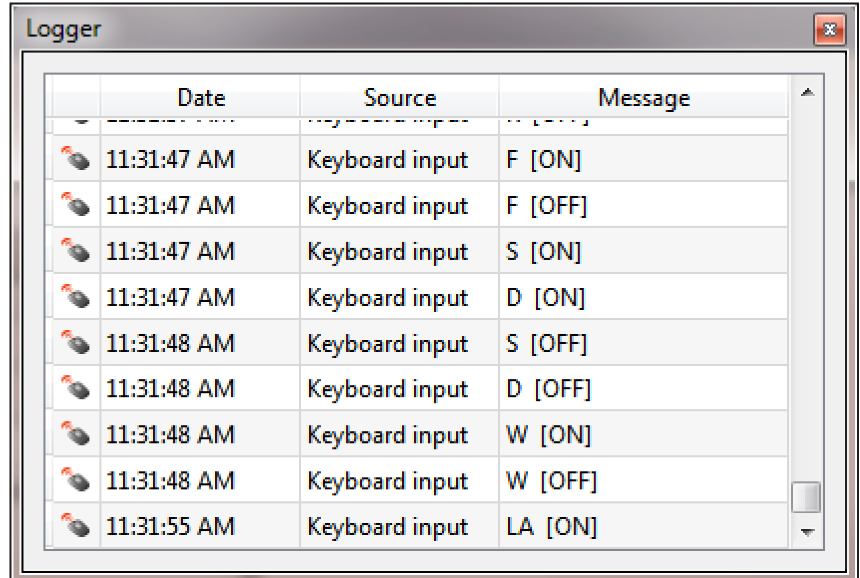


Event Editor

© 2015 Promixis, LLC

From the Log window

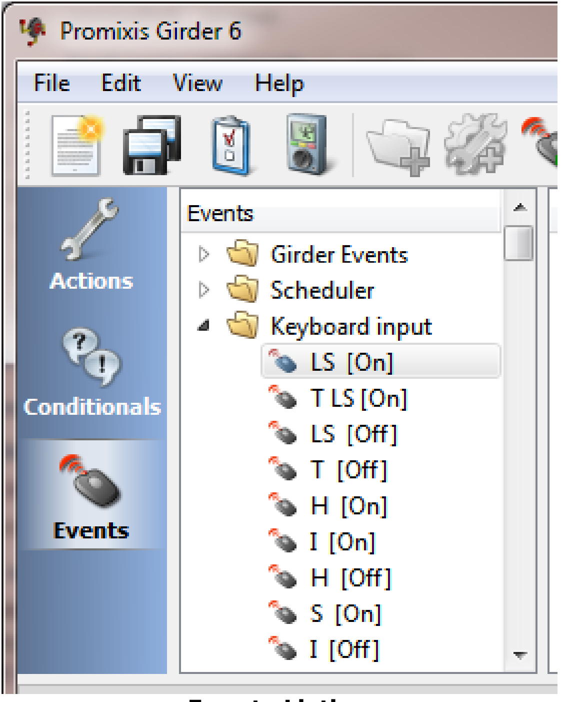
This is a quick and easy way to add events. Simply have the log window open. Trigger the event to happen, whatever it may be. For example the keyboard plugin triggers events when you type. Then drag and drop from the log window onto the action which should get the action!



Log Window

From the Events tree

The Events tree keeps a record of events that passed by recently. It organizes them by source. So all Girder events and Keyboard events are organized neatly. To add one of those events to an action. Highlight the action and double click the event. Or drag and drop it onto an action.



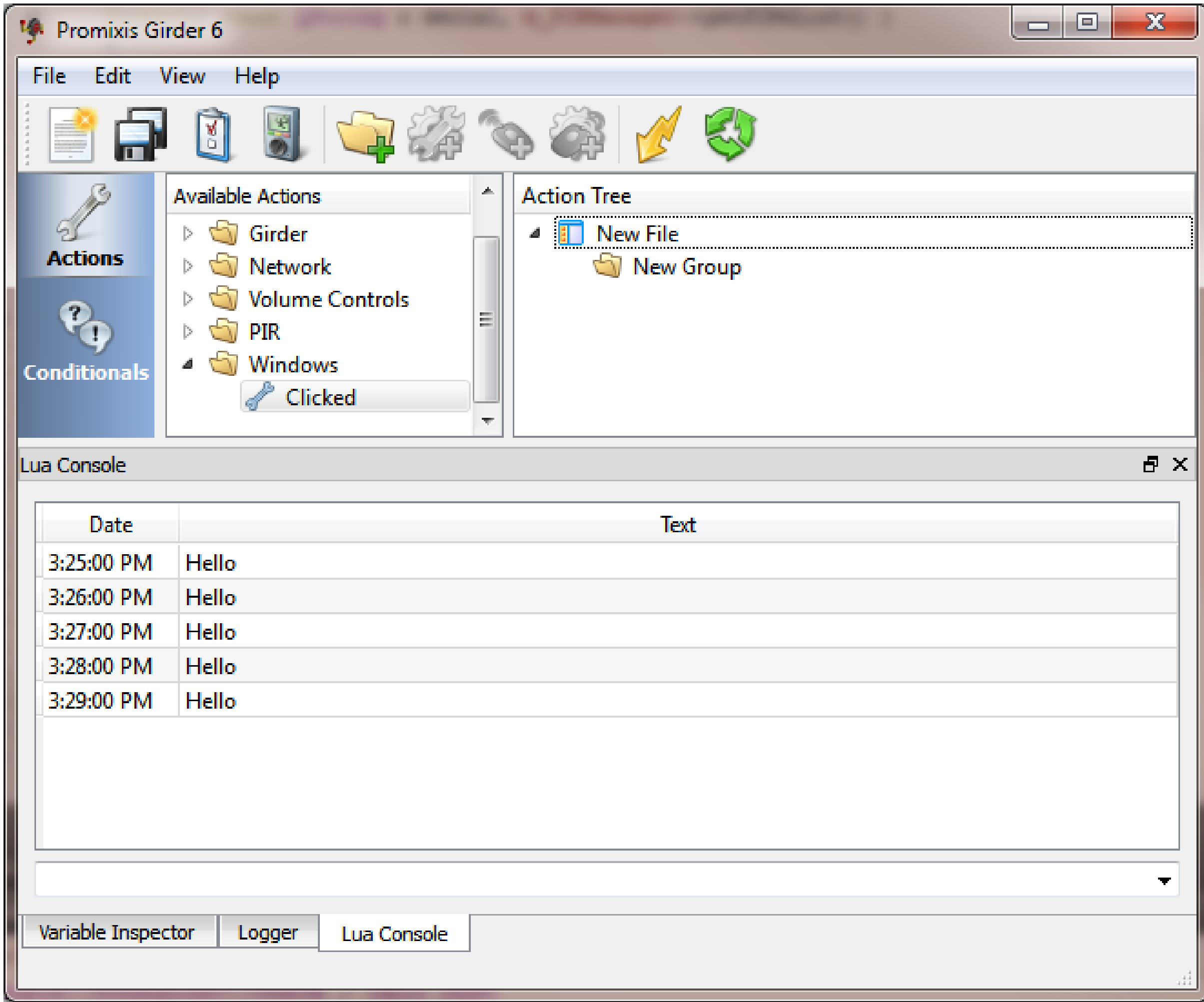
Events Listing

6.3 Pressing a button on Windows Calculator

Concepts: Actions and Events

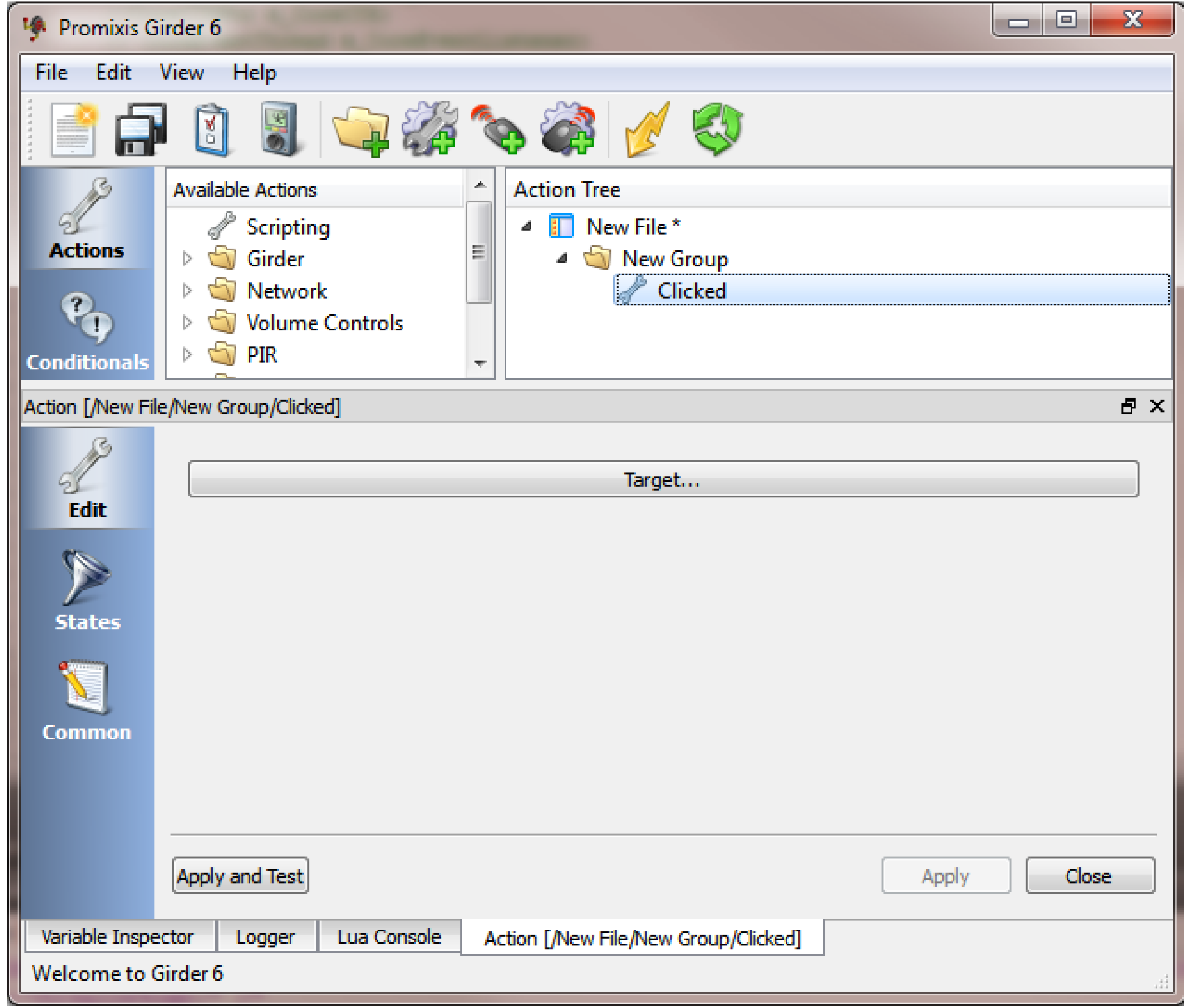
Girder stores it's configuration in an GML tree. The GML Tree lives on the main Girder window on the top-right hand side. In the image below you can see it displaying "New File" and "New Group" etc. On the top left-hand side you can find the available actions. These are actions that Girder can do but are not yet integrated in your configuration.

© 2015 Promixis, LLC



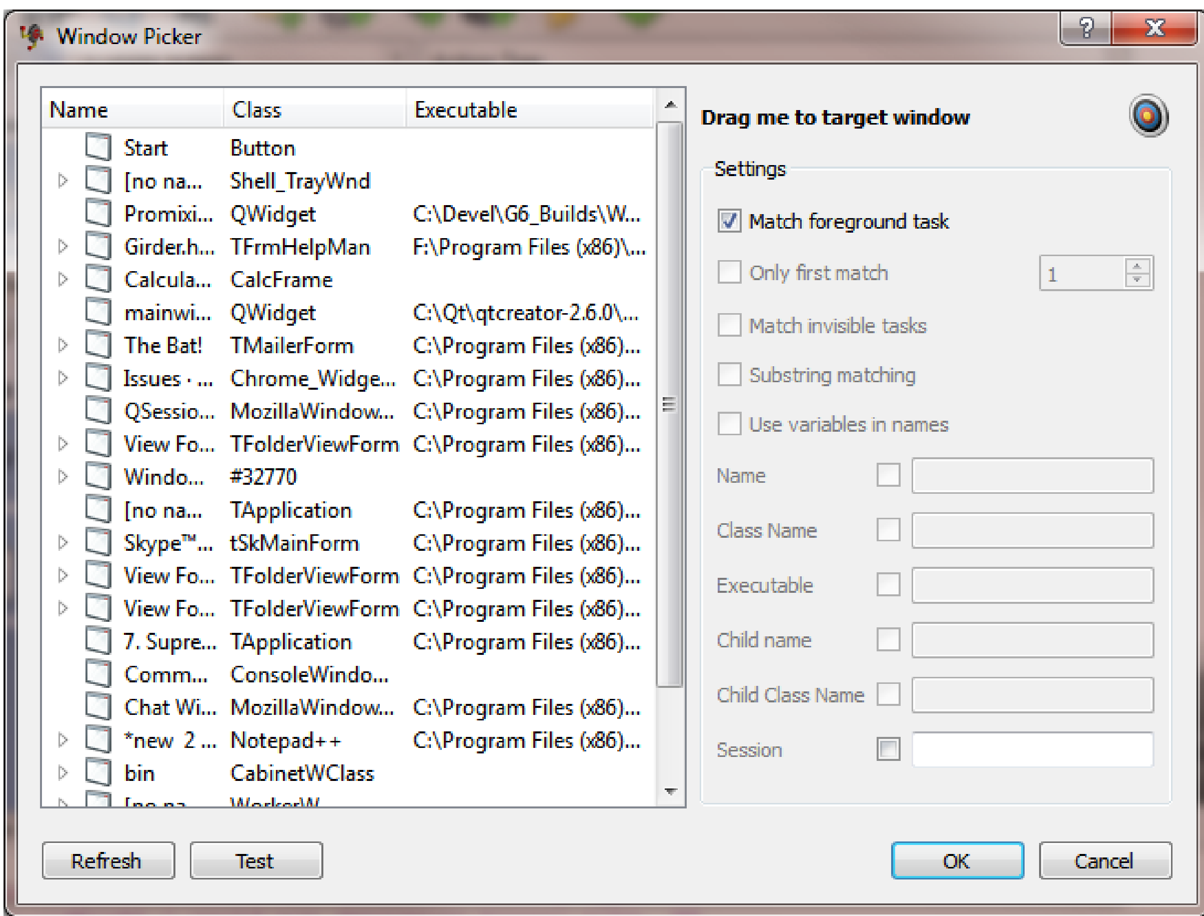
Girder's Main Window

If this is the first time you start Girder you most likely have an empty Girder Tree. Let's start a new GML for it. Click on File->New. You should see a "New File" node pop up. Click on the little triangle in front of it to expose the "New Group" as well. With the new group clicked find the "Windows" folder in the available actions box. Click on it's triangle to open it and find "Clicked". Double click it. This should make "New Group"'s icon have a little triangle as well. This means it has something inside as well. Click on that triangle to open it up. You'll see "Clicked" on the tree. Double click "Clicked".



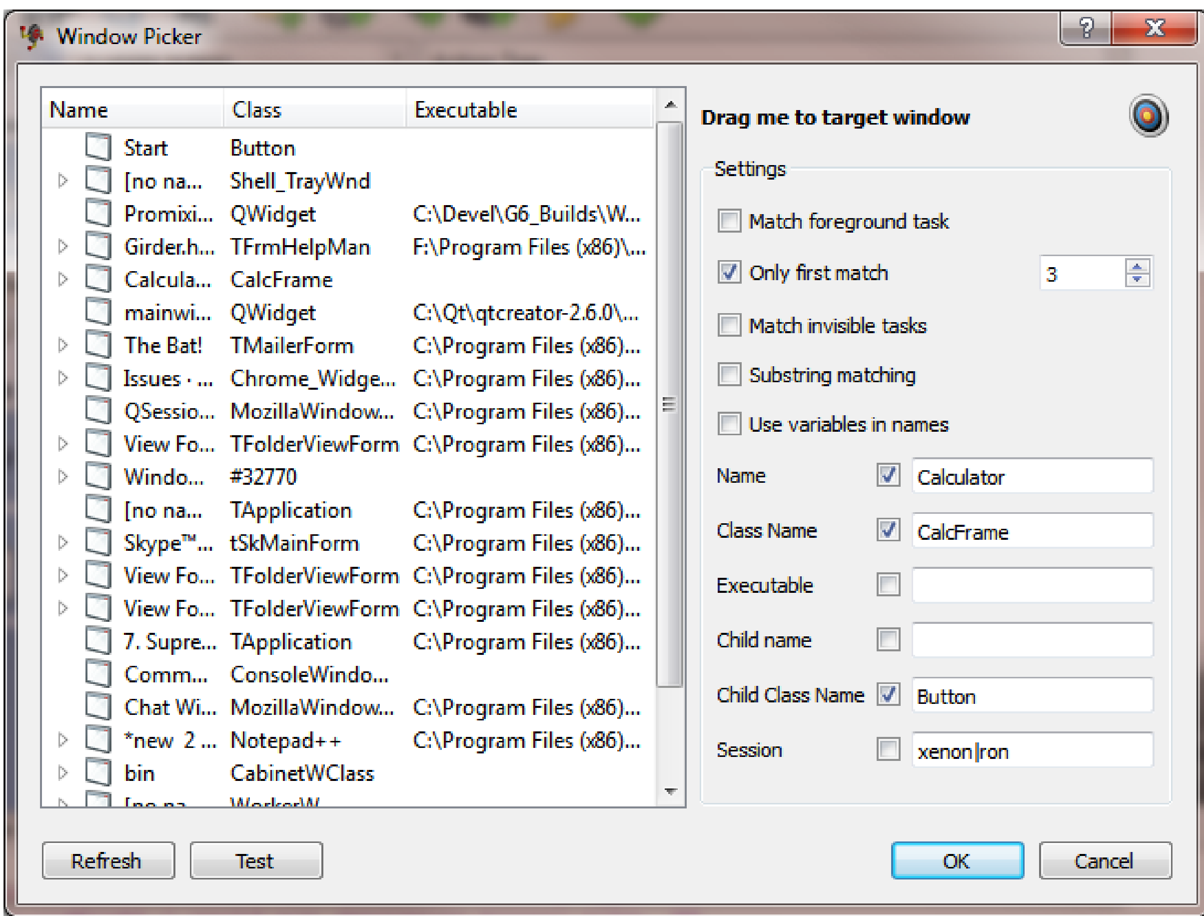
Girder's Main Window with action open.

You should now see a screen much like the one above. This is the action configuration. We'll now tell this action what item to click. Start the Windows Calculator. Then press the "Target..." button.



Window Picker dialog.

The easiest way to configure all these options is by clicking and dragging the bullseye from the top right corner to the button you'd like to press on the calculator. Let's say we want to click the number "7" button. Drag the bullseye to the number 7 button.



Window Picker configured for Windows Calculator

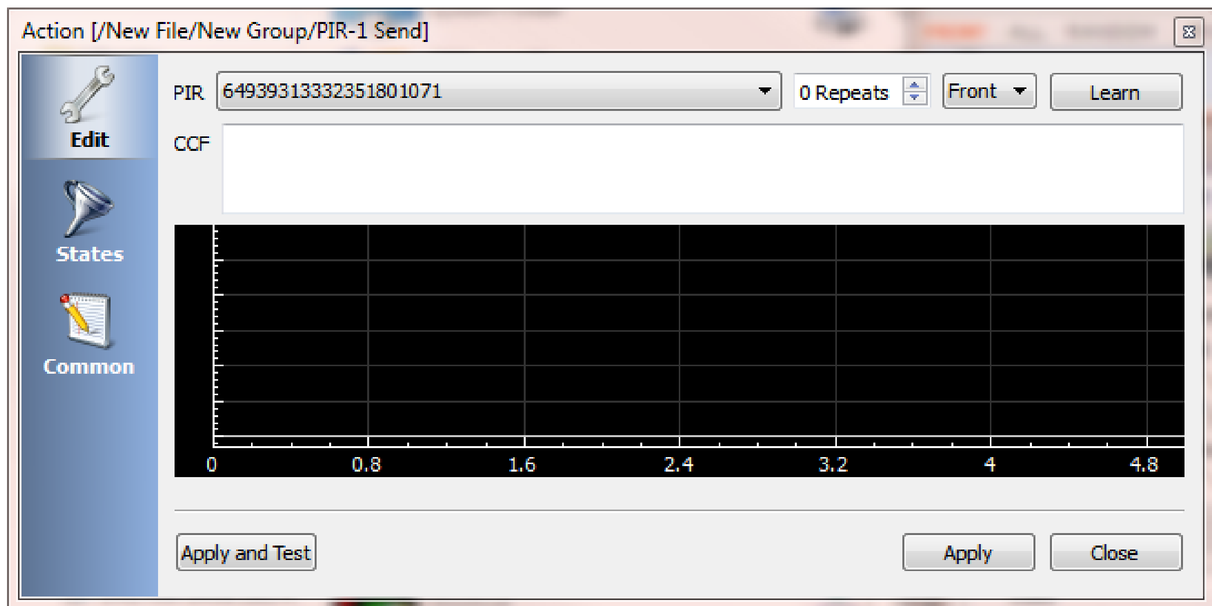
Feel free to hit the test button to make sure the targeting is correct. Click OK when satisfied. Now comes the fun part. Back on the action page click on "Apply and Test". Keep an eye on the Windows Calculator it's now pressing the 7 button automatically.

6.4 IR Codes in an action

Concepts: IR Codes, Actions and Events

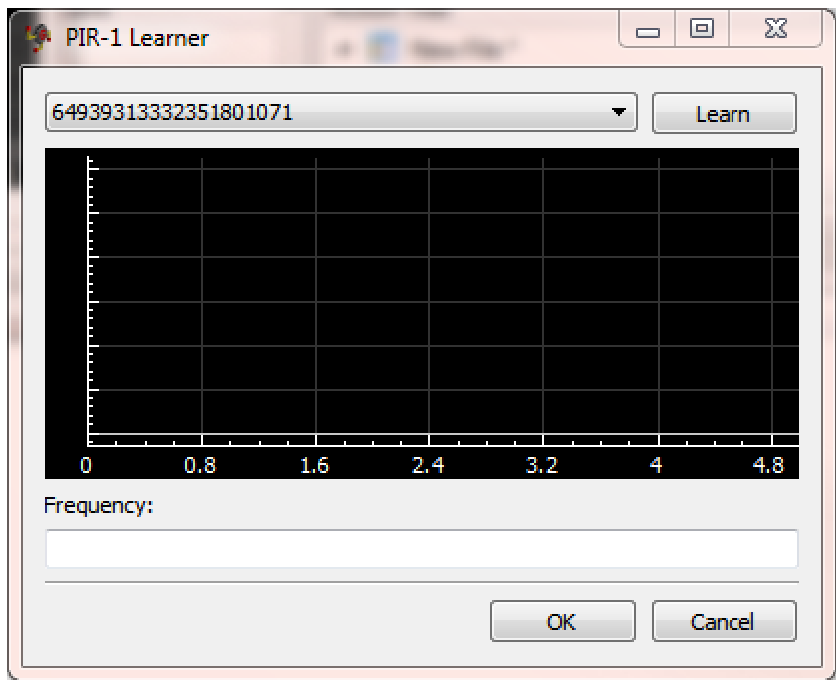
First let's add a PIR action to Girder's action tree. In the available actions open the PIR folder and double click on PIR-1 send. From the PIR-1 drop down list select the PIR-1 you wish to use as output.

© 2015 Promixis, LLC



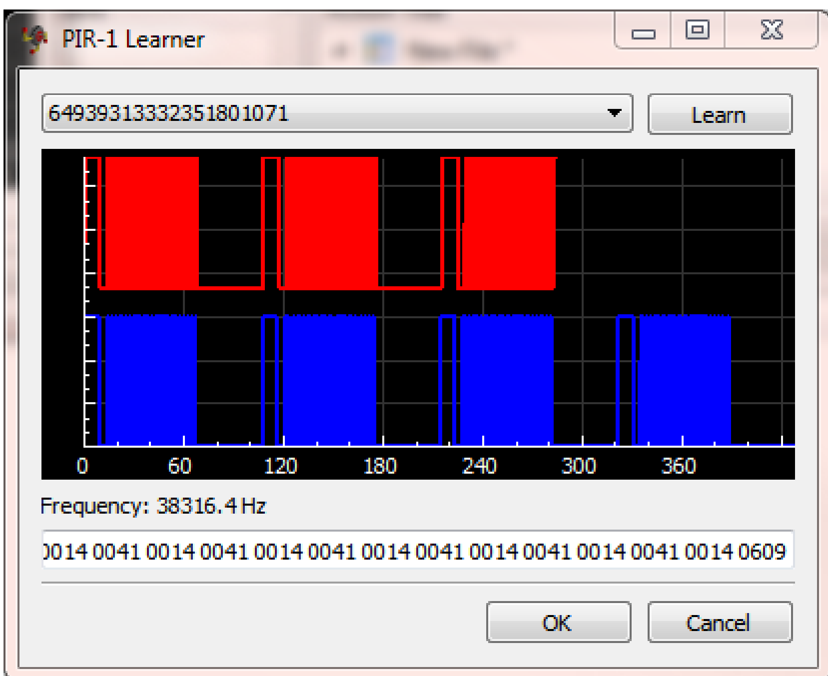
PIR-1 Send Action

Then click on "Learn". This should show a dialog as follows:



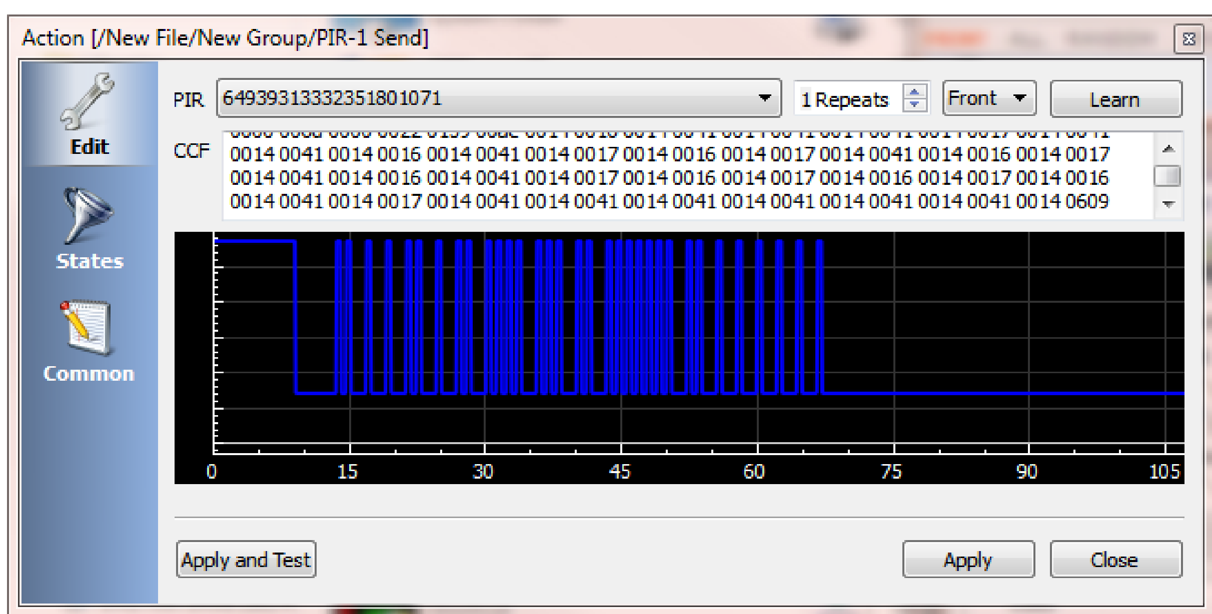
PIR-1 Learn Dialog

Press on learn and hold a remote by the PIR-1. Now press a button of your choice on the remote.



PIR-1 Learn Dialog with Learned code.

The Dialog should now look like this. Press OK.



PIR-1 Action with CCF Code

As you can see the CCF code has now appeared in the action. Note that I had to set the Repeats to 1 or higher for this particular code to work. If nothing shows up try that too. Now press "Apply and Test" to see if your learned code works!

6.5 IR Codes on web page

In this tutorial we'll show you how to put buttons on a web page and have these buttons trigger IR sending from a PIR-1. There are (at least) two approaches that can be taken.

© 2015 Promixis, LLC

The first is to learn the IR signals as in the "IR codes in an action" tutorial. The second approach is to use th "IR Devices" component. Both have their ups and downs. We'll describe both.
.

6.5.1 Webpage IR codes using Actions

For this tutorial please complete the "IR Codes in an Action" first. The next step is to create a new HTML file in the httpd directory of Girder. We'd like to assume you know a little bit about HTML and Javascript. If not please look at some of the introductory HTML tutorials on the web, there are many good ones.

Note that with a little bit of creativity you can see that this tutorial can also be used to learn how to place events on a web page, which is really all this is.

The web page

Let's say we created a file called "irbuttons.html" inside this we'll add the following bits and pieces to create a barebones HTML file with Girder integration.

```
<html>
      <head>
            <title>Girder - IR Codes on Webpage / gir.triggerEvent
example</title>
            <script src="gir/triggerEvent.lhtml"></script>
      </head>
      <body>

      <button onclick="gir.triggerEvent('ir button 1', 18 );">IR button
1</button>

      </body>
</html>
```

There are two lines that are of interest here. First the line:

```
<script src="gir/triggerEvent.lhtml"></script>
```

All this does is it includes the gir/triggerEvent.lhtml file into your page. This is where all the real magic happens. If you are curious go look at it. If not all you need to know is that this file exports a function into javascript called gir.triggerEvent(...). This function has the following parameters:

```
function gir.triggerEvent ( eventString, eventDevice, keyModifier,
payloads, cb )
```

- *eventString*, as the name implies is a string, a piece of text that we can randomly select. Pick something memorable link "IR Button 1".
- *eventDevice* is a number Girder uses to determine who is responsible for this event. In

this case we'll use 18, which is the magic number for Girder's own events (or external events like this).
- *keyModifier* is the number that determines if this is a "NONE" (=0), "ON" (=1), "REPEAT" (=4) or "OFF" (=2).
- *payloads* is an array of strings that should be passed to Girder as a payload.
- *cb* is a callback parameters. Pass a function to this if you'd like to be notified of success or failure.

This function we use on the second line of interest in the HTML file:

```
<button onclick="gir.triggerEvent('IR Button 1', 18 );">IR button 1</
button>
```

This tells HTML to create a button element and when it is clicked it calls our gir.triggerEvent function!

Hooking it all together

Now that we have a very basic we page we'd like pressing that button to do something. You might have noticed that pressing the button on that page will trigger an event in Girder, specifically the "IR Button 1" event. All we need to do now is tell Girder to trigger the Send IR action when this event comes in. This of course is discussed in the "Hook Events to Actions" tutorial. Press the button on the web page after you hooked the event to the IR action and you should magically now be sending IR codes out of your PIR-1! Congratulations.

You can find this also as an example in triggerEvent.html in the httpd directory

6.5.2 Webpage IR codes using the Device Manager

For this to work we first need to setup a IR Devices, device. The has a few steps.

1. Enable the IR Devices Plugin on the File->settings->plugins
2. Go to the Device Manager (View->Devices)
3. If not already there add a IR Devices Component (Edit->Add Component, select the IR Devices plugin)
4. Click on the IR Devices component
5. Add a new Device, (Edit->Add Device) any name you like.
6. Click on that newly created device
7. Either manually add controls and learn IR codes or import from web.

Now that we have controls defined in our newly created device, click on the control you wish to trigger from your web page. In the status bar you should see the ID of this control. For example "Control 8". That means we can use the id 8 as a reference to this control. This ID is stored in a disk database and is the same even after reboots. The only time it changes is when you delete the control or device.

Create the HTML file.

As before we'll assume that you have some understanding of HTML. Create this HTML file in the httpd directory of Girder. Name it control.html

© 2015 Promixis, LLC

```
<html>
      <head>
            <title>Girder - IR Codes on Webpage / gir.deviceManager
example</title>
            <script src="gir/deviceManager.lhtml"></script>
      </head>
      <body>
      <h1>IR Devices</h1>
            <button onclick="deviceManager.requestControlValueChange( 8,
1, 'Web Interface' );">CLICK ME</BUTTON>
            <p>
                  Edit control.html in the httpd directory and change the
first parameter (the value 8 in this file) of requestControlValueChange
to the ID of the control you wish to press.
            </p>
      </body>
</html>
```

There are two parts that are interesting in there. The first is the loading of the device manager helper functions.

```
<script src="gir/deviceManager.lhtml"></script>
```

The second part actually presses the virtual control button. Note the first parameter to "requestControlValueChange". It has value 8. This is the id of the control. We found the value early by clicking on the control we like to control and reading it's value in the status bar.

```
<button onclick="deviceManager.requestControlValueChange( 8, 1, 'Web
Interface' );">CLICK ME</BUTTON>
```

Making it dynamic (ADVANCED)

Of course things change and the control might go away and you'd need to change your id. Instead of hard coding this value you can also use the deviceManager function available to javascript to dynamically create a page. Below is an example on how this is done. This is a bit more advanced. If the deeply nested asynchronous code scares you just move right along.

```
<html>
      <head>
            <title>Girder - IR Codes on Webpage / gir.deviceManager
example</title>
            <script src="gir/deviceManager.lhtml"></script>
            <script src="http://code.jquery.com/jquery-1.10.1.min.js"></
script>
            <script type="text/javascript">
```

```
                    /* A little Javascript magic to factor out
                       some repeated code. Basically it takes the result
                       of the deviceManager.XXX functions ( which is a
table )
                       and turns this into a repeated calling of a
callback

                       This avoid repeating the getItems code over and over
                       nesting inside eachother. If this is nebulous,
don't
                       worry just use it for now.

                       */
                    function getItems ( f, id, cb ) {
                          if ( !cb ) {
                                return;
                          }
                          f(id, function( result, items ) {
                                if ( !result ) {
                                      return;
                                }
                                for ( var id in items ) {
                                      if ( items.hasOwnProperty( id ) ) {

                                            cb( items[id] );
                                      }
                                }
                          });
                    }

                    function getComponents ( id, cb ) {
                          return getItems(deviceManager.components, id,
cb);
                    }

                    function getDevices ( id, cb ) {
                          return getItems(deviceManager.devices, id, cb);
                    }

                    function getControls ( id, cb ) {
                          return getItems(deviceManager.controls, id, cb);
                    }


                    $(document).ready( function () {

                          var c = $("#componentsDiv");

                          // 130 is the plugin id of the IR Devices plugin.
```

© 2015 Promixis, LLC

```
                        getComponents( 130, function ( component ) {

                              var cH2 = $("<h2>");
                              cH2.text( component.name );

                              c.append(cH2);

                              getDevices( component.id, function
( device ) {

                                    var cH3 = $("<H3>");
                                    cH3.text( device.name );
                                    c.append(cH3);

                                    getControls( device.id,
function(control) {

                                          var button = $("<BUTTON>");
                                          button.text ( control.name );
                                          button.click ( function () {


deviceManager.requestControlValueChange( control.id, 1, "Web
Interface" );
                                          });
                                          c.append(button);


                                    });

                              });

                        });

                  });

            </script>
      </head>

      <body>

      <h1>IR Devices</h1>

      <div id="componentsDiv">
      </div>

      </body>
</html>
```

6.6 Control mouse from remote

This tutorial we're going to see how to make Girder control the mouse on your screen by using a standard remote you have around the house.

Requirements: PIR-1

Make sure you have a GML open that we can use, if not add a GML by clicking "File->New".

Let's create a new group to hold the various action together. Let's call it "Mouse Control". Creating groups is always a good policy as it will keep the GML nice and ordered. Once you have 100 actions, you'll thank me if you grouped things.

Next step is to add our first mouse action. Let's start with "Mouse Up". Click on the "Mouse Control" group to make sure it's highlighted. Then go to the "Available Action" tree open the "Mouse" folder and double click on "Move Up". You should see a new action appear in the "Mouse Control" group. Double click on the newly created "Move Up" action in the Action Tree. The defaults are mostly good. Just set the session to "*" for now (or pick a session of your choice).

Next we'll need to hook the button on the remote to the "Move Up" action. The easiest way to do this is by clicking the desired button on the remote while pointing at the PIR-1. Click on the "Events" button in the left hand side bar. The PIR-1 plugin should be listed, if you open the PIR-1 plugin folder you should see the IR events. Pick the event you just pressed. (or use the logger view and drag and drop). Drag and drop this event onto the "Move Up" action.

This is all that is needed to control mouse move up! Test it out by pressing the button on your remote again.

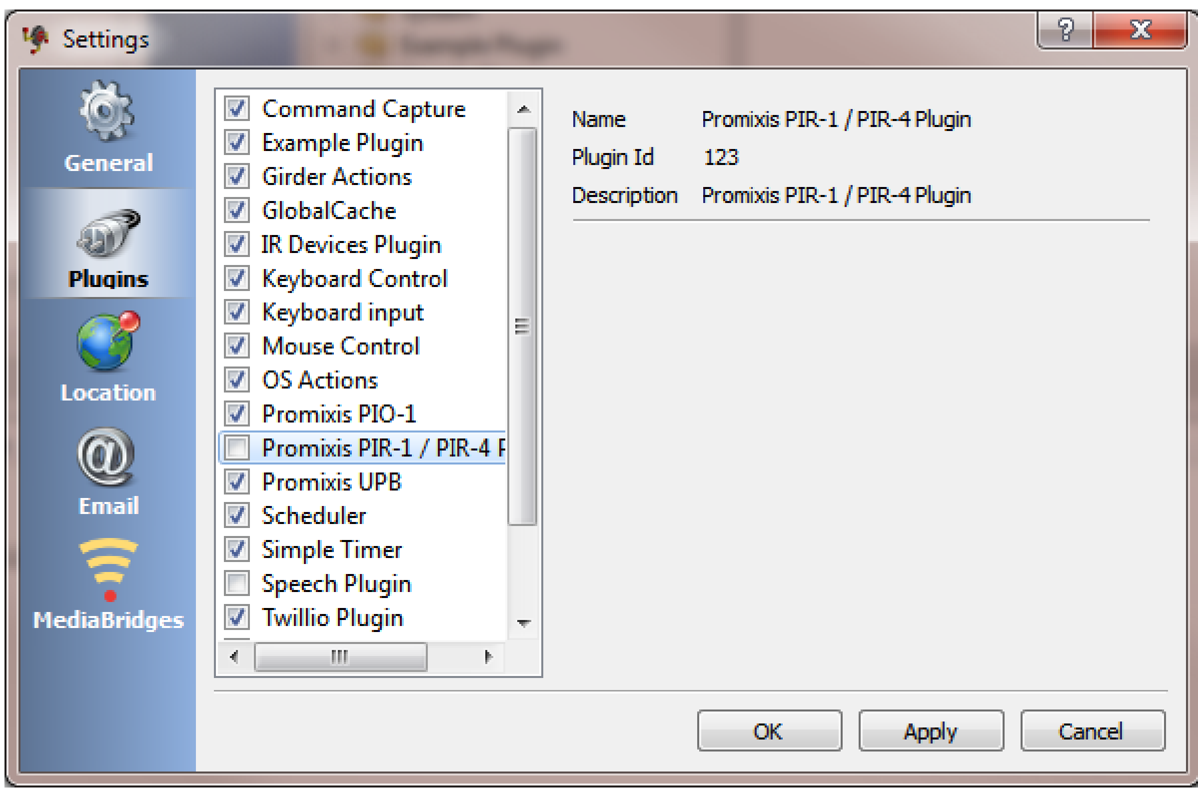
6.7 Enabling Plugins

Much of Girder's functionality is contained in plugins. This allows Girder to only load the functionality you need and not waste computer resources. This tutorial will show you how to enable a plugin. In this case the PIR-1/PIR-4 plugin.

The Settings Dialog

The settings dialog can be opened by clicking File -> Settings on the main Girder window.

© 2015 Promixis, LLC



Settings Dialog

The settings dialog has a few icons going on the left hand side down. Click on the "Plugins" icon to show the list of available plugins. Note that this list will vary on your installation from the picture here.

In the list find the plugin you want. Then click the little checkbox in front of the name. That should do the trick. Click OK to close the dialog.

6.8 Importing IR codes from the web

Promixis provides a library of IR codes that Girder can download automatically into the IR Devices Component. Once the IR codes are imported you can trigger them from an action, a web page or the NetRemote interface.

Let's walk through the steps required to make this work.

Enable your IR output device plugin

The first step is to make sure you have the IR output device enabled. This could be a PIR-1 or PIR-4. Check that the plugin is enabled. The second plugin we need enabled is the "IR Devices Plugin". Make sure this is enabled as well on the settings dialog.

Enable your IR output device component

To expose the IR output capability of your plugin to the device manager you must create

a component and device for it. In the case of the PIR-1 / PIR-4 all you need to do is to create a component of the PIR-1 / PIR-4 plugin. The plugin will automatically create a device for each plugged in PIR-1 or PIR-4.

Once you verified both plugins are enabled you can go to the next step. Creating an IR Devices component.

Creating an IR Devices component

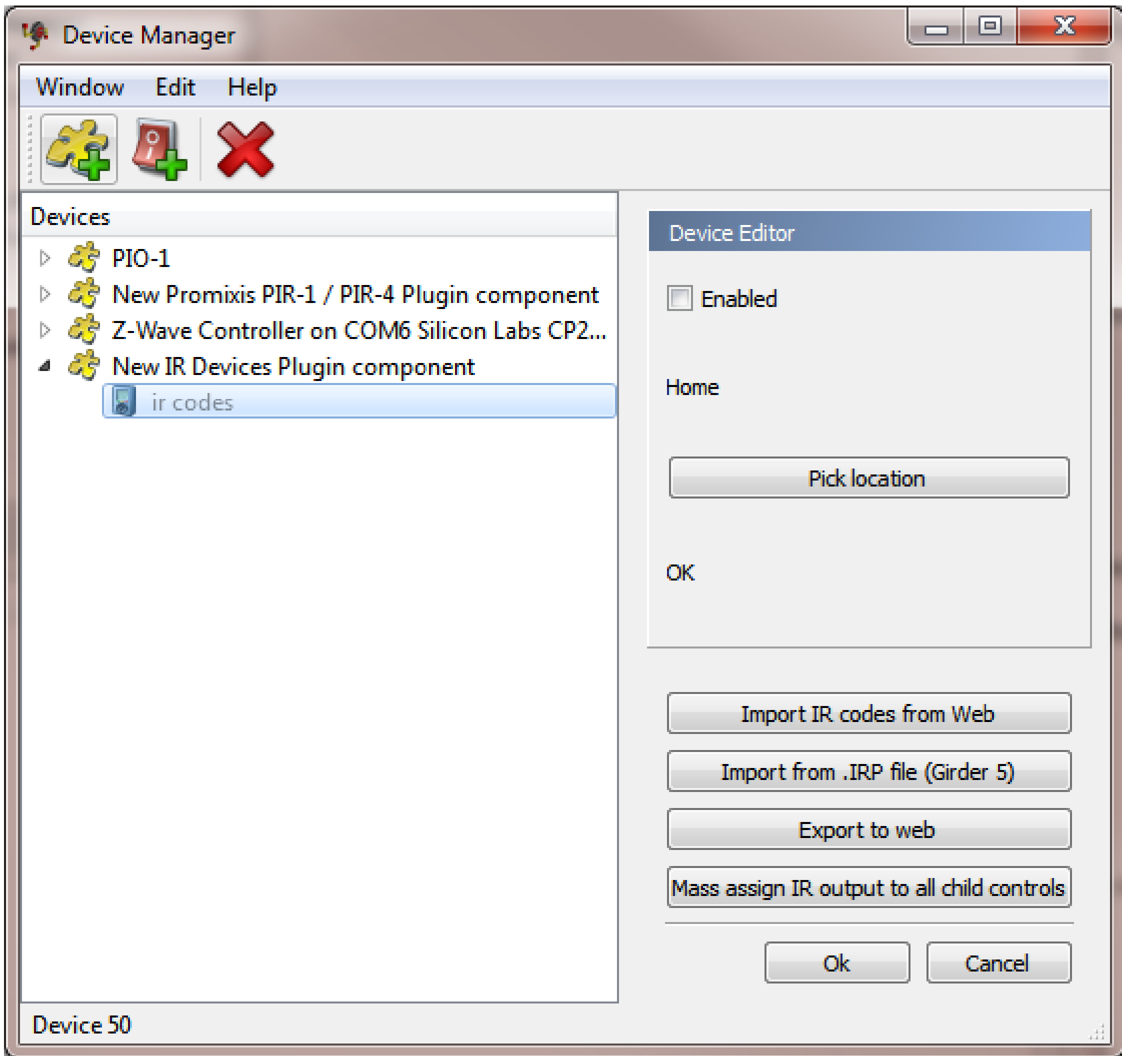
The device manager organizes device and controls inside components. In the case of the the IR Devices plugin you'll only need to create a component once. For some plugins it makes sense to have multiple components but for not for this plugin. So stick with one component.
To create it click "Edit -> Add Component". You should see "New IR Devices Plugin Component". To rename any of the items on this tree click on them, then press F2.

Make sure it's enabled by click on the 'Enabled' checkbox while the component is highlighted.

Add a device

IR Codes need to be contained inside a "Device". So highlight the "New IR Devices Plugin Component" node and click "Edit -> Add Device". Give it a name of your liking and click OK. Click on the newly created device, you should see something similar to the following dialog:

© 2015 Promixis, LLC



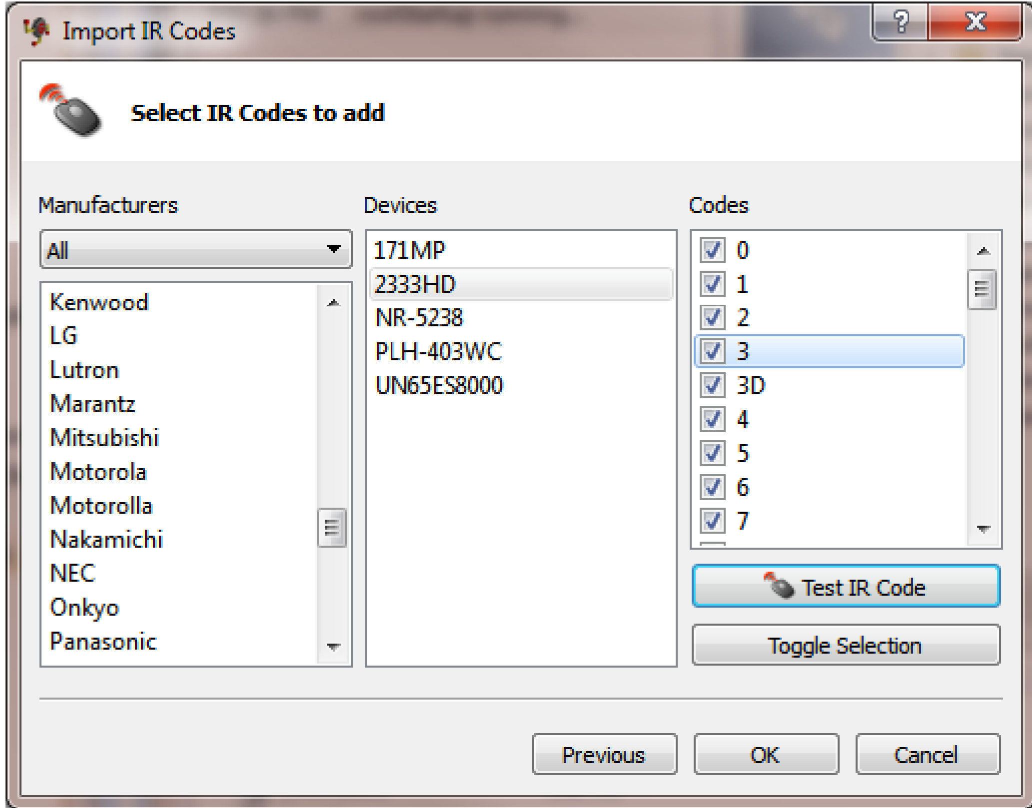
New Device

Notice that again the newly created device is disable by default, so click "Enabled"!

Next we notice we have a few options available. "Import IR codes from web", "Import from .IRP file", "Export to web", "Mass assign IR output to all child controls". We'll now do the actual import.

Importing

Importing first asks what device you will use to actually transmit the IR codes. Pick the device and the output port from the first page of the wizard.



web import

Next we can pick the manufacturer and the device model. Note that often even though we don't have an exact match you can use a IR profile for the same kind of device from the same manufacturer for a different model. For example the Samsung 2333HD might work with several other models, just try it out by selecting the IR code and hitting "Test IR code".

6.9 Event Forwarding

If you have multiple Girders running you might want to have events automatically forwarded to Girders running on different machines. With Girder Pro this can be done with a few simply steps. This example will show how to forward all PIR-1 events.

We'll be using the gir/triggerEvent.lhtml script to trigger events in the remote Girders, so make sure you enable the webserver on each Girder. Then collect the IP addresses of the remote Girders.

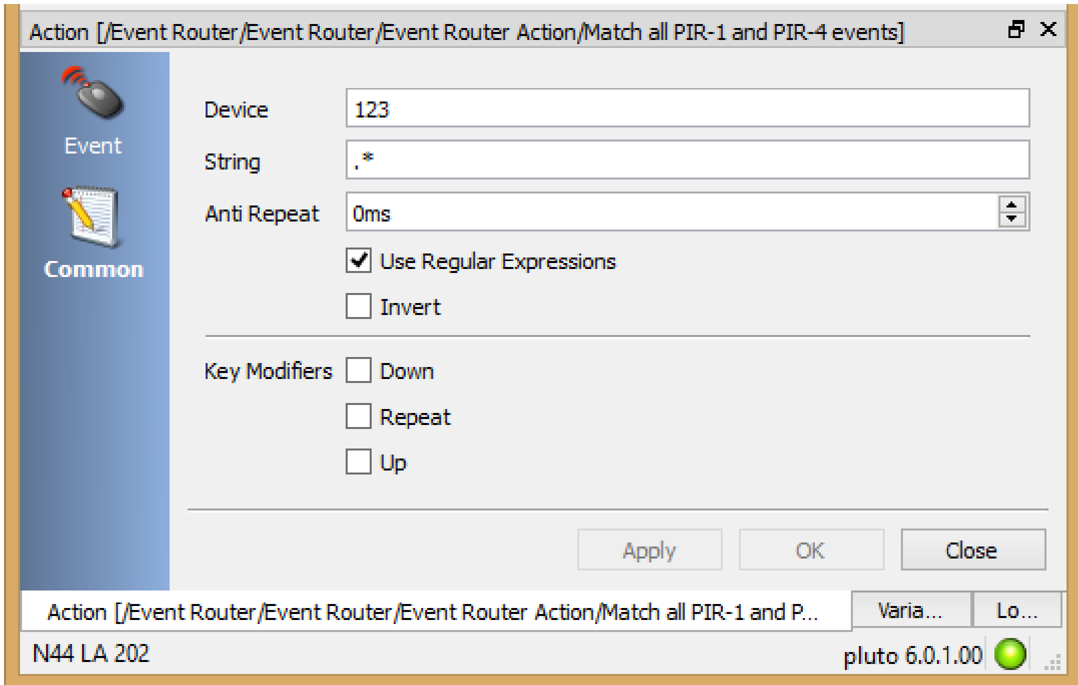
On the Girder instance that has the event source, in this case the PIR-1 attached add a scripting action to the tree and enter this script:

```
if not event then
    print("This action must be triggered by an event.")
    return
end
```

```
require("socket.url")

-- Edit this list to suit the Girder's you have running
-- make sure those Girders have the webserver enabled.

-- Adding localhost is probably not a good idea as you will
-- get an infinite loop. You have been warned.


local pcs = {
 [1] = "192.168.1.101:80",
 [2] = "192.168.1.102:80"
}

for index, host in ipairs(pcs) do

    network.get( "http://" .. host .. "/gir/triggerEvent.lhtml?es=" ..
socket.url.escape(event.event) .. "&ed=" ..
socket.url.escape(event.device+1) .. "&km=" .. event.modifier )

end
```

The important part for you to adjust is the pcs table. Make sure the entries match your IP addresses, you can have as many entries as you need here. Also please don't add the current instance as that will lead to infinite event loops, which will make Girder very busy shuffling events around to no avail.

The last part we need to take care of is the event node on the tree that will trigger this action. Since we want all PIR-1 events to trigger the action we'll use a regular expression to match all events. (.*)



That is all that you'll need to setup to make this work!

7 Send events to Girder

To send events to Girder over the network you can use the commandline tool csevent. It can be found in the Girder installation directory.

The commandline parameters are:

```
csevent hostname <secure> port password eventstring device <payload1> <payload2>
... <payload n>
```

Example

```
csevent 127.0.0.1 1024 girder hello 18 payload1 payload2
```

This will trigger the event "hello" from device 18 (described as "Girder" in the event log source column) with 2 payloads filled with payload1 and payload2.

8 Actions

In this chapter you'll find detailed descriptions of the actions that are built into Girder.

8.1 Command Capture

Command Capture is a way to spy on the inner workings of other applications and recreate some of the actions that happen inside. The Windows operating system internally uses messages to make all the user interaction happen. These messages have names like WM_LBUTTONDOWN, WM_COMMAND etc. These are not exposed to the end user and only happen behind the scenes. However with the use of command capture you can now catch those messages in real time and store them. Once captured you can then recreate them at any time.

Note that since we are grabbing deep into the inner workings of an application in an unexpected way things might not always work. Trial and error work best here.

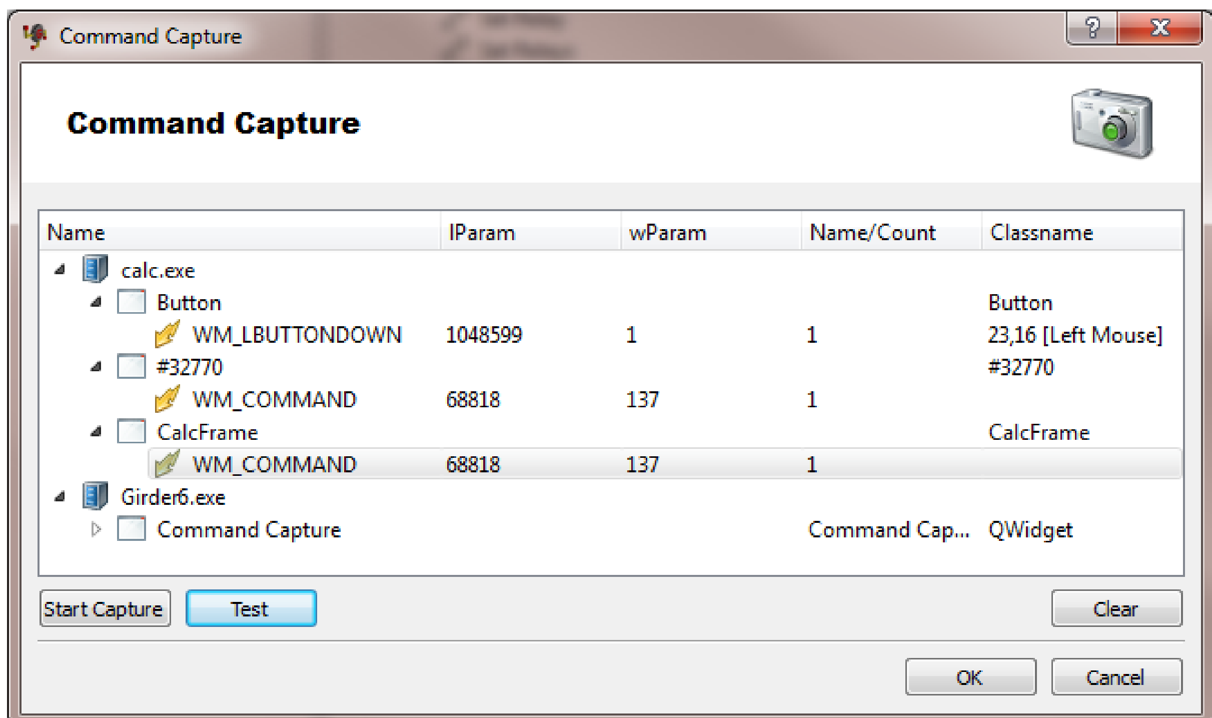
Parameters

This action has 4 parameters plus targeting. The first 3 parameters you may set by hand but since these values are completely undocumented the only way to find out their value is by actually capturing the values. The last parameter is the choice between SendMessage and PostMessage. We suggest going with PostMessage. The reason for this is that PostMessage does not wait for the target application to process the message. SendMessage potentially blocks Girder until the remote app responds.

Capturing

Capturing messages is done by clicking on the "Capture..." button. This will show a dialog like the one below. Press "Start Capture" to actually start capturing the messages.

© 2015 Promixis, LLC

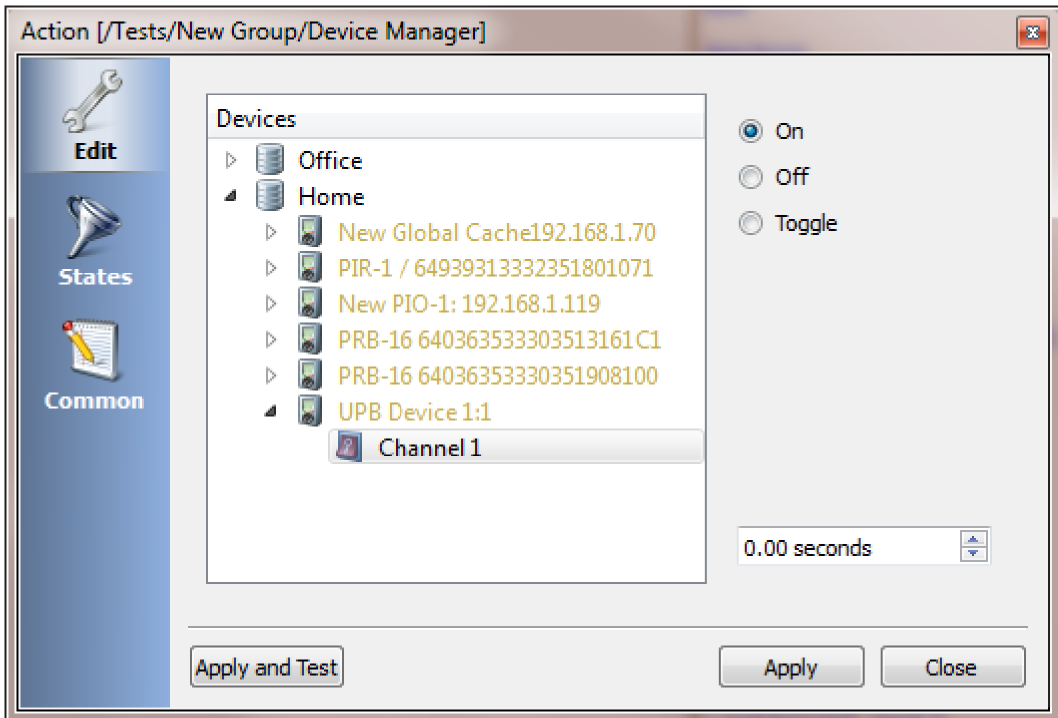


Command Capture in progress

The dialog above shows what it looks like when you press the "7" button on the Windows Calculator. It generated 3 events. The first one is the mouse message and the last two are the actually WM_COMMAND that triggers the internal code to add a 7 to the display. Highlight one of them and press test to see if it works. For Windows 7 64 bit calculator any of them will work. For the application that you are trying to automate you might have to experiment a bit. Try by clicking the desired button, maybe try using the menu bar or shortcut keyboard action. Basically this is trial and error to find the right way to capture. Sometimes it's not possible to capture and you'll have to go with keyboard or mouse actions.

8.2 Device Manager

The device manager action will allow you to change control values from the action tree.



Device Manager Action Dialog

To use this action select a control you wish to manipulate and then pick the operation to perform on it. If the value should revert to where it started set the timeout to a value you desire.

8.3 Flow Control

Enter topic text here.

8.3.1 Checkbox is checked

This action allows you to control the flow based upon if a checkbox is checked or not. Use the targeting button to specify what window to test for. Note that targeting checkboxes is not trivial and might require a bit of trial and error. For some applications this will not work.

Availability

Windows.

Girder 6 Change

Note that this is an asynchronous call if the backend is not running in your user session (for example running as a service or running on a different machine).

8.3.2 Explicit Return

Used to jump back to where a goto called this session.

8.3.3 File Exists

Depending if a file exists this action will jump to one of the specified nodes.

8.3.4 Goto Action

The Goto action will jump the processing of the event to the node (either a command or a macro) selected. Once processing is done in that location it returns to this action and proceeds normally.

8.3.5 Stop Processing

Stop processing will stop the event from being processed any further.

8.3.6 Window Exists

This action allows you to control the flow based upon if a window exists or not. Use the targeting button to specify what window to test for.

Availability

Windows.

Girder 6 Change

Note that this is an asynchronous call if the backend is not running in your user session (for example running as a service or running on a different machine).

8.3.7 Window is Foreground

This action allows you to control the flow based upon if a window is the foreground window or not. Use the targeting button to specify what window to test for.

Availability

Windows.

Girder 6 Change

Note that this is an asynchronous call if the backend is not running in your user session (for example running as a service or running on a different machine).

8.4 Girder Actions

8.4.1 Enable / Disable Node

Enables or disables the targeted node.

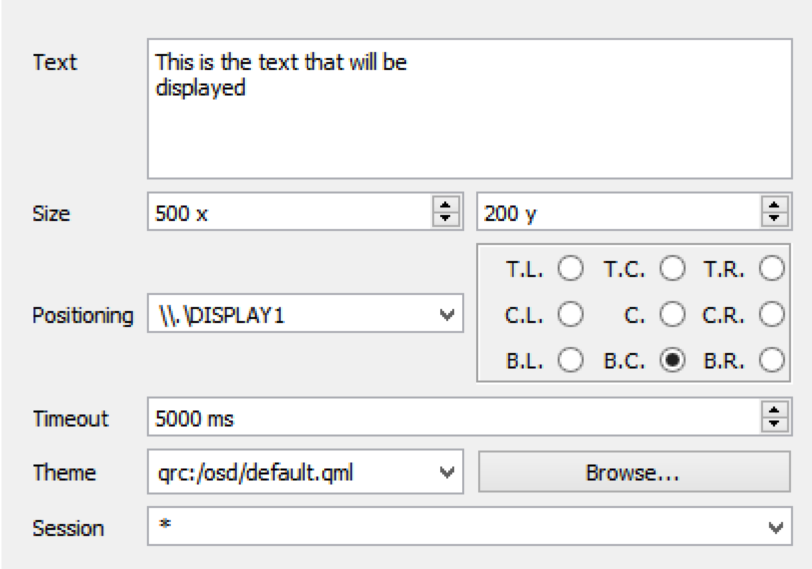
© 2015 Promixis, LLC

8.4.2 Enable / Disable Plugin

Enables or disables a plugin. Enter the number of the plugin. You can find this number on the Settings dialog on the plugin panel.

8.4.3 OSD

The OSD Action allows you to display messages on the screen. The OSD has a few configuration parameters and can also be customized very easily.



OSD Action

Parameter	Text
Text	Text to display on OSD. This can include [...] notation, which is parsed. For example [_VERSION] displays Lua 5.1
Size	The width and height in pixels for the OSD
Positioning	The first drop down specifies on which monitor to display. The 9 radio buttons take care of positioning, ranging from Top Left, Top Center, Top Right to Center to Bottom left, Bottom Center and Bottom Right.
Timeout	The number of milliseconds this window should be visible.
Theme	Girder comes with 4 predefined OSD's, default, information, warning and error. You can also create your own. Simply create a QML file and click the browse button to find that file.
Session	The session in which to display the OSD. (A session is a desktop instance of Girder)

© 2015 Promixis, LLC

Customizing the OSD

The OSD uses Qt QML files to create it's OSD, not unlike NetRemote. The OSD doesn't have all the features of NetRemote. As an example we include source for the built-in OSD here. The built-in OSD uses two files to create the OSD. First there is the file called "BaseOSDDialog.qml". This contains all the code needed to display the OSD in a few different colors and with icons.

```
import QtQuick 2.0
import QtGraphicalEffects 1.0

Rectangle {

    id: main
    property string icon: ""
    property alias showLogo: girderLogo.visible
    property alias caption: captionElement.text
    property alias theme: main.state

    anchors.fill: parent;
    border.width: 2
    radius: 10
    opacity: 1

    gradient: Gradient {

        GradientStop {
            id: gradStop1
            position: 0
            color: "#dd4072b1"
        }

        GradientStop {
            id: gradStop2
            position: 1
            color: "#dd7ca0cc"
        }
    }

    states: [
        State {
            name: "blue"
            PropertyChanges { target: gradStop1; color: "#dd4072b1" }
            PropertyChanges { target: gradStop2; color: "#dd7ca0cc" }
        },
        State {
            name: "red"
            PropertyChanges { target: gradStop1; color: "#ddb14040" }
            PropertyChanges { target: gradStop2; color: "#ddcc7c7c" }
        },
        State {
```

© 2015 Promixis, LLC

```
            name: "green"
            PropertyChanges { target: gradStop1; color: "#dd4cb140" }
            PropertyChanges { target: gradStop2; color: "#dd9bcc7c" }
        },
        State {
            name: "yellow"
            PropertyChanges { target: gradStop1; color: "#ddabb140" }
            PropertyChanges { target: gradStop2; color: "#ddc6cc7c" }
        }

    ]

    Image {
        source: "qrc:/osd/texture.png"
        opacity: .05
        fillMode: Image.Tile
        anchors.fill: parent;
    }

    MouseArea {
        anchors.fill: parent
        onClicked: {
            window.close();
        }
    }

    Image {
        id: girderLogo
        source: "qrc:/formIcons/icons/Girder 6.ico"
        anchors.bottom: parent.bottom
        anchors.right: parent.right
        anchors.margins: 10
        opacity: .50
    }

    Image {
        id: icon1
        source: icon
        anchors.left: parent.left
        anchors.verticalCenter: parent.verticalCenter
        anchors.margins: icon.length > 0 ? parent.width / 20 : 0
    }

    Text {
        id: captionElement
        color: "#ffffff"
        text: "none"
        anchors.right: parent.right
        anchors.left: icon1.right
        anchors.bottom: parent.bottom
        anchors.top: parent.top
        anchors.margins:  10
```

© 2015 Promixis, LLC

```
        wrapMode: Text.WordWrap
        font.pointSize: 24

        verticalAlignment: Text.AlignVCenter
        horizontalAlignment: Text.AlignHCenter

    }

    DropShadow {
        anchors.fill: captionElement
        horizontalOffset: 3
        verticalOffset: 3
        radius: 8.0
        samples: 16
        color: "#000000"
        source: captionElement
    }

}
```

Then there is the files called default.qml. Which as you can see uses the BaseOSDDialog above to actually render the text in the theme "blue".

```
import QtQuick 2.0

BaseOSDDialog {

    caption: osdText
    showLogo: true
    theme: "blue"

}
```

Girder exposes a few values to the QML that you may use:

Parameter	Text
osdText	The parsed text that should be shown.
girder.sendEvent(eventString, eventDevice, keyMod, payloads)	Function that allows you to send events into Girder. This can make interactive OSDs!
girder.triggerNode(fileId, nodeId)	Trigger a node on the tree directly.
girder.sendLog(level, source, message, fileId, nodeId)	Add an entry into the log.
window	Exposes the QQuickWindow.
osdWidth	The width of the window that this QML is displayed in. Just use anchors.file: parent in

© 2015 Promixis, LLC

	top rectangle.
osdHeight	The height of the window that this QML is displayed in.
osdTimeout	The timeout for the OSD. The timeout is handled internally.
osdFileId	The fileId of the node that created the OSD. Can be used in girder.sendLog.
osdNodeId	The nodeId of the node that created the OSD. Can be used in girder.sendLog.
osdEventString	If this node was triggered by an event this will be set to the event string.
osdEventDevice	If this node was triggered by an event this will be set to the event device number.

8.4.4 Reset All States

Resets the states of all nodes.

8.4.5 Reset Node State

Resets the node state.

8.5 Keyboard Actions

Keyboard actions come in two varieties. One is the targeted action and the other is non targeted. Both accept the same sequence of characters. The text to send has a special markup to send special keys like control, alt and windows keys. These are listed below.

Force Send Keys is a alternative way of sending key strokes. If keyboard sending doesn't work try checking this.

Keyboard sending markup

Modifier	Key
@	Alt
^	Shift
*****	Control
$	Windows Key

Special Keys

Label	Key

ALT	Alt Key
BACKSPACE	Backspace
DELETE	Delete
DOWN	Down arrow
LEFT	Left Arrow
RIGHT	Right Arrow
UP	Up Arrow
END	End
ENTER	Enter or Return key
ESCAPE	Escape
F1 - F12	Function keys
INSERT	Insert
PAGE_UP	Page Up
PAGE_DOWN	Page Down
SPACE	Space bar
TAB	Tab
PRINT_SCREEN	Print screen button
LWIN	Left Windows Key
RWIN	Right Windows Key
SCROLL_LOCK	Scroll Lock Key
NUM_LOCK	Num Lock Key
PAUSE	Pause
CAPS_LOCK	Caps Lock Key
NUMPAD0 - NUMPAD9	Numerical Key pad numbers
MULTIPLY	Multiply
ADDITION	Add
SUBTRACT	Subtract
DECIMAL	Decimal
DIVIDE	Divide
LEFT_CTRL	Left Control
RIGHT_CTRL	Right Control
LEFT_ALT	Left Alt
RIGHT_ALT	Right Alt
LEFT_SHIFT	Left Shift
RIGHT_SHIFT	Right Shift

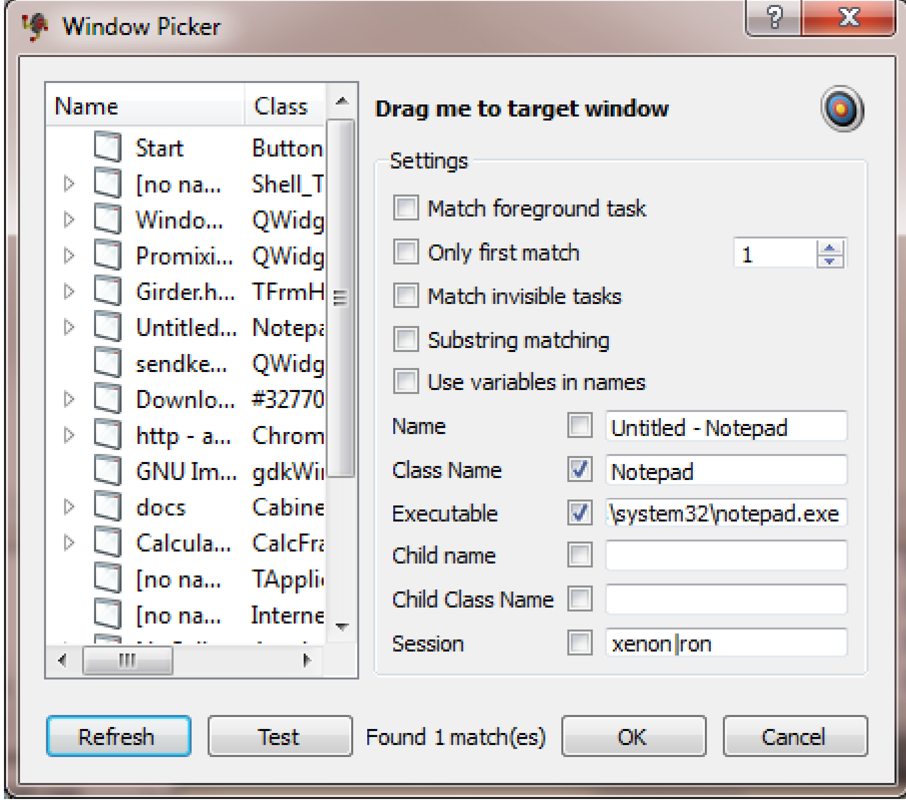
© 2015 Promixis, LLC

SLEEP	Sleep
NUMPADENTER	Enter on Numpad
BROWSER_BACK	Browser Back
BROWSER_FORWARD	Browser forward
BROWSER_FAVORITES	Browser favorites
BROWSER_HOME	Browser home
VOLUME_MUTE	Volume Mute
VOLUME_UP	Volume Up
VOLUME_DOWN	Volume Down
MEDIA_NEXT	Media Next
MEDIA_PREVIOUS	Media Previous
MEDIA_STOP	Media Stop
MEDIA_PLAY_PAUSE	Media Play/Pause
CTRL_UP	Control Up
CTRL_DOWN	Control Down
ALT_UP	Alt Up
ALT_DOWN	Alt Down
SHIFT_UP	Shift Up
SHIFT_DOWN	Shift Down
LWIN_UP	Left Windows Key Up
LWIN_DOWN	Left Windows Key Down
RWIN_UP	Right Windows Key Up
RWIN_DOWN	Right Windows Key Down

Example File Open - Ctrl-O

To send the Ctrl-o key sequence to Notepad. First target the main Notepad window. Targeting options will looks similar to this:

© 2015 Promixis, LLC



Targeting options

Then in the Text to Send box type *o

That's it. Now when you trigger this action Notepad will show the file open dialog.

Example Function Key - F5

If you wanted to insert the current date and time into notepad you can do that by pressing F5. We can simulate pressing F5 by type <F5> into the "Text to Send" box. You should see the date and time appear in Notepad.

8.6 Monitor Action

The monitor actions allow you to control the power state of the monitor.

8.7 Mouse Actions

There are two groups of mouse actions. The targeted group and the non-targeted group. For the targeted group you must specify the window of control on a window that the mouse action is supposed to work on. The mouse is not actually moved, it's all simulated. The non-targeted group however actually moves the mouse around on the screen.

© 2015 Promixis, LLC

8.7.1 Targeted Mouse Actions

These actions include the left, middle, right click, double click and "Move Relative to Window".

Parameters

These actions take 3 parameters. X and Y are relative to the targeted control. The third parameter is the key modifier. By checking Control Key or Shift, the action is sent with the keys flags set.

8.7.2 Un-Targeted Mouse Actions

This group contains all the actions that are needed to control the mouse cursor from events in Girder. For example in combination with a PIR-1 you could control your computers mouse using a Remote control.

Movement Actions

This include Move Up, Down, Left Right, the diagonal movements. There are 4 options to these.

Speedup Delay

This determines how long Girder waits before starting to use the "Large Step" movement increments.

Small Step

This is the amount in pixels that the mouse moves during the beginning of movement.

Large Step

This is the amount in pixels the mouse moves when the action continues to be triggered for longer than the "speed up delay".

Session

Since Girder's front-end and back-end are separate processes and multiple front-end can connect to one back-end it might be necessary to specify the use session to which these actions are sent. Typically this list is pre-filled with the currently attached front-end sessions.

Click Actions

The click actions include Left, Middle, Right click and double click. The only parameter for these is the session parameter which is described above.

© 2015 Promixis, LLC

Direct positioning

The last action here is the direct positioning. This allows you to position the mouse on the screen using X and Y coordinates in pixels.

8.8 Network Actions

8.8.1 Email Action

Sends an email.

8.8.2 HTTP Request

HTTP request allows you to do simple HTTP requests. The result is not stored. If you wish to use the result as well, look into the networking.get or networking.post lua actions.

Parameters

The URL is the fully qualified URL to post to. So include http://

Username and Password only need to be specified if the url has password protection.

POST/GET selection determines the method used to request the URL.

Body is the text that will be sent to the remote server.

Mime type is the mime type of the body. For example "application/json" for a JSON request or "application/xml" for xml and "application/x-www-form-urlencoded" for url encoded POST requests.

The On Success and On Failure node selectors determine what happens if the remote service returns "200" (=OK) or something else.

8.9 PIR Actions

The PIR-1 and PIR-4 actions are collected here. The PIR-1 and PIR-4 can both send IR codes. The PIR-1 also can learn and emulate key-presses without Girder help.

8.9.1 PIR-1 Actions

PIR-1 Send

The PIR-1 action sends IR codes. These are in the CCF format. These codes have a repeating and non-repeating part. To control the number of times the repeating part of the code is sent change the "Repeats" spinner. Note that if this is set to 0 and the CCF code only have a repeating part, nothing is sent!

© 2015 Promixis, LLC

PIR-1 can send IR codes out the front through it's built-in IR diodes or through the back using an plugin IR emitter.

The CCF code can either be found on the web on websites like www.remotecentral.com or learned directly with your PIR-1 by pressing the Learn button.

PIR-1 Set Keyboard

The PIR-1 has a four channel contact closure detection built in marked "Switch input". This allows Girder to respond to external events triggered by contact closure. For example an movement detector. Alternatively the PIR-1 can also send key presses directly. These however have to be configured.

Key	Code
a	4
b	5
c	6
d	7
e	8
f	9
g	10
h	11
i	12
j	13
k	14
l	15
m	16
n	17
o	18
p	19
q	20
r	21
s	22
t	23
u	24
v	25
w	26
x	27

© 2015 Promixis, LLC

y	28
z	29
1	30
2	31
3	32
4	33
5	34
6	35
7	36
8	37
9	38
0	39
Return	40
Escape	41
Backspace	42
Tab	43
Space	44
- _	45
= +	46
[{	47
] }	48
\ |	49
; :	51
' "	52
` ~	53
, <	54
. >	55
/ ?	56
Caps Lock	57
F1	58
F2	59
F3	60
F4	61
F5	62
F6	63

© 2015 Promixis, LLC

F7	64
F8	65
F9	66
F10	67
F11	68
F12	69
Print Screen	70
Scroll Lock	71
Break / Pause	72
Insert	73
Home	74
Page Up	75
Delete	76
End	77
Page Down	78
Right Arrow	79
Left Arrow	80
Down Arrow	81
Up Arrow	82
Num Lock	83
Keypad /	84
Keypad *	85
Keypad -	86
Keypad +	87
Keypad Enter	88
Keypad 1	89
Keypad 2	90
Keypad 3	91
Keypad 4	92
Keypad 5	93
Keypad 6	94
Keypad 7	95
Keypad 8	96
Keypad 9	97
Keypad 0	98

© 2015 Promixis, LLC

Keypad .	99
Keypad =	103
F13	104
F14	105
F15	106
F16	107
F17	108
F18	109
F19	110
F20	111
F21	112
F22	113
F23	114
F24	115
Volume Up	128
Volume Down	129
Left Control	224
Left Shift	225
Left Alt	226
Left Windows	227
Right Control	228
Right Shift	229
Right Alt	230
Right Windows	231

8.9.2 PIR-4 Actions

The PIR-4 action sends IR codes. These are in the CCF format. These codes have a repeating and non-repeating part. To control the number of times the repeating part of the code is sent change the "Repeats" spinner. Note that if this is set to 0 and the CCF code only have a repeating part, nothing is sent!

8.10 PRB-16 Actions

The PRB-16 is a 16 relay USB connected board. This allows you to automate anything with a dry contact input. For example door operators can be driven by this board. The PRB-16 has two actions. One action controls individual relays and their timeouts the other sets all relays at once.

Set Relay

This action allows you to set the individual relay state. Plus you can set the duration for that state. If you only want a relay to actuate for a short period enter a non-zero duration value.

Set Relays

This action allows you to set all relays in one quick action.

8.11 Scripting Action

This action creates a lua script action inside the action tree and allows for this script to run inside the normal tree flow. Please be aware that any code in this script is executed on the central processing thread and no new actions can trigger while it is running. Consider placing long running scripts on a separate thread.

Branching

Based on the return value from the script, either no return value, true or false. This action branches differently. If no value is returned no branching is performed. If true is returned tree processing will jump to the "True Node". If false is returned the branching will jump to the "False Node". After processing is complete on the "True Node" or "False Node" node event propagation will continue as normal.

Event Details

If your action needs access to the event details it can access these in the "event" global variable. This is a table with the following content.

Name	Type	Description
event	string	The event string
device	number	The device id
modifier	number	The modifier of the event.
payloads	table of strings	A table with all payloads for this event.

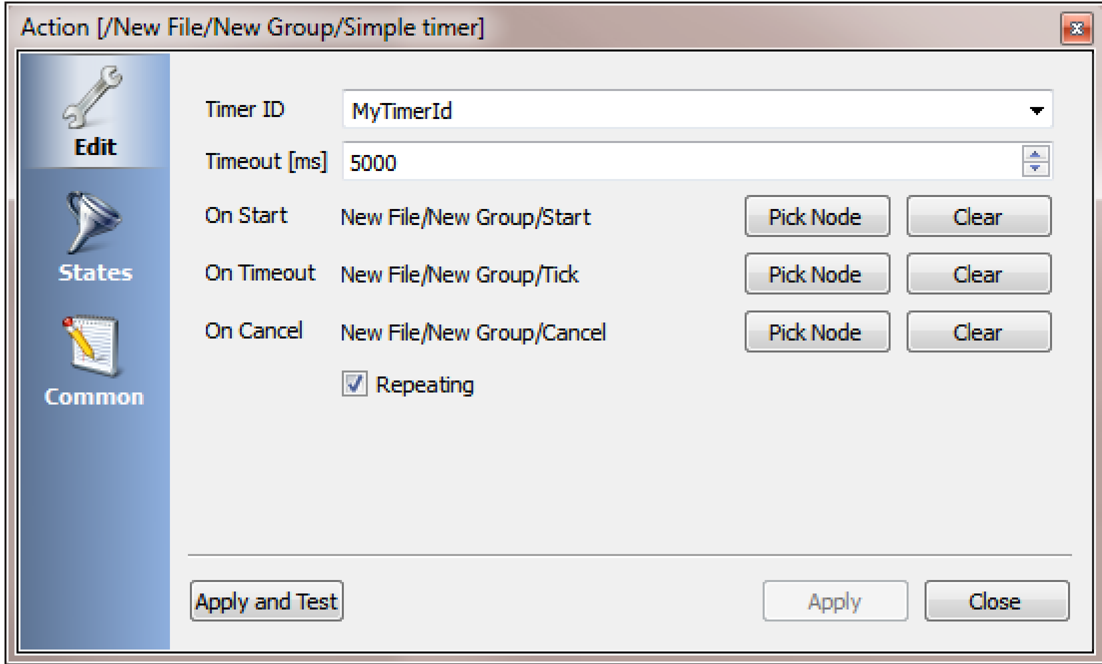
Lua

The scripting language used in Girder is called Lua. More specifically Lua 5.1. You can find the Lua 5.1 manual here or a good book here. Lua by nature is a very extensible language and Girder has added a lot of functionality. You can find the details of this in the scripting chapter of this manual.

8.12 Simple Timer

The Simple Timer action allows you to trigger nodes at a specific interval. If you need to have actions occur at specific times please use the Scheduler.

NOTE: The actions "Start" and "Stop" operate on the built in timers. The actions themselves are not the timer. So merely having them on the main tree does not activate the timers. You must trigger the actions for the timer to start or stop.



Simple Timer

Girder can use many timer at the same time. Give each timer a unique id. This can be a number or any string you like. In the screenshot above the timer id is "MyTimerId". The timeout is specified in milliseconds. This is one thousands of a second. So for a 5 second timeout enter 5000. Like above. The next 3 options determine which actions are triggered.

On Start is the action that is triggered when this timer is started.
On Timeout is the action that is triggered when this timer's time runs out
On Cancel is when this timer is stopped by the Stop Timer action.

8.13 Speech Action

The Speech action allows you to have your computer speak to you. You can enter plain text here. On Microsoft Windows this also accepts Microsofts "SAPI TTS XML". At the time of writing this documentation can be found at: http://msdn.microsoft.com/en-us/library/ms717077(v=vs.85).aspx

© 2015 Promixis, LLC

8.14 System Actions

8.14.1 Play Wav

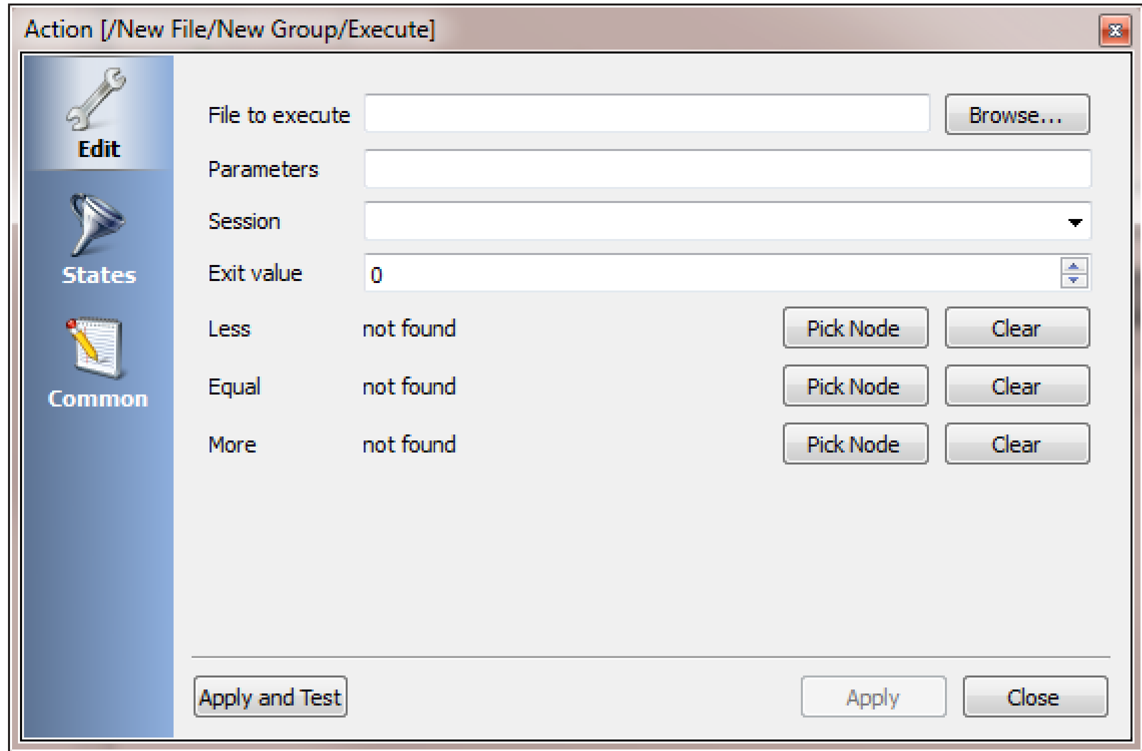
Plays a wav file on the selected session. Note that the wav file must exist on the file system of the front-end or back-end that is running the action. For example if you have a front-end connected on a different machine from the back-end make sure that the wav file is accessible to the front-end.

8.14.2 Shutdown, Reboot, Poweroff and Logoff

Use these actions to shutdown, reboot power off or log off the selected session.

8.14.3 Execute Action

File execute allows you to run applications either in the back-end session or in one of the attached front-ends. Once the external app completes the exit value can be compared to a value and a branch made based on it.



Execute file

Parameters

The file to execute is the full path and filename to execute. Parameters is the parameter string to pass to the executable. Session is the session to run in and "Exit Value" is the value used to compare against. If the exit value is less than the value defined here the

first node is triggered. If it's the same the second node is triggered and if it's more the 3rd node is triggered.

8.14.4 Eject / Load Media

Ejects or load the CD, DVD or BD drive. Simply enter the drive name and the session to operate on.

8.15 Twilio Actions

The Twilio action allows the sending of text messages and make automated phone calls. For this functionality to work you must have an account with Twilio. Once you are registered with Twilio please enter the sid and auth token on the plugin settings page. Without these entered these functions will not work. These functions can be useful to notify a user of a critical error condition or keep them up to date on the system status.

Send SMS

Send SMS allows you to send a SMS (also known as text message). Simply enter the recipient number and the sender number plus the text. Note that the from number must be registered with Twilio for this to work.

Call Phone

Calls the phone number and speaks the words in this box.

8.16 Twitter

Girder can send Tweets using this action. Simply enter the text you wish to send. The text may at maximum be 140 characters long. Before you can use the Twitter action you must Authorize Girder 6 to access your twitter account.

Authorzing Girder

To authorize Girder go to File->Settings->Plugins. Then select the Twitter plugin from the plugin list. Now first you'll need to click "Step 1: Request Authorization". This will return a URL for you to visit. This URL is on Twitter and will enable you to allow the plugin to access your account. Once you allow this on Twitter you will receive a PIN code from Twitter, copy and paste that in the the box right above the button named "Step 3: Verify Pin". Once you entered the pin click on that button. Done!

8.17 USB-UIRT

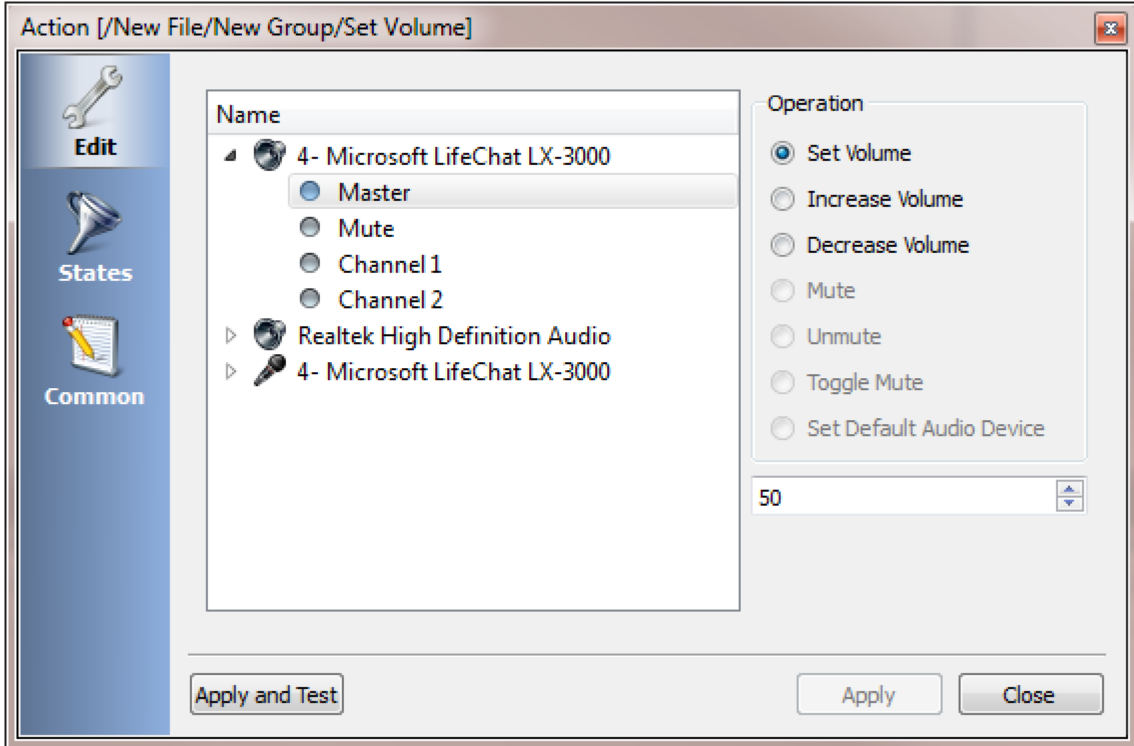
This action allows you to send and learn IR codes use the USB-UIRT.

8.18 Volume Action

Girder can change the levels and mute of the input and output devices on your Windows Vista and up computer. It can also set the default audio device on Windows 7 and

© 2015 Promixis, LLC

Windows Vista.



Set Volume

Volume channels accept a value between 0 and 100.

Increase Volume

Volume channels are changed by the value given. (0-100)

Decrease Volume

Volume Channels are decreased by the value given. (0-100)

Mute / Unmute / Toggle

Control the mute state of the audio channel.

Set default audio device

When selecting and audio device this allows you to set the default audio device.

8.19 Window Actions

The Window Actions contain the various actions that operate on application Windows.

9 Conditionals

Conditionals as the name implies make parts of the GML tree execute conditionally. Specifically the node that a conditional is attached to (thus it's parent) will not be triggered or traversed if the conditional prohibits it. How a conditional prohibits depends on the exact conditional. For example the script conditional will only allow nodes to be traversed if the return value from the script is true.

Conditional checking is done at all times. Even if you press "Test Action". So if your action is not working as expected, check if a conditional is blocking it.

9.1 Script

The script conditional runs a script every time the parent node is traversed. This means that in order to keep Girder fast make sure you do not make scripts that have a long run time. Return true to allow the parent node to traverse or trigger. Return null or false to block.

9.2 Device Manager

The Device Manager conditional allows you to make node traversal depend on the status and / or value of a control in the device manager.

9.3 Time of Day

The time of day conditional controls where the parent node is triggered depending on the time of day and day of week.

10 Speech

Girder includes support for both generating Speech as well as control of device manager control through speech. To use this functionality enable the Speech plugin on the settings dialog.

Speech output

To make Girder talk use the Speech/Text to Speech action.

Speech commands

Girder can respond to voice commands to control the level of controls defined in your Device Manager. To enable this feature make sure you have checked "Enable Voice Recognition" on the Speech plugin settings page. Then you'll need to tell Girder which of your devices can be controlled by speech. This is done by opening the Device Manager, finding the control you wish to use voice commands on and pressing the "Edit/Voice Settings..." button.

© 2015 Promixis, LLC

This will bring up a dialog that accepts a name and has a checkbox. Put the name you wish to speak for this control in there and check the box. By default Girder comes with an English grammar but you can modify this to your liking.

Grammar

Girder defines a limited grammar to allow for good recognition. The default grammar has the following structure.

For buttons and toggles the commands are **open**, **turn on**, **click**, **call**, or **close**, **turn off** followed by the name of the control (optional you can say please at the end.). Say you set a control's voice command name to "hallway light". You could now speak to Girder the following command: "turn on hallway light please"!

Range controls are set using the command **dim** followed by the name of the control, then the percentage you'd like to set to **0 percent, 10 percent etc..**

Controls that have a list of options are controlled by the command **set** followed by the control name followed by the name of the option you'd like to set. For example "set thermostat mode cool"

You can customize these things by copying the file "vr.xml" from the application directory to the Girder settings directory and modifying things.

NOTE: This feature is still under development and feedback is welcome.

11 Girder Command Line Options

Girder uses command line options to determine how it will start up. The accepted options are:

tray, gui, inproc, host, password, port and secure.

tray

Runs Girder in the tray on startup.

```
girder.exe --tray
```

gui

The gui parameter will open a QML gui on startup. Combin e this with the tray parameter to have Girder start hidden with a gui of choice displayed.

```
girder.exe --gui=c:\test.qml
```

© 2015 Promixis, LLC

inproc

Inproc starts the Girder backend inside the front-end process. This is the way Girder 5 used to run. This allows Girder to run using only one process.

```
girder.exe --inproc
```

password

Sets the password to connect to the backend communications.

```
girder.exe --password=secret
```

secure

Tells Girder to secure it's communications with SSL.

```
girder.exe --secure
```

12 NetRemote Command Line Options

NetRemote accepts a few command line options.

host

Specifies the hostname to connect to:

```
netremote --host=127.0.0.1
```

password

Specifies the password to use for the connection

```
netremote --password=girder
```

port

Specifies the port to connect to:

```
netremote --port=1024
```

© 2015 Promixis, LLC

secure

If specified the connection is attempted using SSL.

```
netremote --secure
```

gui

Specifies the file to open

```
netremote --gui="c:\program files\Promixis\Girder 6\ui\dynamicButtons
\dynamicButtons.qml"
```

clear

If you previously clicked "remember" in the NetRemote picker use this command to clear the settings.

```
netremote --clear
```

geometry

Specifies the size and position of the Window.

```
netremote --geometry=100x100x300x400
```

The parameter is Left X Top X Width X Height

13 IR Devices

The IR devices plugin presents an easy way to export IR sending to the Device Manager. For example if you have a PIR-1 or a PIO-1 and wish to create a virtual remote on a web page IR devices is the way to go.

You can create IR devices manually and contribute them to the Girder community or download already existing profiles for your own use.

Manually Adding IR Codes

All configuration for the IR devices plugin is done using the Device Manager interface. To open this click on "View->Device Manager".

...

© 2015 Promixis, LLC

14 Plugins

Documentation for the various plugins.

14.1 1-Wire

The 1-Wire functionality is currently only exposed to Lua you can find the documentation for that here.

14.2 CM11

To use the CM11 functionality enable the CM11 plugin on the settings dialog. Reopen the settings dialog and then add the CM11A by clicking "Add CM11".

14.3 Insteon

The Insteon Plugin is controlled from the Device Manager. Simply enable it on the settings dialog and go to the device manager to actually create a component for it.

14.4 IR Devices

IR Devices are also controlled from the Device Manager Window. Enable the plugin here then go to the Device Manager to actually load IR codes.

14.5 IRTrans

To use the IR trans functionality for enable the plugin on the settings dialog. Next go the device manager and add a component. Select "IRTrans IRServer" from the list and enter the IP address of the IR server or network connected IRTrans.

14.6 Nest

The Nest Plugin is controlled from the Device Manager Window. Enable the Plugin from the Settings Windows then configure it on the Device Manager Window.

14.7 RFXCom

To use the RFXCom functionality enable the RFXCom plugin on the settings dialog. Reopen the settings dialog and then add the RFX by clicking "Add RFX".

14.8 Scheduler

The scheduler documentation can be found here.

14.9 SimpleTransport

This plugin allows you to easily create drivers for serial or tcp controlled hardware.

15 User GUI

Girder has a built in configurable user GUI (or graphical user interface). This is designed the be used by the end-user. Someone who needs to operate the automation system but doesn't want to see Girder. Someone who wants to see pretty designs.

© 2015 Promixis, LLC

The User GUI uses the industry standard QML format. Looking at these files you'll notice they look a lot like Javascript with extra's. Which is exactly what they are. You can use the Qt Creator to create the QML files and load them in Girder.

Functions available in QML:

"girder" object:

SLOTS:
    void quit( );
    void sendEvent(const QString & eventString, int eventDevice, int keyMod, const QStringList & payloads = QStringList());
    void triggerNode(const QString & fileId, int nodeId);
    void sendLog(int level, const QString & source, const QString & message, const QString & fileId, int nodeId);
    void requestUpdateControlValue(quint64 controlId, const QString & value, const QString & sender);
PROPERTIES
    QString url
SIGNALS:
    void event(int eventDevice, QString eventString, int keyMod, QStringList payloads);
    void closeRequest();
    void urlChanged();

"kv" object:

SLOTS:
    virtual Promixis::IKVObject * value (const QString & key) = 0;
    virtual void preload(const QString & match) = 0;

Promixis::IKVObject

PROPERTIES:
        QString val
        bool isDefined
SIGNALS:
    void valChanged();
    void isDefinedChanged();

"mbFactory" object of type IMBFactory

IMBFactory

© 2015 Promixis, LLC

```
class IMBFactory : public QObject
{
    Q_OBJECT
public:
    explicit IMBFactory(QObject *parent = 0);

    virtual QList<IMediaBridge*> mediabridges() = 0;

    Q_PROPERTY(QList<IMediaBridge*> mediabridges READ mediabridges NOTIFY
mediaBridgesChanged )
signals:
    void mediaBridgesChanged();
public slots:
    virtual bool add( const QString & url ) = 0;
};
```

IMediaBridge

```
class IMediaBridge : public QObject
{
    Q_OBJECT

public:
    explicit IMediaBridge(QObject *parent = 0);

    virtual bool isConnected() const = 0;
    virtual QList<IMBZone*> zones() const = 0;
    virtual IMBItem * albumRoot() = 0;
    virtual IMBItem * genresRoot() = 0;
    virtual IMBItem * artistsRoot() = 0;
    virtual IMBItem * playingNow() = 0;
    virtual IMBItem * playlists() = 0;
    virtual QString name() const = 0;
    virtual IMBPage * page() = 0;
    virtual void setPassword( const QString & password ) = 0;
    virtual QString password() const = 0;
    virtual int zoneCount () const = 0;

    /* 0 = No image
     * 1 = Small
     * 2 = Medium
     * 3 = Large
     * 4 = Full
     */
    virtual void setDefaultImageSize(int size ) = 0;
    virtual int defaultImageSize() const = 0;
```

© 2015 Promixis, LLC

```
    Q_PROPERTY(QString name READ name NOTIFY nameChanged)
    Q_PROPERTY(bool isConnected READ isConnected NOTIFY
isConnectedChanged )
    Q_PROPERTY(QList<IMBZone*> zones READ zones NOTIFY zonesChanged )
    Q_PROPERTY(IMBItem * albumRoot READ albumRoot NOTIFY
albumRootChanged )
    Q_PROPERTY(IMBItem * genresRoot READ genresRoot NOTIFY
generesRootChanged )
    Q_PROPERTY(IMBItem * artistsRoot READ artistsRoot NOTIFY
artistsRootChanged )
    Q_PROPERTY(IMBItem * playingNow READ playingNow NOTIFY
playingNowChanged )
    Q_PROPERTY(IMBItem * playlists READ playlists NOTIFY
playlistsChanged )
    Q_PROPERTY(int defaultImageSize READ defaultImageSize WRITE
setDefaultImageSize NOTIFY defaultImageSizeChanged)
    Q_PROPERTY(IMBPage * page READ page NOTIFY pageChanged)
    Q_PROPERTY(QString password READ password WRITE setPassword NOTIFY
passwordChanged)
    Q_PROPERTY(int zoneCount READ zoneCount NOTIFY zoneCountChanged)

signals:
    void isConnectedChanged(bool isConnected);
    void zonesChanged( );
    void albumRootChanged();
    void generesRootChanged();
    void artistsRootChanged();
    void playingNowChanged();
    void playlistsChanged();
    void nameChanged();
    void pageChanged();
    void defaultImageSizeChanged();
    void passwordChanged();
    void zoneCountChanged();

public slots:
    virtual void connect() = 0;
    virtual void disconnect() = 0;
};
```

IMBZone

```
class IMBZone : public QObject
{
    Q_OBJECT
```

© 2015 Promixis, LLC

```
    QString m_Name;
public:

    explicit IMBZone(QObject *parent = 0);

    virtual void setName( const QString & name );
    virtual QString name() const;

    virtual void setVolume( int volume ) = 0;
    virtual int volume() const = 0;
    virtual void setMuted ( bool muted ) = 0;
    virtual bool muted() const = 0;

    virtual QString currentTrack ( ) const = 0;
    virtual QString currentAlbum ( ) const = 0;
    virtual QString currentArtist( ) const = 0;
    virtual QTime currentDuration( ) const = 0;
    virtual QTime currentPosition( ) const = 0;
    virtual bool isPlaying( ) const = 0;
    virtual QString currentCoverUrl ( ) const = 0;

    Q_PROPERTY(QString name READ name WRITE setName NOTIFY nameChanged)
    Q_PROPERTY(bool muted READ muted WRITE setMuted NOTIFY mutedChanged)
    Q_PROPERTY(int volume READ volume WRITE setVolume NOTIFY
volumeChanged)

    Q_PROPERTY(bool isPlaying READ isPlaying NOTIFY isPlayingChanged )
    Q_PROPERTY(QString currentTrack READ currentTrack NOTIFY
currentTrackChanged )
    Q_PROPERTY(QString currentArtist READ currentArtist NOTIFY
currentArtistChanged )
    Q_PROPERTY(QString currentAlbum READ currentAlbum NOTIFY
currentAlbumChanged )
    Q_PROPERTY(QTime currentDuration READ currentDuration NOTIFY
currentDurationChanged )
    Q_PROPERTY(QTime currentPosition READ currentPosition NOTIFY
currentPositionChanged )
    Q_PROPERTY(QString currentCoverUrl READ currentCoverUrl NOTIFY
currentCoverUrlChanged )


signals:
    void nameChanged( QString name );
    void mutedChanged( bool muted );
    void volumeChanged( int volume );
    void currentTrackChanged( QString track );
    void currentArtistChanged( QString artist);
    void currentAlbumChanged( QString album );
    void currentDurationChanged(QTime duration);
    void currentPositionChanged(QTime position);
```

© 2015 Promixis, LLC

```
    void isPlayingChanged(bool isPlaying);
    void currentCoverUrlChanged();

public slots:

    virtual void play() = 0;
    virtual void next() = 0;
    virtual void previous() = 0;
    virtual void pause() = 0;
    virtual void stop() = 0;

    virtual void play( IMBItem * item ) = 0;
    virtual void playAll( IMBPage * item ) = 0;
    virtual void add( IMBItem * item ) = 0;
    virtual void addAll( IMBPage * item ) = 0;
};
```

IMBPage

```
class IMBPage : public QObject
{
    Q_OBJECT
public:
    explicit IMBPage(QObject *parent = 0);

    virtual QList<IMBItem*> items() = 0;
    virtual int count() const = 0;
    virtual void setCount(int count) = 0;
    virtual int requestedCount() const = 0;
    virtual void setRequestedCount( int count ) = 0;
    virtual int startIndex() const = 0;
    virtual void setStartIndex(int index) = 0;
    virtual void setCoverImageSize ( int size ) = 0;
    virtual int coverImageSize( ) const = 0;
    virtual QString name () const = 0;
    virtual void setName( const QString & name ) = 0;
    virtual QString coverUrl ( ) const = 0;

    /* 0 = No image
     * 1 = Small
     * 2 = Medium
     * 3 = Large
     * 4 = Full
     */
    virtual void setImageSize ( int size ) = 0;
    virtual int imageSize( ) const =0;
```

```
    Q_PROPERTY(QList<IMBItem*> items READ items NOTIFY itemsChanged)
    Q_PROPERTY(int count READ count NOTIFY countChanged)
    Q_PROPERTY(int requestedCount READ requestedCount WRITE
setRequestedCount NOTIFY requestedCountChanged)
    Q_PROPERTY(int startIndex READ startIndex WRITE setStartIndex NOTIFY
startIndexChanged)
    Q_PROPERTY(int imageSize READ imageSize WRITE setImageSize NOTIFY
imageSizeChanged)
    Q_PROPERTY(int coverImageSize READ coverImageSize WRITE
setCoverImageSize NOTIFY coverImageSizeChanged)
    Q_PROPERTY(QString name READ name WRITE setName NOTIFY nameChanged )
    Q_PROPERTY(QString coverUrl READ coverUrl NOTIFY coverUrlChanged )

signals:

    void itemsChanged();
    void countChanged(int count);
    void requestedCountChanged(int count);
    void startIndexChanged(int index);
    void imageSizeChanged(int size);
    void nameChanged(QString name);
    void coverUrlChanged(QString cover);
    void coverImageSizeChanged(int size);


public slots:
    virtual void loadCover() = 0;
    virtual void loadItems( IMBItem * parent) = 0;
    virtual bool back() = 0;
};
```

IMBItem

```
class IMBItem : public QObject
{
    Q_OBJECT
    QString m_Name;
    bool m_IsFile;
public:
    explicit IMBItem(QObject *parent = 0);

    virtual void setName( const QString & name );
    virtual QString name() const;
    virtual bool isFile ( ) const;
    virtual void setIsFile( bool isFile );
    virtual QString coverUrl ( ) const = 0;
    virtual int subCount() const = 0;

    Q_PROPERTY(QString name READ name WRITE setName NOTIFY nameChanged)
    Q_PROPERTY(bool isFile READ isFile NOTIFY isFileChanged)
```

© 2015 Promixis, LLC

```
    Q_PROPERTY(QString coverUrl READ coverUrl NOTIFY coverUrlChanged)
    Q_PROPERTY(int subCount READ subCount NOTIFY subCountchanged)

signals:
    void nameChanged( QString name );
    void isFileChanged( bool isFile );
    void coverUrlChanged(QString url);
    void subCountchanged(int subCount);
public slots:

};
```

15.1 dmModel Object

The dmModel object exposes the Device Manager. It has the following properties and functions.

Functions

Return Value	Signature	Description
	load()	Loads the DM objects
Component	findComponent(quint64 componentId)	Finds the component with id.
Device	findDevice(quint64 deviceId)	Find the device with id.
Control	findControl(quint64 controlId)	Finds the control with id.
Location	findLocation(quint64 locationId)	Finds the location with id.
Device	findDeviceByName(string name)	Finds the first device with name.

Properties

Name	Type	Description
base	enumerations. Either Promixis.DeviceManagerModel.COMPONENT_BASED or Promixis.DeviceManagerModel.LOCATION_BASED	Determines how the model is filled
pluginIdFilter	integer	The ID of the plugin to filter for (default = 0, means get all components).

Events

Name	Type	Description
onLoaded	onLoaded()	This is called when the DM tree is fully loaded.

15.2 dmModelAdapter Object

The dmModelAdapter object is adapts the dmModel object to a QAbstractItemModel. This is a model that Qt and QML can use for filling lists.

15.3 kv

The kv namespace allows the user interface to get K-V values from Girder. These are bound values meaning that when the core K-V changes it's representation in the QML file changes as well.

Functions

Return Value	Signature	Description
KV Object	value(string name)	Gets a bound KV value.
	preload(string pattern)	Preloads the KV values of your choice (this allows loading of several KVs at once)

KV Object

Return Value	Signature	Description
string	val	Property that is bound to the kv that was requested.
boolean	isDefined	returns true if this value exists in the KV namespace.

Example

The following QML file will display a red or green rectangle depending on the value of the KV value "button.color".

```
import QtQuick 2.0

Rectangle {

    Rectangle {
        id: buttonToChange;

        width: 100
        height: 100

        anchors.centerIn: parent

        color: "#00ff00"

        state: kv.value("button.color").val
        states: [
            State {
                name: "red";
                PropertyChanges { target: buttonToChange; color:
"#ff0000" }
            }
        ]
    }
}
```

To trigger the changes use the following Lua code in Girder Script Actions on the Action tree.

```
local kvs = {}

kvs["button.color"] = "red";

kv.set(kvs, function( success )
        print("Set KV: " , success )
end)
```

```
local kvs = {}

kvs["button.color"] = "";

kv.set(kvs, function( success )
        print("Set KV: " , success )
end)
```

© 2015 Promixis, LLC

See Also

Lua KV

15.4 window

The window object has a few properties allowing you to control various aspects of the NetRemote window.

Properties

Property	Type	Description
url	string	URL of the QML file to load. For example file:///c:/my files/test.qml.
fullscreen	bool	make the window fullscreen or not.
pos	object with numbers x and y	Sets the position of the window. For example window.pos.x = 10;
size	object with number width and height	Sets the size of the window. For example window.size.width=100;
maximumWidth, maximumHeight	number	Sets the maximum width or height
minimumWidth, minimumHeight	number	Sets the minimum width or height

There are more properties and function that can be used. These can be found in the QWidget documentation of Qt here. Look for the "Properties" and the "Public Slots".

15.5 girder

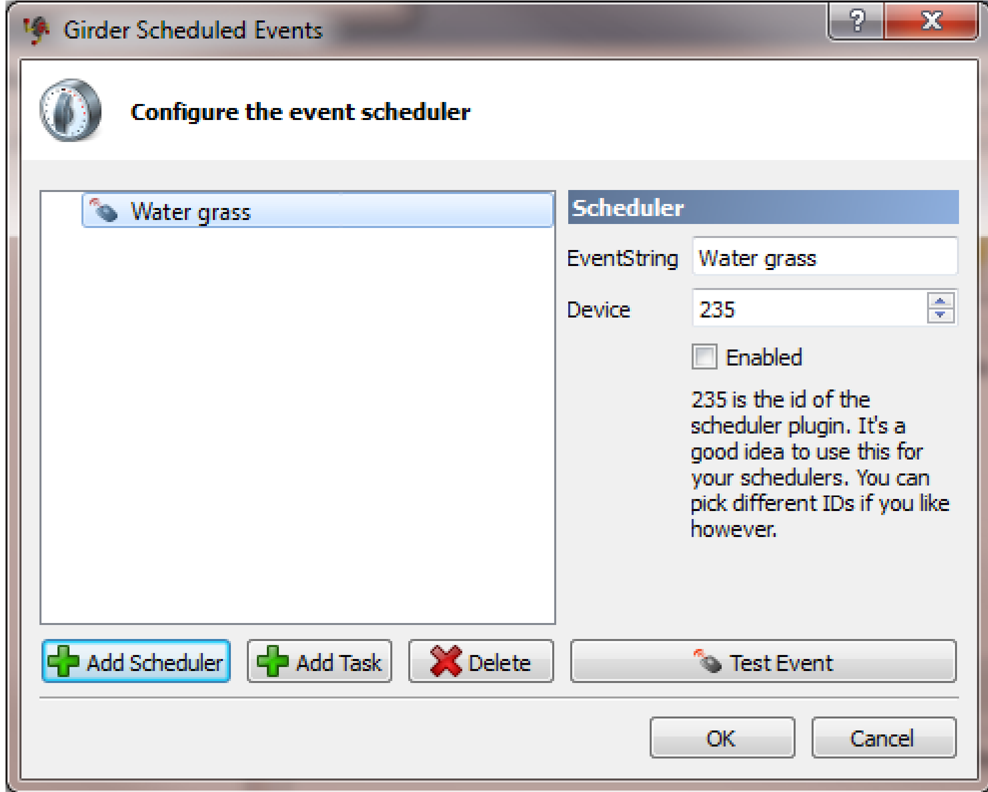
Properties

Property	Type	Description
batteryLevel	number	Percentage battery level

© 2015 Promixis, LLC

16 Scheduler

The scheduler allows you to trigger events inside Girder at set intervals. The intervals can be configured to be very flexible. We'll explain how to use the scheduler by the "Water Grass" example. In this example we want the grass to be water every other Monday at 9am.



Scheduler Main Window

The first thing to do is to open the scheduler. You can do so from the main menu (View->Scheduler). You should see a window like the one above without the "Water Grass" entry. To add this click on "Add Scheduler". This will add a scheduler named "New Scheduler" change this to "Water grass", make sure Device is set to 235. 235 is the ID of the scheduler plugin. You can pick other numbers but it's probably good to stick with 235 until you really know what you are doing.

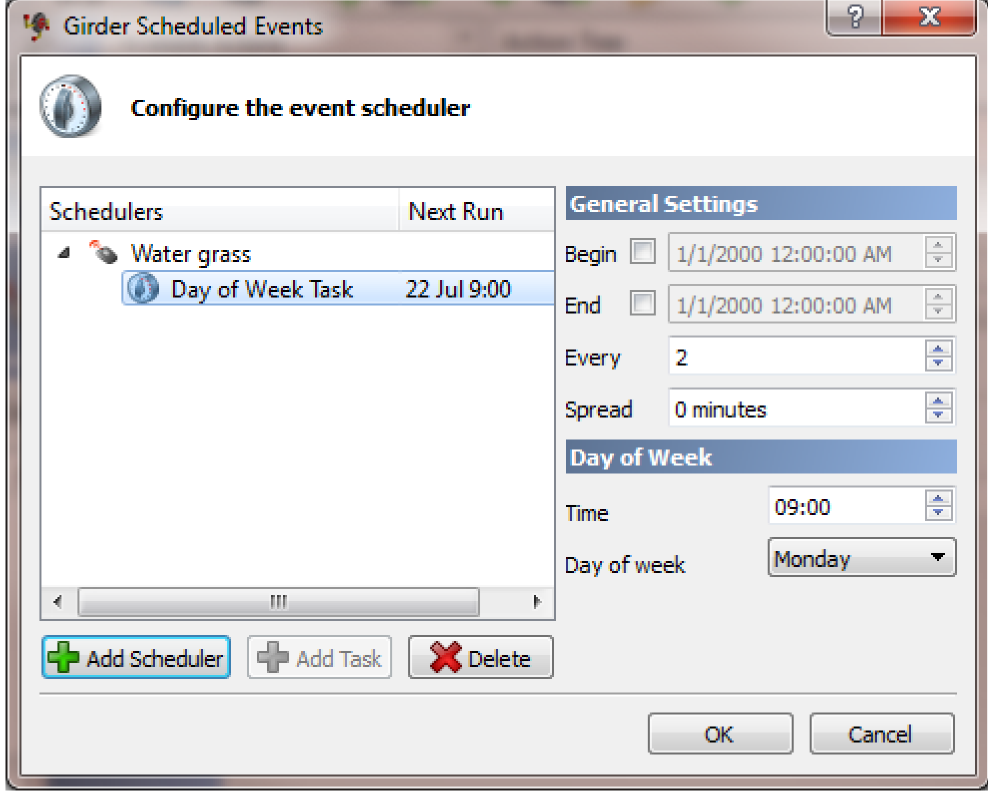
Don't forget to check the enabled checkbox or nothing will happen.

Next up we need to add the Tasks. Tasks are the parts of a scheduler that determine when it will fire. We want to fire every other Monday at 9am. If not selected click on "Water grass" again and click on "Add Task". A dialog will popup with a selection of different Tasks. Possible selections are:

- Day of Week
- Day of Month

- Sunset
- Sunrise
- Hour
- Minute
- Day

In our case we'll need to choose "Day of Week". The Window should look something like this now:



Scheduler with Tasks assigned

As you can see the Day of Week Task was added to the "Water grass" scheduler. To make it do every other Monday we set "Every" to 2. This means trigger this scheduler every 2nd Monday. Spread allows you to offset the trigger by a random number of minutes. For example entering 10 will change the event times by a random number of minutes between -10 and 10. This will create the illusion that a human is actually doing the triggering. Which could potentially be good for thwarting thieves into thinking someone is home since the action doesn't occur on exactly the same time every day. After you press OK the scheduler is active.

To make the scheduler actually do a task, take the event and attach it to an action on the action tree.

Availability

This is a Girder Pro and up feature.

© 2015 Promixis, LLC

17 Webserver

Girder has a built in web server that allows you to access Girder with a regular browser from any computer on the network. The web server can be password protected and SSL encrypted. If Girder is behind a home router (like a cable box) it might still be possible to reach Girder from the internet if you set the port forward in your cable modem or router.

The web pages allow for Lua script to be used server side. This was you can create your very own dynamic web pages!

The web server also supports websockets for fast two way communications. This can be very useful for device manager updates.

An example webpage would be:

```
<html>
<head>
	<title>Example Webpage</title>
</head>
<body>
	<h1>Example Girder webpage</h1>
	The hostname you used to get this page was: <% print( getHost() ) %
>
</body>
</html>
```

Availability

This is a Girder Pro and up feature.

17.1 Password protection

Each directory in the webserver directory can be password protected by placing a special file in the directory. The name of the file is "__access.txt" (two underscores at the beginning). In that file the first line is the "Realm" a piece of information that is displayed to the user when the password prompt comes up in the browser. Thereafter you can add the usernames and password separated by a colon. For example:

__access.txt

```
Promixis Controls

admin:verysecret
```

17.2 Add SSL Protection

Girder's webserver can encrypt the pages it sends out. To do this it needs a few files. Girder includes a tool that can generate those files for you, or you can generate them

yourself.

Certificate files

To properly serve SSL pages girder needs 3 files. The CA certificate (girderca.cert), the server certificate (girder.cert) and the server key (girder.key). These need to be placed in the Girder settings directory.

You can get these from a proper CA authority (and pay for that privilege) -or- generate them yourself. When you generate them yourself you will create self-signed certificates and you'll have to add your own certificate to your browsers trust chain. Since you control that certificate that should be safe.

Generating your own

You have two options. 1 Use Girder's certgen to generate the files (this can be done by pressing Generate SSL certificates... on the settings screen) or options 2: manually using openssl.exe. You can find a guide here: https://enterprise.github.com/help/articles/using-self-signed-ssl-certificates

Since SSL certificates are tied to the hostname you must give the hostname that you will be using to access the server. If you are using Girder on your local machine only you can enter "localhost". If you need to access it from elsewhere you'll need to make sure the name in the certificate matches the name you use to access the server. (for example http://192.168.1.23, would be hostname 192.168.1.23)

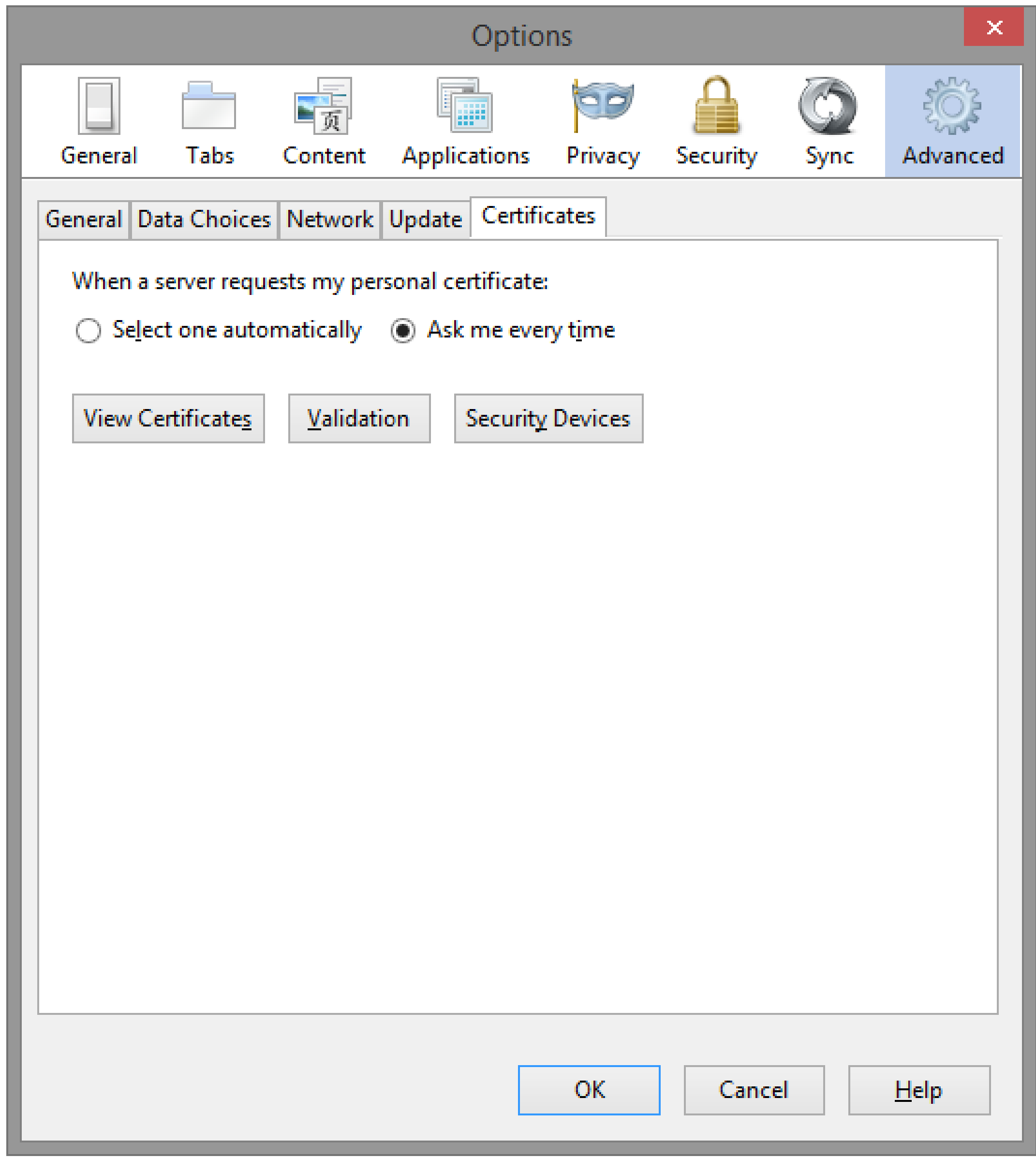
Once you generated the files exit Girder and restart Girder.

The browser

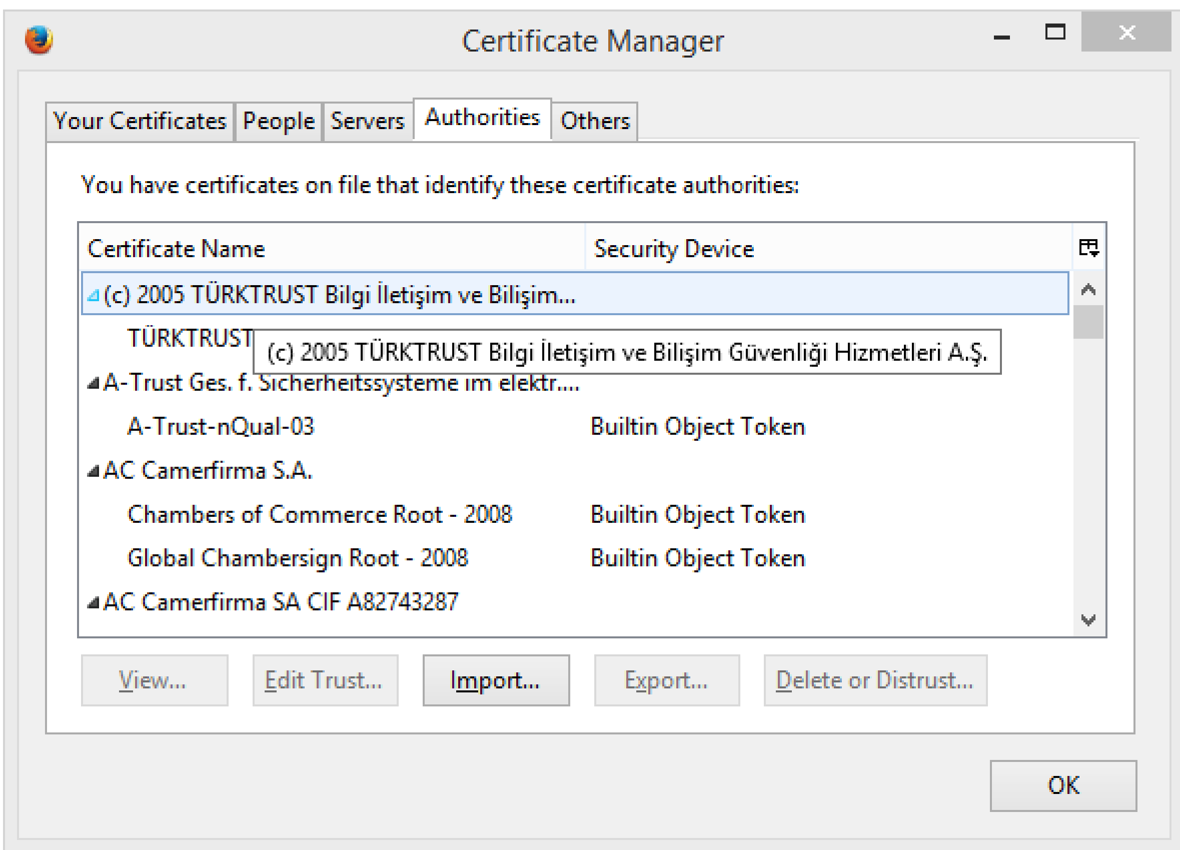
Of course since we self-signed the certificates the browser is not going to be very impressed with this. It doesn't know about our new Certificate Authority. So let's add our certificate in. In firefox you can do this as follows:

Open the Options dialog , click on Advanced, then press the Certificates tab:

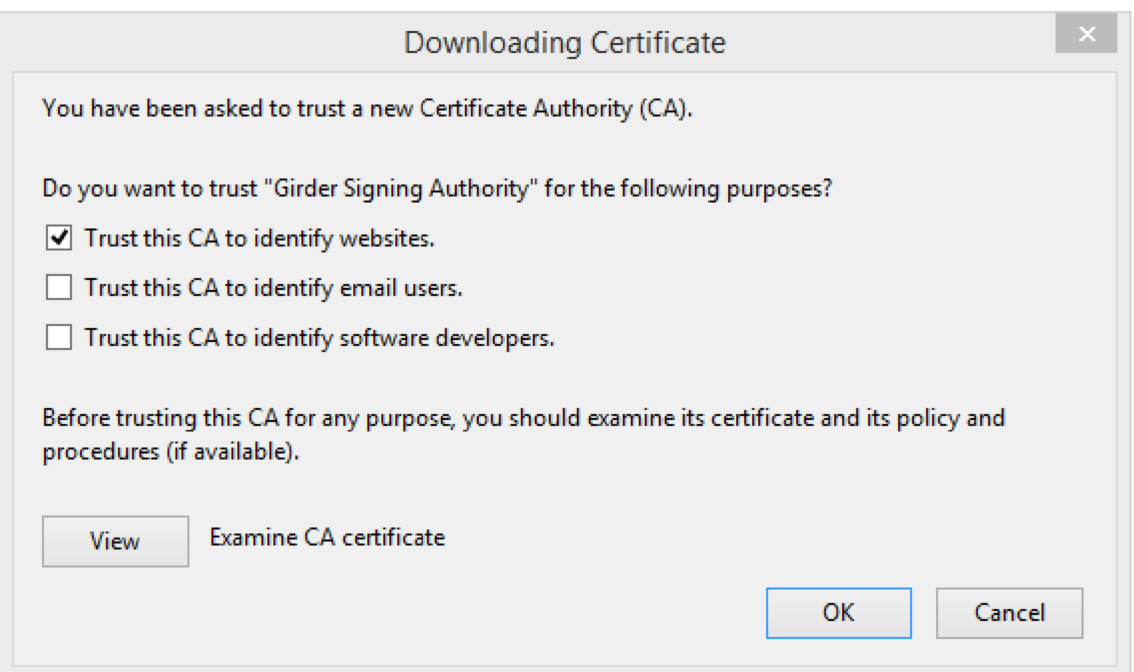
© 2015 Promixis, LLC



Click on View Certificates, go to the authorities tab and press import



In import browse to your Girder settings directory (Typically c:\users\<YOURNAME>\appData\local\Promixis\Girder 6\ ) and find girderca.cert. Import that one.



Make sure you check "Trust this CA to identify websites", as that is the whole point of this exercise. That's it. Now load your page and it should come up as SSL secured.

17.3 addModRewrite

addModRewrite allows you to create URL that do not exists in your httpd folder and translate them to existing files. This for example is very handy if you wish to create a Restful server in Girder.

For example say you'd like to accept a url in this form:

http://localhost/devices/{deviceId}/action/{actionName}/{actionParam}

You could create a .lhtml that takes 3 cgi parameters:

```
<html>
<body>
<%
      local cgi = getCGI()
      print(cgi['deviceId'] .. "<BR>")
      print(cgi['actionName'] .. "<BR>")
      print(cgi['actionParam'] .. "<BR>")
%>
</body>
</html>
```

let's name that modrewrite.lhtml and place it in the httpd directory. In a scripting action register the rewrite. Probably a good idea to attached a GirderStarted event to this scripting action.

```
webserver.addModRewrite("http://([a-zA-Z0-9.]+)/devices/(\\d+)/action/([^/]+)/(.*)", "http://$1/modrewrite.lhtml?deviceId=$2&actionName=$3&actionParam=$4")
```

Now next time you pull up this page you should see the parameters displayed.

Note: webserver.addModRewrite should be called before the HTTP request is made.

17.4 clearModRewrite

Removes all the modRewrite rules.

17.5 escape

escapes <, > and " into their HTML entity equivalents.

Definition

© 2015 Promixis, LLC

```
escapedString = escape( string )
```

Example

```
local escaped = escape( "hello <b> this is not bold </b>" )
print(escape)
```

Availability

Lua context during page load.

17.6 getBody

Returns the request body. For example a POST request will have that.

Definition

```
data = getBody()
```

Example

```
local body = getBody()

print( body )
```

Alternative name

```
webserver:GetBody()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

© 2015 Promixis, LLC

17.7 getCGI

Gets the CGI values passed to the server.

Definition

```
cgis = getCGI()
```

cgis is a table of key value pairs representing the POST or GET variables.

Example

```
local cgis = getCGI()

for key,value in pairs(cgis) do
      print(key, value)
end
```

For the url http://localhost/index.lhtml?a=1&b=hello this would print

```
a 1
b hello
```

Alternative name

```
filename = webserver:GetCGI()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.8 getCookies

Returns the cookies that were present in the HTTP request.

Definition

```
table = getCookies()
```

© 2015 Promixis, LLC

Example

```
local t = getCookies()

for name, value in pairs(t) do
      print( name, value )
end
```

Alternative name

```
webserver:GetCookies()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.9 getFilename

Returns the fully qualified filename of the script.

Definition

```
filename = getFilename()
```

filename is a string containing the request host for example c:/program files/Promixis/Girder 6/httpd/index.lhtml

Alternative name

```
filename = webserver:GetFilename()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

© 2015 Promixis, LLC

17.10 getHeader

Returns the header with the name that was passed as the function argument.

Definition

```
headerValue = getHeader( headerName )
```

Name	Type	Description
headerValue	string	Value of the header or nil if it didn't exist.
headerName	string	Header name for example "Location".

Alternative name

```
headerValue = webserver:GetHeader( headerName )
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.11 getHeaders

Returns all the request headers for this page.

Definition

```
headers = getHeaders()
```

headers is a table of key value pairs representing the request headers.

Example

```
local headers = getHeaders()

for key,value in pairs(headers) do
      print(key, value)
end
```

© 2015 Promixis, LLC

This would print for example

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Host: localhost
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.16 (KHTML, like Gecko)
Connection: keep-alive
Accept-Language: en-US,en;q=0.8
Accept-Encoding: gzip,deflate,sdch
```

Alternative name

```
headers = webserver:GetHeaders()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.12 getHost

Returns the host name from the request url.

Definition

```
host = getHost()
```

host is a string containing the request host for example localhost if the URL was http://localhost/chat.lhtml

Alternative name

```
host = webserver:GetHost()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

© 2015 Promixis, LLC

17.13 getMethod

Returns the request method. For example "GET" or "POST"

Definition

```
data = getMethod()
```

Example

```
local method = getMethod()

print( body )
```

Alternative name

```
webserver:GetMethod()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.14 getPort

returns the port number on which the server is running.

Definition

```
portNumber = getPort(  )
```

Example

```
local port = getPort( )
print(port)
```

© 2015 Promixis, LLC

Availability

Lua context during page load.

17.15 getPort

Returns the host name from the request url.

Definition

```
port = getPort()
```

host is a number containing the request host port number.

Availability

Lua context during page load.

17.16 getURL

Returns the url for the current request.

Definition

```
url = getURL()
```

url is a string containing the request URL for example "http://localhost/index.lhtml"

Alternative name

```
url = webserver:GetURL()
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

© 2015 Promixis, LLC

17.17 print

prints the arguments to this function to the web page.

Definition

```
print( value,... )
```

value can be any object that can be translated to a string. This is pretty much anything in Lua. Just like with the normal print function multiple values can be passed separated by commas. In the output these values are separated by tabs. Note that in HTML tabs are treated as a single space.

Example

```
print("Hey awesome webapge, with lua version: ", _VERSION)
```

This would print

```
Hey awesome webapge, with lua version: Lua 5.1
```

Alternative name

```
webserver:print( value, ... )
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.18 registerWebSocketServer

This function registers a websocket server on the webserver. Websockets are a way for 2 way communication from the webserver to the browser. This allows the webpage to be dynamically updated without having to poll the webserver for changes. This is because the webserver can now initiate a transfer to the browser.

This websocket implementation supports two different websocket wireformats:

- http://tools.ietf.org/html/draft-hixie-thewebsocketprotocol-76
- http://tools.ietf.org/html/rfc6455

This should cover most of the browsers on the market today.

© 2015 Promixis, LLC

Definition

```
webserver.registerWebSocketServer(hostname, path, websocketFactory)
```

Name	Type	Description
hostname	string	The name of the host this websocket should listen on or "*" for all hosts.
path	string	The path of the websocket on the server starting with a slash. For example "/test"
websocketFactory	function	Function that is called when a websocket is requested. This function is passed the websocket communication object. This function must return a websocket lua object.

Websocket Lua Object

Return value	Signature	Description
string	accept(self, secure, url, headers, peerAddress, peerPort)	This function is called when a new websocket is requested. Note that you must return the protocol this socket speaks if you accept the connection or nil to reject
	connected()	This function is called when the browser also accepts the connection and the connection is ready for data.
	disconnected()	This function is called when the websocket connection was closed.
	received(string)	This function is called when data comes in.

© 2015 Promixis, LLC

Websocket Communication Object

Return value	Signature	Description
	sendText (string)	sends the string to the remote end
	close()	closes the connection

Example

This code should be run when Girder starts (or after a lua reset.). You can place it in an action with the appropriate events attached. If all is well you should see "websocket registered" in the lua console.

```
ExampleWebSocket = require("examples.websocket")

print(webserver.registerWebSocketServer("*", "/test", function( ws )
    print("starting!")

    exampleWebSocket = ExampleWebSocket.new(ws)
    return exampleWebSocket

end))

print("websocket registered")
```

The examples/websocket.lua file contains this:

```
local base = require('Class')
local timer = require('timer')

local print = print

module(...)

base:subclass( _M)

function accept ( self, secure, url, headers, host, port )
      print(secure, url)
      return true, "test" -- you must return a matching protocol here!
end

function onTimeout(self)
      self.ws:sendText("Ping")
end
```

© 2015 Promixis, LLC

```
function connected(self)
	print("Websocket connected")
	self.timer = timer.new(1000, function()
		onTimeout(self)
	end)
	self.timer:start();
end

function disconnected(self)
	print("Websocket disconnected")
	if self.timer then
		self.timer:stop()
		self.timer:deinit()
		self.timer = nil
	end
end

function received(self, text)
	print("Received: ", text)
	self.ws:sendText("You sent me: " .. text)
end

function deinit(self)
	self.ws = nil
end

function init ( self, ws )
	base.init(self)
	self.ws = ws
end
```

The HTML file that goes with this looks like this:

```
<html>
<head>
	<title>
	Chat demo
	</title>
<style type="text/css">
body {
	background-color: #a0a0a0;
	background-image: url(Girder3D-436.png);
}

#connectedDiv {
	display:none;
}

#frame {
```

© 2015 Promixis, LLC

```
        width:800px;
        background-color:white;
        border: 3px solid black;
        padding: 20px;

}

</style>
<script type="text/javascript" src="https://ajax.googleapis.com/ajax/
libs/jquery/1.8.1/jquery.min.js"></script>
<script type="text/javascript">

        var chatName = "";
        var timer = 0;
        var status;
        var txt;
        var line;
        var broadcastSocket;
        var secure = "wss://<% print(getHost() .. ":" .. getPort()) %>/
test";
        var plain = "ws://<% print(getHost() .. ":" .. getPort()) %>/test";
        var useSecure = false

        function startChat(url) {


                broadcastSocket = new WebSocket(url, "test");

                broadcastSocket.onopen = function (event) {
                        $("#status").text( "Status: connected");
                        broadcastSocket.send($("#nameEdit").val() + " has
joined.");
                };
                broadcastSocket.onclose = function (event) {
                        $("#status").text( "Status: disconnected");
                        setTimeout( function () {
                                $("#status").text( "Status: connecting...");
                                startChat( useSecure ? secure : plain );
                        }, 5000);
                };
                broadcastSocket.onmessage = function(event) {
                        $("#status").text( "Status: connected");
                        $("#text").text( $("#text").text() + "\n" +
event.data);
                };




        }
```

© 2015 Promixis, LLC

```
        $(document).ready( function () {

                status = $("#status");
                txt = $("#text");
                line = $("#line");

                $("#nameSubmitPlain").click( function () {

                        if ( $("#nameEdit").val() === "" ) {
                                alert("Please enter a name");
                                return;
                        }
                        chatName = $("#nameEdit").val();
                        useSecure = false;
                        startChat( useSecure ? secure : plain );

                        $("#connectedDiv").show();
                        $("#nameDiv").hide();

                });

                $("#nameSubmitSecure").click( function () {

                        if ( $("#nameEdit").val() === "" ) {
                                alert("Please enter a name");
                                return;
                        }
                        chatName = $("#nameEdit").val();
                        useSecure = true;
                        startChat( useSecure ? secure : plain );

                        $("#connectedDiv").show();
                        $("#nameDiv").hide();

                });


                $("#sendButton").click( function () {
                        broadcastSocket.send( chatName + " | " + line.val() );
                        line.val("");
                });

        });

</script>

</head>
<body>

<center>
```

© 2015 Promixis, LLC

```
<div id="frame">
<h1>Chat Demo</h1>

<div id="nameDiv">
Please enter your name
<input type="text" id="nameEdit"><button id="nameSubmitPlain">Connect
Plain</button>
<button id="nameSubmitSecure">Connect Secure</button>
</div>

<div id="connectedDiv">

<p id="status">Status: Unknown<p>
<textarea id="text" cols="80" rows="20"></textarea><br>
<input type="text" id="line"> <button id="sendButton">Send</button>
</div>
</div>
<center>

</body>
</html>
```

The file above is available "wsexample.lhtml" after making sure the websocket lua code was loaded hit http://localhost/wsexample.lhtml. This then should print "Websocket connected" on the lua console. Now you can send text to the browser with `exampleWebSocket.ws:SendText("Hello")`

17.19 setCookie

Sets the cookies Header in the response.

Definition

```
setCookie( name, value, [expirationSeconds = 0], [path = /], [domain =
""], [secureOnly] )
```

Example

```
setCookie( "cool", "Eat This Cookie")
```

Alternative name

© 2015 Promixis, LLC

```
webserver:SetCookies(...)
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.20 setHeaderEx

Adds a HTTP header.

Definition

```
setHeaderEx( field, value, replace )
```

Example

```
setHeaderEx( "X-Server", "Girder 6", true )
```

Alternative name

```
webserver:SetHeaderEx( field, value, replace )
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

17.21 setStatus

Sets the HTTP response status. Default is code 200, "OK"

Definition

```
setStatus( code, codeString )
```

© 2015 Promixis, LLC

Example

```
setStatus( 404, "Not Found!" )
```

Alternative name

```
webserver:SetStatus( code, codeString )
```

Provided for Girder 5 compatibility.

Availability

Lua context during page load.

18 Scripting

Girder has two built-in scripting languages. One for back-end processing called Lua and Javascript for interface building. Some of the objects and methods documented here are available to both languages.

The detailed Lua 5.1 manual can be found here: http://promixis.com/lua/contents.html

Often these libraries need to be included into the script before they will work. For example for the Raspberry Pi library you must do the following:

```
local raspi = require("raspi")
```

So if you are getting errors with the examples, make sure to include the library you are dealing with.

18.1 bit

Loading the BitOp Module

The suggested way to use the BitOp module is to add the following to the start of every Lua file that needs one of its functions:

```
local bit = require("bit")
```

This makes the dependency explicit, limits the scope to the current file and provides faster access to the bit.* functions, too. It's good programming practice not to rely on the global variable bit being set (assuming some other part of your application has already loaded the module). The require function ensures the module is only loaded once, in any case.

Defining Shortcuts

It's a common (but not a required) practice to cache often used module functions in locals. This serves as a shortcut to save some typing and also speeds up resolving them (only relevant if called hundreds of thousands of times).

```
local bnot = bit.bnot
local band, bor, bxor = bit.band, bit.bor, bit.bxor
local lshift, rshift, rol = bit.lshift, bit.rshift, bit.rol
-- etc...

-- Example use of the shortcuts:
local function tr_i(a, b, c, d, x, s)
  return rol(bxor(c, bor(b, bnot(d))) + a + x, s) + b
end
```

Remember that and, or and not are reserved keywords in Lua. They cannot be used for variable names or literal field names. That's why the corresponding bitwise functions have been named band, bor, and bnot (and bxor for consistency).
While we are at it: a common pitfall is to use bit as the name of a local temporary variable — well, don't! :-)

About the Examples

The examples below show small Lua one-liners. Their expected output is shown after -->. This is interpreted as a comment marker by Lua so you can cut & paste the whole line to a Lua prompt and experiment with it.
Note that all bit operations return signed 32 bit numbers. And these print as signed decimal numbers by default.
For clarity the examples assume the definition of a helper function printx(). This prints its argument as an unsigned 32 bit hexadecimal number on all platforms:

```
function printx(x)
  print("0x"..bit.tohex(x))
end
```

bit.tobit

```
y = bit.tobit(x)
```

Normalizes a number to the numeric range for bit operations and returns it. This function is usually not needed since all bit operations already normalize all of their input arguments.

```
print(0xffffffff)                 --> 4294967295 (*)
print(bit.tobit(0xffffffff))      --> -1
printx(bit.tobit(0xffffffff))     --> 0xffffffff
print(bit.tobit(0xffffffff + 1)) --> 0
print(bit.tobit(2^40 + 1234))     --> 1234
```

bit.tohex

```
y = bit.tohex(x [,n])
```

Converts its first argument to a hex string. The number of hex digits is given by the absolute value of the optional second argument. Positive numbers between 1 and 8 generate lowercase hex digits. Negative numbers generate uppercase hex digits. Only the least-significant 4*|n| bits are used. The default is to generate 8 lowercase hex digits.

```
print(bit.tohex(1))               --> 00000001
print(bit.tohex(-1))              --> ffffffff
print(bit.tohex(0xffffffff))      --> ffffffff
print(bit.tohex(-1, -8))          --> FFFFFFFF
print(bit.tohex(0x21, 4))         --> 0021
print(bit.tohex(0x87654321, 4))   --> 4321
```

bit.bnot

```
y = bit.bnot(x)
```

Returns the bitwise not of its argument.

```
print(bit.bnot(0))             --> -1
printx(bit.bnot(0))            --> 0xffffffff
print(bit.bnot(-1))            --> 0
print(bit.bnot(0xffffffff))    --> 0
```

© 2015 Promixis, LLC

```
printx(bit.bnot(0x12345678))  --> 0xedcba987
```

bit.bor, bit.band, bitxor

```
y = bit.bor(x1 [,x2...])
y = bit.band(x1 [,x2...])
y = bit.bxor(x1 [,x2...])
```

Returns either the bitwise or, bitwise and, or bitwise xor of all of its arguments. Note that more than two arguments are allowed.

```
print(bit.bor(1, 2, 4, 8))                --> 15
printx(bit.band(0x12345678, 0xff))        --> 0x00000078
printx(bit.bxor(0xa5a5f0f0, 0xaa55ff00))  --> 0x0ff00ff0
```

bit.lshift, bit.rshift, bit.arshift

```
y = bit.lshift(x, n)
y = bit.rshift(x, n)
y = bit.arshift(x, n)
```

Returns either the bitwise logical left-shift, bitwise logical right-shift, or bitwise arithmetic right-shift of its first argument by the number of bits given by the second argument.
Logical shifts treat the first argument as an unsigned number and shift in 0-bits.
Arithmetic right-shift treats the most-significant bit as a sign bit and replicates it.
Only the lower 5 bits of the shift count are used (reduces to the range [0..31]).

```
print(bit.lshift(1, 0))              --> 1
print(bit.lshift(1, 8))              --> 256
print(bit.lshift(1, 40))             --> 256
print(bit.rshift(256, 8))            --> 1
print(bit.rshift(-256, 8))           --> 16777215
print(bit.arshift(256, 8))           --> 1
print(bit.arshift(-256, 8))          --> -1
printx(bit.lshift(0x87654321, 12))   --> 0x54321000
printx(bit.rshift(0x87654321, 12))   --> 0x00087654
printx(bit.arshift(0x87654321, 12))  --> 0xfff87654
```

© 2015 Promixis, LLC

bit.rol, bit.ror

```
y = bit.rol(x, n)
y = bit.ror(x, n)
```

Returns either the bitwise left rotation, or bitwise right rotation of its first argument by the number of bits given by the second argument. Bits shifted out on one side are shifted back in on the other side.
Only the lower 5 bits of the rotate count are used (reduces to the range [0..31]).

```
printx(bit.rol(0x12345678, 12))   --> 0x45678123
printx(bit.ror(0x12345678, 12))   --> 0x67812345
```

```
printx(bit.rol(0x12345678, 12))   --> 0x45678123
printx(bit.ror(0x12345678, 12))   --> 0x67812345
```

bit.bswap

```
y = bit.bswap(x)
```

Swaps the bytes of its argument and returns it. This can be used to convert little-endian 32 bit numbers to big-endian 32 bit numbers or vice versa.

```
printx(bit.bswap(0x12345678)) --> 0x78563412
printx(bit.bswap(0x78563412)) --> 0x12345678
```

Example Program

This is an implementation of the (naïve) Sieve of Eratosthenes algorithm. It counts the number of primes up to some maximum number.
A Lua table is used to hold a bit-vector. Every array index has 32 bits of the vector. Bitwise operations are used to access and modify them. Note that the shift counts don't need to be masked since this is already done by the BitOp shift and rotate functions.

```
local bit = require("bit")
local band, bxor = bit.band, bit.bxor
local rshift, rol = bit.rshift, bit.rol
```

© 2015 Promixis, LLC

```
local m = tonumber(arg and arg[1]) or 100000
if m < 2 then m = 2 end
local count = 0
local p = {}

for i=0,(m+31)/32 do p[i] = -1 end

for i=2,m do
  if band(rshift(p[rshift(i, 5)], i), 1) ~= 0 then
    count = count + 1
    for j=i+i,m,i do
      local jx = rshift(j, 5)
      p[jx] = band(p[jx], rol(-2, j))
    end
  end
end

io.write(string.format("Found %d primes up to %d\n", count, m))
```

Caveats

Signed Results

Returning signed numbers from bitwise operations may be surprising to programmers coming from other programming languages which have both signed and unsigned types. But as long as you treat the results of bitwise operations uniformly everywhere, this shouldn't cause any problems.
Preferrably format results with bit.tohex if you want a reliable unsigned string representation. Avoid the "%x" or "%u" formats for string.format. They fail on some architectures for negative numbers and can return more than 8 hex digits on others.
You may also want to avoid the default number to string coercion, since this is a signed conversion. The coercion is used for string concatenation and all standard library functions which accept string arguments (such as print() or io.write()).

Conditionals

If you're transcribing some code from C/C++, watch out for bit operations in conditionals.
In C/C++ any non-zero value is implicitly considered as "true". E.g. this C code:

```
if (x & 3) ...
```

must not be turned into this Lua code:

```
if band(x, 3) then ... -- wrong!
```

In Lua all objects except nil and false are considered "true". This includes all numbers. An explicit comparison against zero is required in this case:

```
if band(x, 3) ~= 0 then ... -- correct!
```

© 2015 Promixis, LLC

Comparing Against Hex Literals

Comparing the results of bitwise operations (signed numbers) against hex literals (unsigned numbers) needs some additional care. The following conditional expression may or may not work right, depending on the platform you run it on:

```
bit.bor(x, 1) == 0xffffffff
```

E.g. it's never true on a Lua installation with the default number type. Some simple solutions:

Either never use hex literals larger than 0x7fffffff in comparisons:

```
bit.bor(x, 1) == -1
```

Or convert them with bit.tobit() before comparing:

```
bit.bor(x, 1) == bit.tobit(0xffffffff)
```

Or use a generic workaround with bit.bxor():

```
bit.bxor(bit.bor(x, 1), 0xffffffff) == 0
```

Or use a case-specific workaround:

```
bit.rshift(x, 1) == 0x7fffffff
```

This text was adapted from the BitOp library documentation.

18.2 class

This namespace holds the unified class implementation for Girder. It is suggested you follow this class structure when building your own scripts.

Example

The basic structure of a new class is as follows

```
local Base = require('Class') -- the base class for this object.

module (...)

Base:subclass(_M)

local privateNumber = 1

local privateFunction = function(self )

end

function init ( self, param1, param2 )
	print(param1, param2)
	Base.init(self)
end
```

© 2015 Promixis, LLC

```
function deinit( self )
      Base.deinit(self)

end
```

If you want to subclass this new class simply require it on the first line and assign it to the base variable. Any public variables or functions will be part of the public interface of the class. Any local functions or variables will be private.

If we saved this class under the name "example/test.lua" we could use the following code to instantiate the class:

```
local ExampleClass = require('example.test')
local exampleObject = ExampleClass:new( param1, param2 )
```

As you can see param1 and param2 will be passed to the init function of our class. You can have as many of these as you like. Also note that the class names are capitalized and the variable names start with a small letter. This helps keep the Class vs. instantiated objects clear.

18.3 cm11a

The cm11a table allows you control over the CM11A's on your system from Lua.

18.3.1 getList

Returns a list of port names of attached CM11A units.

Definition

```
portNameList = cm11a.getList()
```

Name	Type	Description
portNameList	table of strings	Names by which you can refer to the CM11A's attached to your system.

© 2015 Promixis, LLC

Example

```
table.print( cm11a.getList() )
```

This will print

```
Wed Jan 22 15:21:29 2014       { -- #0
Wed Jan 22 15:21:29 2014          [1] = "COM7",
Wed Jan 22 15:21:29 2014       } -- #0
```

18.3.2 registerForX10Events

cm11a.registerForX10Events registers a callback that is called when an X10 event arrives.

Definition

```
regId = cm11a.registerForX10Events( callback )
```

Name	Type	Description
callback	function	The callback function.
regId	number	The id used to register with the plugin. This can be used with cm11a.unregisterFromX10Events(regId) to unregister from events.

Callback Definition

```
function( addresses, command, v )
```

Name	Type	Description
addresses	table of strings	The addresses referenced by this command
command	number (see cm1a.commands)	The command used. See the lua table for their meaning.
v	value	Dim and Bright pass a v, otherwise nil

© 2015 Promixis, LLC

Example

```
local x = cm11a.registerForX10Events( function( addresses, command, v )

  table.print(addresses)
  print(command,v)

end)
```

For a B2 Off command you'd get:

```
Wed Jan 22 15:28:12 2014      { -- #0
Wed Jan 22 15:28:12 2014         [1] = "B2",
Wed Jan 22 15:28:12 2014      } -- #0
Wed Jan 22 15:28:12 2014      3      nil
```

18.3.3 X10 Commands

There are 10 commands available for controlling X10 devices:

```
result = cm11a.allLightsOff( portname, callback )
result = cm11a.allLightsOn( portname, callback )
result = cm11a.allOff( portname, callback )
result = cm11a.on( portname, houseCode, deviceCode, callback )
result = cm11a.off( portname, houseCode, deviceCode, callback )
result = cm11a.dim( portname, houseCode, deviceCode, percent, callback )
result = cm11a.bright( portname, houseCode, deviceCode, percent,
callback )
result = cm11a.deviceStatus( portname, houseCode, deviceCode, callback )
result = cm11a.hail( portname, houseCode, deviceCode, callback )
result = cm11a.sendBinary( binary, callback )
```

Name	Type	Description
callback	function (sent)	The callback function, sent is a boolean
portname	string	The portname of the CM11A to use for sending. cm11a.getList() returns a list of available outputs.
houseCode	Single character	Character "A" through "P"
deviceCode	number	Device Code 1 through 16
percent	dim steps	0-100 (internally X10 only 22 steps so your value will be

© 2015 Promixis, LLC

		rounded to the nearest increment)
binary	string	You can send your own commands using this. The command, device and houseCode ids are available in cm11a.commands, cm11a.houseCodes and cm11a.deviceCodes
result	true or nil, error	If the CM11A with portname is not available an error will be returned.

18.4 date

This table holds the date and time manipulation routines.

18.4.1 Date Object

The date object provides a range of properties and functions to manipulate dates and times.

Properties
Methods
Operators

Properties

Name	Type	Description
year	number	The year
month	number	The month
day	number	The day
hour	number	The hour
minute	number	The minutes
second	number	The second
millisecond	number	The millisecond. Note that the date object does not have millisecond granularity.

© 2015 Promixis, LLC

Methods

Return	Signature	Flags
Time	**time**()	[const]
	setTime(Time)	
number	**toTime_t**()	[const]
	setTime_t (number)	
	addSeconds (number)	
	addDays (number)	
	addMonths (number)	
	addYears (number)	
boolean	**isValid** ( )	[const]
number	**secsTo** (Date)	[const]
number	**daysTo** (Date)	[const]
Date	**toLocal** ( )	[const]
Date	**toUtc** ( )	[const]
boolean	**isLocal** ( )	[const]

Date::time()

[const] Returns the Time object for this Date object. Since the date object is really a Date-Time object. This does not change the object's value as it makes a copy of the time.

Date::setTime(Time)

Sets the Time object for the Date object.

Date::toTime_t()

[const] Returns the number of seconds since 1970 Jan 1 0:00 UTC.

Date::setTime_t(secondsSinceEpoch)

Sets the date object to the date that corresponds to seconds since 1970 Jan 1 0:00 UTC.

Date::addSeconds(seconds)

Adds seconds to the date value.

© 2015 Promixis, LLC

Date::addDays(days)

Adds days to the date value.

Date::addMonths(months)

Adds months to the date value.

Date::addYears(years)

Adds years to the date value.

Date::isValid()

[const] Returns true if the date is valid.

Date::secsTo(Date)

[const] Returns the number of seconds to the parameter Date object.

Date::daysTo(Date)

[const] Returns the number of days to the parameter Date object.

Date::toLocal()

[const] Returns a new Date object set to local time.

Date::toUtc()

[const] Returns a new Date object set to Utc time.

Date::isLocal()

[const] Returns true if the time is set to local time.

Operators

The date object has a few operators overloaded

Date::-Time
Date::-number
Date::+Time
Date::+number
Date::<,>,<=,>=,== Date

Date::- Time

Subtract time from date.

© 2015 Promixis, LLC

Date::- number

Subtract number of seconds from date

Date::+ Time

Add time to date.

Date::+ number

Add number of seconds to date.

Date::<, >, <=, >=, == Date

[const] The usual relational operators allowing for quick date comparisons.

Example

```
local localTime = date.now()
print(localTime)
localTime = localTime + 100
print(localTime)
```

Prints

```
Sun Oct 14 21:59:41 2012
Sun Oct 14 22:01:21 2012
```

Related

Time Object
Date functions

18.4.2 newDate

Creates a new Date object. The value of the date is dependent on the optional parameter passed along.

Definition

```
Date = date.newDate( param )
```

Name	Type	Description

© 2015 Promixis, LLC

param *[optional]*	string or number	When a string is passed the function tries to parse it into a date and time. If a number is passed it is interpreted as the number of seconds since January 1st 1970, 0:00 UTC.
Date	Date Object	Date return value object.

Example

```
local d = date.newDate(6 * 3600)
print(d)
```

This will print

```
Thu Jan 1 01:00:00 1970
```

If this function was run in the Eastern time zone (UTC -5)

Related

Date Object

Availability

Lua

18.4.3 newTime

This creates a new Time object. Passing a number allows the time since 0:00 to be set in milliseconds.

Definition

```
Time = date.newTime( hour, minute, seconds, milliseconds )
```

Name	Type	Description
hour *[optional]*	number	hours since midnight
minutes *[optional]*	number	minutes passed the hour
seconds *[optional]*	number	seconds
milliseconds *[optional]*	number	milliseconds
Time	Time object	Returns a Time object

© 2015 Promixis, LLC

Example

```
myTime = date.newTime( 16, 30, 10 )
print(myTime)
```

This will print

```
16:30:10
```

Related

Time object
Date object

Availability

Lua

18.4.4 now

This function creates a date object set to the current local time.

Definition

```
myDate = date.now()
```

Name	Type	Description
myDate	Date Object	The date object set to the current local time.

Example

```
myDate = date.now()
print(myDate)
```

prints

```
Mon Oct 15 9:10:15 2012
```

Related

© 2015 Promixis, LLC

Time object
Date object

Availability

Lua

18.4.5 nowUtc

This function creates a date object set to the current UTC time.

Definition

```
myDate = date.nowUtc()
```

Name	Type	Description
myDate	Date Object	The date object set to the current UTC time.

Example

```
myDate = date.nowUtc()
print(myDate)
```

prints

```
Mon Oct 15 14:10:15 2012
```

Related

Time object
Date object

Availability

Lua

18.4.6 Time Object

The time object holds a time object able to express times from 0:00 to 23:59.

Properties
Methods
Operators

© 2015 Promixis, LLC

Properties

Name	Type	Description
hour	number	The hour part of the time [0..23]
minute	number	The minute part of time [0..59]
second	number	The second part of time [0..59]
millisecond	number	The milliseconds [0..999]
elapsed [const]	number	time in milliseconds since start was called

Methods

Return	Signature	Description
	start()	Starts the 24 hour stopwatch.
	restart()	restarts the stopwatch
boolean	isValid()	returns true if this object is valid.

start()

Starts the 24 hour stopwatch. Note that this object can at maximum count 24 hours.

restart()

Restarts the 24 hour stopwatch.

isValid()

isValid returns true if the timer countains a valid time.

Operators

Time::+ Time
Time::- Time

© 2015 Promixis, LLC

Time::<, <=, >, >=, == Time

Time::+Time / Time::-Time

Adds or subtracts time. Note that time will wrap at either 0 or 24 hours as it cannot give negative numbers or hours beyond 24. To go beyond 24 hours use the Date object and add to that.

Time Relational operators

The relational operators allow quick comparison between time objects.

Examples

```
local t1 = date.newTime(10,10)
local t2 = date.newTime(10,20)
local t3 = date.newTime(0,30)

print( t1 < t2 )

t1 = t1 + t3

print( t1 < t2 )
```

This will print

```
true
true
```

Related

Date Object
newTime

Availability

Lua

18.4.7 utcOffset

Returns the offset in seconds from GMT for the local computer on a certain day.

Definition

```
seconds = date.utcOffset( date )
```

© 2015 Promixis, LLC

Name	Type	Description
seconds	number	seconds offset from UTC for example Florida in winter is -300
date	Date Object	optional date object to specify the date you want to know the offset for. If not given current date is used.

Example

```
seconds = date.utcOffset( )
print(seconds)
```

Availability

Lua

18.5 delay

timer.new creates a new timer object.
Delay allows you to quickly schedule the delayed execution of a lua function.

Definition

```
timerObj = delay.run( timeout, callback )
```

Name	Type	Description
timerObj	Timer Object	The underlying timer object.
timeout	number	Number of miliseconds till callback is called.
callback	function(Timer Object)	The function to call.

Example

```
delay.run(500, function(t)
  print("hello")
end)
```

© 2015 Promixis, LLC

This prints "hello" after 500 milliseconds.

Related

Timer Object

Availability

Lua

18.6 deviceManager

The device manager is a core concept inside Girder. This is the script access to this object.

18.6.1 areEventsBlocked

Returns true if events are currently being blocked, false if not.

Definition

```
block = deviceManager.areEventsBlocked( deviceId )
```

Name	Type	Description
block	boolean	true if currently blocked.
deviceId	number	Id of the device.

Related

Control Object

Availability

Lua

18.6.2 blockEvents

Blocks event generation by value change of a device's controls.

Definition

© 2015 Promixis, LLC

```
success = deviceManager.blockEvents( deviceId )
```

Name	Type	Description
success	boolean	true if successful.
deviceId	number	Id of the device.

Related

Control Object

Availability

Lua

18.6.3 component

Returns the component with the given id.

Definition

```
components = deviceManager.component( id )
```

Name	Type	Description
component	Component	Return value of type Component
id	number	Id of the component

Example

```
component = deviceManager.component( 1 )
print(component.name)
print(component.enabled)
```

Related

Component Object

Availability

Lua

© 2015 Promixis, LLC

18.6.4 Component Object

The component object represents the components loaded in Girder.

Properties

Name	Type	Description
name	string	name of component
pluginId	number	Id of the plugin that owns this component
enabled	boolean	true if component is enabled, false otherwise
id	number	Id of the component
config	string	Typically a JSON string containing whatever config the object needs.
internalId	string	An component specific Id for this component.

Related

deviceManager.component
deviceManager.components

18.6.5 components

Lists the components in the system.

Definition

```
components = deviceManager.components( pluginId )
```

Name	Type	Description
components	table of components	Returns a table of components indexed by component Id.
pluginId	number	Number of the plugin that owns the components or 0 for all.

© 2015 Promixis, LLC

Example

```
local components = deviceManager.components(0)

for id, component in pairs(components) do
      print(id, component.name)
end
```

Related

Component object

Availability

Lua

18.6.6 control

Return the control object.

Definition

```
control = deviceManager.control( controlId )
control = deviceManager.control( controlPath )
```

Name	Type	Description
control	control object	The control object requested or nil if not found.
controlId	number	The Id of the control requested.
controlPath	string	The Path of the control.

Example

```
control = deviceManager.control( 1 )
print(control.name)
```

will print the name of control with id 1.

© 2015 Promixis, LLC

```
control, err = deviceManager.control( "Home///Projector///Focus +")

if control then
    deviceManager.requestControlValueChange(control.id, 1)
else
    print(err)
end
```

Will send out the IR signal Focus + to the Projector at home.

Related

Control Object

Availability

Lua

18.6.7 Control Object

Properties

Name	Type	Description
name	string	name of control
deviceId	number	Id of the device that owns this control
id	number	Id of the device
config	string	Typically a JSON string containing whatever config the object needs.
internalId	number	The plugin specific internal id.
value	string	value of the control
valueType	number	The number of the control type.

Related

deviceManager.control
deviceManager.controls

18.6.8 controls

Get a list of controls owned by a device.

Definition

```
controls = deviceManager.controls( deviceId )
```

Name	Type	Description
controls	table	table of control objects indexed by control id.
deviceId	number	Id of the device that owns the controls.

Example

```
controls = deviceManager.controls( 1 )
for id, control in pairs(controls) do
      print(id, control.name)
end
```

Related

Control Object

Availability

Lua

18.6.9 device

Return the device object.

Definition

```
device = deviceManager.device( deviceId )
```

Name	Type	Description
device	Device Object	The device object requested

© 2015 Promixis, LLC

		or nil if not found.
deviceId	number	The Id of the device requested.

Example

```
device = deviceManager.device( 1 )
print(device.name)
```

Related

Device Object

Availability

Lua

18.6.10 Device Object

Properties

Name	Type	Description
name	string	name of device
componentId	number	Id of the component that owns this device
enabled	boolean	true if device is enabled, false otherwise
id	number	Id of the device
config	string	Typically a JSON string containing whatever config the object needs.
locationId	number	The Id of the location that this device is nested in.

Related

deviceManager.devices
deviceManager.device

18.6.11 devices

Get a list of devices owned by a component

Definition

```
devices = deviceManager.devices( componentId )
```

Name	Type	Description
devices	table	table of Device objects indexed by device id.
componentId	number	Id of the component that owns this requested devices.

Example

```
devices = deviceManager.devices( 1 )
for id, device in pairs(devices) do
      print(id, device.name)
end
```

Related

Device Object

Availability

Lua

18.6.12 Locations

Get a list of locations defined inside Girder

Definition

```
locations = deviceManager.locations( )
```

© 2015 Promixis, LLC

Name	Type	Description
locations	table	table of location name strings indexed by location Id

Example

```
locations = deviceManager.locations( )
for id, name in pairs(locations) do
        print(id, name)
end
```

Availability

Lua

18.6.13 requestControlValueChange

The only way to change the value of a control is to request the change from the device manager system. The request will then be routed to the specific code that controls the control in question. Only if that control accepts the change will the value actually change.

Definition

```
deviceManager.requestControlValueChange( controlId, value, sender)
```

Name	Type	Description
controlId	number	The Id of the control to change.
value	string	The value to change the control to.
sender	string	The string identifying the sender of the request.

Example

```
deviceManager.requestControlValueChange(1, "100", "Web Interface")
```

© 2015 Promixis, LLC

Availability

Lua

18.7 gd

This is the Lua-GD library. Documentation can be found here: http://www.promixis.com/lua/luagd

18.8 gir

The girder table.

18.8.1 addEventHandler

The addEventHandler function allows for a script to register a callback with the event processing core. This allows the script to catch events as they are processed by Girder. This is useful to create advanced event processing flows. Note that the callback is called on the Girder event thread and as such all processing should be kept to a minimum. If any tasks take too long they should be spun onto their own thread.

Definition

```
handlerId = gir.addEventHandler( eventString, minDevice, maxDevice,
eventHandlerCallback )
```

Name	Type	Description
eventstring	string	The eventstring to match against. This is a regular expression.
minDevice	number	The lower bound of the event device number to match
maxDevice	number	The high bound of the event device number to match
eventHandlerCallback	callback function	The function that is called when the event matches the regular expression and device range.
handlerId	number	The id that can be used to unregister event handler

© 2015 Promixis, LLC

Callback Signature

The callback function has the following signature:

```
eventHandlerCallback ( eventString, deviceNumber, keyMod, payloads,
captures )
```

Name	Type	Description
eventString	string	The eventstring of the event that just matched the parameters given to addEventHandler
deviceNumber	number	The device number of the plugin that created the event.
keyMod	number (enum)	Indicates whether this was a keydown, repeat or keyup.
payloads	table of strings	A table filled with the payloads that came with this event.
captures	table of strings	If the eventString passed to addEventHandler had captures, this is where they will be passed back to the script.

Examples

Example - Device Manager Events

Below is an example showing how this function can be used to catch all events from the device manager.

```
handlerId = gir.addEventHandler("dm\\..*", 18, 18,
    function ( event, device, keyMod, payloads, captures )
        print(event, device)
    end
)
```

The event string is a regular expression. In the example above this is "dm\\..*" part. A quick primer to regular expressions can be found here. Events that this expression will match are dm.toggle.18 or dm.button.32. This example expression is built out of three parts. "dm", "\\." and ".*". The first part matches "dm" only. The second part only matches a dot "." and the last part matches zero or more characters of any kind.

© 2015 Promixis, LLC

Example - All Events

By passing .* as the regular expression we catch all events from the system with event device numbers between 0 and 65000.

```
handlerId, err = gir.addEventHandler( ".*", 0, 65000,
function( eventString ,eventDevice, keyMod, payloads, captures )

    print("Event: ", eventString, eventDevice, keyMod)
    table.print(payloads)

end)

if not handlerId then
    print("Add Handler registration failed.", err)
else
    print("Handler registered.")
end
```

Example - PIR-1 Button handling with captures

One useful thing to be able to do is capture the button presses from a PIR-1 and extract the button that was pressed and which PIR-1 triggered it. Potentially to strip the PIR-1 serial from it and resend the event. PIR-1's event looks like this:

```
"A4931313633351612171 button 1"
```

```
regId, err = gir.addEventHandler( "([a-fA-F0-9]{20}) button (\\d)", 123,
123, function( eventString ,eventDevice, keyMod, payloads, captures )

    print("Event: ", eventString, eventDevice, keyMod, #captures)
    table.print(captures)
      if #captures >= 3 then
            print("PIR: ", captures[2], " button: ", captures[3])
      end

end)

if not regId then
    print("Add Handler registration failed.", err)
else
    print("Handler registered.")
end
```

The main difference with previous examples is the regular expression. It now has two captures. These are the parts between parenthesis. First it's `([a-zA-Z0-9]{20})` and `(\d)` . The first part matches any 20 hexadecimal character string. The second part

matches any single number. Note that the returned captures includes the whole match as the first capture. Thus the PIR-1 serial is the second capture and the button is the third. (Note Lua's tables are 1 based, not zero as in C or other languages).

To read more about regular expression go to the Regular Expression Chapter.

Example - Removing Event Handler

If you need to unregister the handler that is possible. Make sure you stored the handlerId that was returned by gir.addEventHandler and then pass that to gir.removeEventHandler.

```
if handlerId then
    gir.removeEventHandler(handlerId)
    handlerId = nil
end
```

Related

gir.removeEventHandler
thread.newthread

Availability

Lua

18.8.2 getversion

getVersion retrieves the Girder version currently.

Definition

```
version = gir.getVersion()
```

Name	Type	Description
version	table	table indexed with major, minor, micro and build indicating the Girder version

Example

```
local t = gir.getVersion()

print(t.major, t.minor, t.micro, t.build)
```

© 2015 Promixis, LLC

18.8.3 girderClosing

This is a boolean that is normally not present. Only during script reset or shutdown is this available and set to true.

18.8.4 log

Allows a log message to be generated from Lua.

Definition

```
gir.log(logLevel, source, message[, fileId, nodeId])
```

Name	Type	Description
logLevel	Promixis.Log	Constant that indicates level of severity.
source	string	The source of the error (for example Lua, or X10)
message	string	The message to print.
fileId	string	The file id of the node that produced the error. (Optional)
nodeId	int	The node id of the node that produced the error. (Optional

Example

```
gir.log(Promixis.Log.OK, "Lua", "This is just a message")
```

Logs the Message "This is just a message" from source "Lua" and severity "Ok" in the Girder log.

Related

Promixis.Log

Availability

Lua

© 2015 Promixis, LLC

18.8.5 parseString

This function parses a string for the square bracket notation. For example

"Hello my Lua script is version: [_VERSION]" will be parsed to

"Hello my Lua script is version: 5.1"

Definition

```
pstr = gir.parseString(str)
```

Name	Type	Description
str	string	Original String
pstr	string	Parsed String

18.8.6 removeEventHandler

Once an event handler is no longer needed unregister it with this function. It is good practice to unregister event handlers once they are not needed anymore as they are processed every time an event arrives.

Definition

```
gir.removeEventHandler( handlerId )
```

Name	Type	Description
handlerId	number	This is the id returned by addEventHandler

Example

Below is an example that first registers an event handler. Once the event "StopMe" arrives the event handler unregisters.

```
local handlerId
handlerId = gir.addEventHandler(".*", 18, 18, function ( event, device,
keyMod, payloads )
	print(event, device)
	if event == "StopMe" then
		gir.removeEventHandler( handlerId )
		handlerId = nil
	end
```

© 2015 Promixis, LLC

```
end)
```

Related

gir.addEventHandler

Availability

Lua

18.8.7 settings

The settings objects inside Girder can be access using the girder.settings() call. This returns the settings object inside Girder. Note that you can modify settings using this method.

Definition

```
local settings = gir.settings()
```

The settings object has several properties.

Name	Type	Description
latitude	number	The estimated latitude of this computer
longitude	number	The estimated longitude of this computer
country	string	The country
state	string	The state
city	string	The city
installPath	string	The path to the Girder installation
settingsPath	string	The path to the Girder settings directory
hostname	string	The host name of the computer
username	string	The user name under which the Girder core is running
luaPath	string	The path to the lua script files
webserver	IWebserverSettings object	The webserver settings object
proxy	IProxySettings object	The proxy settings object

© 2015 Promixis, LLC

email	IEmailSettings object	The email settings object

Related

Webserver Settings
Proxy Settings
Email Settings

Availability

Lua
Plugins
Javascript

18.8.7.1 Webserver Settings

The web server settings are stored in this object.

Name	Type	Description
plainPort	number	Port number of the HTTP server
securePort	number	Port number of the HTTPS server
runPlain	boolean	Decides if the HTTP server runs
runSecure	boolean	Decides if the HTTPS server runs
caFile	string	Filename of the CA certificate
certFile	string	Filename of the certificate of the server
keyFile	string	Filename of the private key
defaultPath	string	Path to the HTTP directory

Related

settings

Availability

Lua
Plugins
Javascript

© 2015 Promixis, LLC

18.8.7.2 Proxy Settings

Global Proxy settings

Name	Type	Description
hostname	string	Hostname of the proxy server
username	string	User name if needed
password	string	Password if needed
port	number	Port number of proxy server
proxyType	Promixis.IProxySettings.ProxyType	Type of proxy

Related

Promixis.IProxySettings.ProxyType

Availability

Lua
Plugins
Javascript

18.8.7.3 Email Settings

The email settings

Name	Type	Description
hostname	string	Hostname of the email SMTP server
username	string	Username to login on SMTP server
password	string	Password for SMTP server
port	number	SMTP server port
sender	string	Default sender to use
errorTo	string	Recipient of Error reports from Girder
connectionType	Promixis.IEmailSettings.ConnectionType	Connection type (SSL, TLS or Plain)

Related

© 2015 Promixis, LLC

Promixis.IEmailSettings.ConnectionType

Availability

Lua
Plugins
Javascript

18.8.8 shutdownNotifier

The shutdown notifier is a object of class type publisher. Register with this object to be notified of script reset or Girder shutdown.

Example

```
gir.shutdownNotifier:subscribe( function ( )
  print("SHUTTING DOWN PANIC!!")
end)
```

This will print "SHUTTING DOWN PANIC!!" when the Lua script engine is reset or Girder is shutdown.

18.8.9 triggerEvent

Trigger event allows a script to send events into Girder. This could be useful when creating plugins or to do advanced scripting with Girder.

Definition

```
gir.triggerEvent(eventString, eventDevice, keyMod, payloads)
```

Name	Type	Description
eventString	string	The eventstring to send into Girder
eventDevice	number	The device number
keyMod	Promixis.Event.Modifiers	The modifier for this event

Example

```
local payloads = {
      "pop",
      "crackle"
```

© 2015 Promixis, LLC

```
}
gir.triggerEvent("Bang!", 18, Promixis.Event.MOD_ON, payloads)
```

This example will trigger the event "Bang!" on device 18 inside Girder with 2 payloads "pop" and "crackle".

Related

Promixis.EventNode.Modifiers

Availability

Lua

18.9 hid

Girder provides access to the HID subsystem to Lua via the hid namespace. To use a HID device you'll need to know it's vendor and product id.

18.9.1 enumerate

enumerate allows you to query the currently attached USB-HID devices.

Definition

```
deviceInfoList = hid.enumerate( vendorId, productId )
```

Parameters

Name	Type	Description
vendorId	number	the vendor id or 0 for all vendors.
productId	number	the product id or 0 for all products.
deviceInfoList	table	Table with device information.

© 2015 Promixis, LLC

DeviceInfoList

Name	Type	Description
path	string	The path to the USB device, for use in hid.open()
vendorId	number	The vendor Id
productId	number	The product Id
serial	string	The device serial number
manufacturer	string	The manufacturer string
product	string	The product string
interfaceNumber	number	The interface number

Example

```
table.print( hid.enumerate(0,0) )
```

With only one USB device attached:

```
Thu Jan 2 09:39:10 2014 { -- #0
Thu Jan 2 09:39:10 2014    [1] = { -- #1
Thu Jan 2 09:39:10 2014       ["path"] = "\\\\?\\hid#vid_20a0&pid_413f&mi_00#7&f979fa
Thu Jan 2 09:39:10 2014       ["manufacturer"] = "Promixis, LLC",
Thu Jan 2 09:39:10 2014       ["productId"] = 16703,
Thu Jan 2 09:39:10 2014       ["product"] = "Promixis, LLC",
Thu Jan 2 09:39:10 2014       ["interfaceNumber"] = 0,
Thu Jan 2 09:39:10 2014       ["serial"] = "A4931313633351612171",
Thu Jan 2 09:39:10 2014       ["vendorId"] = 8352,
Thu Jan 2 09:39:10 2014    } -- #1,
Thu Jan 2 09:39:10 2014 } -- #0
```

Note that it's not always possible to get HID reports from mouse and keyboard, specifically Microsoft Windows will block this.

18.9.2 open

Opens a device to allow reading and writing of data to endpoints or to get or set feature reports.

Definition

```
hidDevice, err = hid.open( vendorId, productId, serial )
```

```
hidDevice, err = hid.open( path )
```

Parameters

Name	Type	Description
vendorId	number	The vendor id or 0 for all vendors.
productId	number	The product id or 0 for all products.
path	string	The path to the hid device from enumeration.
hidDevice	Hid Device Object	Hid Device or nil
err	string	Descriptive error if hidDevice is nil

Example

The PIR-1 exposes two interface. The first is a keyboard the second is the IR generic hid device. To be able to open the second device we'll need to first use enumerate and try to open interface with id 0. On Linux interfaceNumber is not specified so just try to open both and use the one that actually opens.

```
function openFirstPIR1()

    local pir1Devices = hid.enumerate(0x20a0, 0x413f)
    table.print(pir1Devices)

    for idx, pir1Device in ipairs(pir1Devices) do

      if pir1Device.interfaceNumber == -1 or pir1Device.interfaceNumber
== 0 then
        local hidDev, err = hid.open(pir1Device.path)
        if hidDev then
            return hidDev
        else
            print(err)
        end

     end

    end

end
```

© 2015 Promixis, LLC

```
hidDev = openFirstPIR1()
print(hidDev)
```

18.9.3 HidDevice

The HidDevice object is use to interact with the opened hid device.

Methods

Return	Signature	Description
bytesWritten, error	write(data)	Writes data and returns number of bytes written
data, error	read(len, timeoutMS)	Reads data of length len and with timeout timeoutMS
boolean, error	setNonBlocking(boolean)	Sets the device to be non-blocking or not.
bytesWritten, error	setFeatureReport(data)	Writes a feature report
data, error	getFeatureReport(len, reportId)	Reads a feature report
string, error	getSerial()	Gets the serial number
string, error	getManufacturerString()	Gets the manufacturer string
string, error	getProductString()	Gets the product string
	close()	Closes the device.

HidDevice::write

The first byte of data must contain the Report ID. For devices which only support a single report, this must be set to 0x0. The remaining bytes contain the report data. Since the Report ID is mandatory, calls to write will always contain one more byte than the report contains. For example, if a hid report is 16 bytes long, 17 bytes must be passed to write, the Report ID (or 0x0, for devices with a single report), followed by the report data (16 bytes). In this example, the length passed in would be 17.

hid_write() will send the data on the first OUT endpoint, if one exists. If it does not, it will send the data through the Control Endpoint (Endpoint 0).

HidDevice::read

Input reports are returned to the host through the INTERRUPT IN endpoint. The first byte will contain the Report number if the device uses numbered reports. Timeout is the number of milliseconds to wait or 0 if wait indefinitely. Len is the length of the data to receive.

© 2015 Promixis, LLC

HidDevice::setNonBlocking

In non-blocking mode calls to read will return immediately with a value of 0 if there is no data to be read. In blocking mode, read will wait (block) until there is data to read before returning.

HidDevice::setFeatureReport

Feature reports are sent over the Control endpoint as a Set_Report transfer. The first byte of data must contain the Report ID. For devices which only support a single report, this must be set to 0x0. The remaining bytes contain the report data. Since the Report ID is mandatory, calls to setFeatureReport will always contain one more byte than the report contains. For example, if a hid report is 16 bytes long, 17 bytes must be passed to setFeatureReport: the Report ID (or 0x0, for devices which do not use numbered reports), followed by the report data (16 bytes). In this example, the length passed in would be 17.

HidDevice::getFeatureReport

Gets the feature report for report id reportId. Note that report id is passed as a number not a binary.

HidDevice::close

Closes the resources associated with the hid device. Make very sure that you are no longer using this device, especially any blocking HidDevice::read or getFeatureReport functions. These will crash Girder if you close while they are still running.

Examples

Sending data.

The code below sends a CCF code from a PIR-1 or PIR-4. The hidDev variable came from the example code in the hid.open section. What the function below does is basically split the CCF code into 60 byte chunks. Prepends a header with report id = 0, PIR command id = 14, flags 0 (continue), 1 (first packet) or 2 (last packet) plus the length of the data in bytes followed by the CCF data. The reason for splitting the CCF code like this is that USB-HID has a maximum transfer size of 64 bytes.

```
function sendCCF( dev, bitmask, repeats, ccf )

    local ccfParts = string.split(ccf," ")
    table.print(ccfParts)
    local firstPacket = true
    local data = '';
    for i, v in ipairs(ccfParts) do
```

© 2015 Promixis, LLC

```
      if i == 1 then
          data = data .. string.char( repeats ) .. string.char(bitmask)
      end


      local value = math.hextodecimal(v)

      local vl = bit.band(value,0xff)
      local vh = bit.rshift(value,8)

      data = data .. string.char(vh) .. string.char(vl)

      local lastPacket = i == #ccfParts;

      if string.len(data) >= 60 or lastPacket then

        local flags = 0;
        if firstPacket then
            flags = bit.bor(flags,1)
        end
        firstPacket = false;
        if lastPacket then
            flags = bit.bor(flags,2)
        end

        local reportHeader = '\000\014' .. string.char(flags) ..
string.char( string.len(data))
        local packet = reportHeader .. data
        hidDev:write(packet)
        data = ''

      end

    end

end

sendCCF(hidDev, 3, 2, "0000 006E 0000 0022 0156 00A9 0014 0014 0014 0040
0014 0040 0014 0040 0014 0014 0014 0040 0014 0040 0014 0014 0014 0040
0014 0014 0014 0014 0014 0014 0014 0040 0014 0014 0014 0014 0014 0040
0014 0040 0014 0014 0014 0040 0014 0040 0014 0040 0014 0014 0014 0014
0014 0014 0014 0014 0014 0040 0014 0014 0014 0014 0014 0014 0014 0040
0014 0040 0014 0040 0014 05EF")
```

Receiving data.

Receiving data is as easy as calling HidDevice::read(reportSize + 1, 1000). This is great as long as your device does not generate any data unsolicited. How would you know when to call read in that case. To solve this problem you can start a new thread that reads with a timeout.

```
hidTerminated = false

function receiveIrCodes( dev )

  thread.newthread( function()

    while not hidTerminated and not gir.girderClosing do

        local data, err = dev:read(65, 250)
        if not data then
            hidTerminated = nil
            print(err)
            return
        end

        if string.len(data) > 0 then
            print(string.len(data))
            print(math.binarytohex(data))
        end


    end

    hidTerminated = nil


  end,{})


end

receiveIrCodes(hidDev)
print("Receiving...")
```

Closing up.

Since we cannot rip the HidDevice from underneat the thread without crashing Girder we're going to do it the nice way. We check if hidTerminated has been set to false. If so we set it to true and wait for the thread to set it to false or nil.

```
if hidTerminated == false then
    hidTerminated = true
    print("Waiting...")
    while ( hidTerminated ) do
      print("Waiting...")
    end
end

hidDev:close()
```

© 2015 Promixis, LLC

```
hidDev = nil
```

The code above work just fine but it uses some global variables and is scattered across a few actions. Let's put it into a nice class.

18.9.4 PIR-1 HID Complete Example

To actually make the code useful and maintainable that we developed in the hid.open and HidDevice chapters we place it into an organized class. I suggest you do the same with your hid code. That way you can also share them easier with other users on the forum!

```
--[[
    PIR-1 Lua HID example
    Copyright 2014 (c) Promixis, LLC

      Usage
      =====

      Starting up:

      ph = require('examples.pirhid')
      PIR = ph.new()

      Sending a CCF code:
      PIR:sendCCF(3, 2, "0000 006E 0000 0022 0156 00A9 0014 0014 0014
0040 0014 0040 0014 0040 0014 0014 0014 0040 0014 0040 0014 0014 0014
0040 0014 0014 0014 0014 0014 0014 0014 0040 0014 0014 0014 0014 0014
0040 0014 0040 0014 0014 0014 0040 0014 0040 0014 0040 0014 0014 0014
0014 0014 0014 0014 0014 0014 0040 0014 0014 0014 0014 0014 0014 0014
0040 0014 0040 0014 0040 0014 05EF")

--]]

local hid = require('hid')
local bit = require('bit')
local string = require('string')
local math = require('math')
local Base = require('Class')
local table = require('table')
local thread = require('thread')
local print = print
local ipairs = ipairs
local gir = gir
local pcall = pcall
```

```
module (...)

Base:subclass( _M)

-- Opens the first available PIR-1
local openFirstPIR1 = function ()

    local pir1Devices = hid.enumerate(0x20a0, 0x413f)

    for idx, pir1Device in ipairs(pir1Devices) do

      if pir1Device.interfaceNumber == -1 or pir1Device.interfaceNumber
== 0 then
        local hidDev, err = hid.open(pir1Device.path)
        if hidDev then
            return hidDev
        else
            print(err)
        end

     end

    end

end


-- The Receive thread, this will exit automatically during a script
reset.
local recv = function ( self )

    while not self.terminated and not gir.girderClosing do

        local data, err = self.dev:read(65, 250)
        if not data then
            print(err)
            return
        end

        if string.len(data) > 0 then
            print(math.binarytohex(data))
        end

    end

end


-- Sends the CCF code to the loaded PIR with bitmask ( 1 = back, 2 =
front, 3 = back and front ) and repeat count.
function sendCCF( self, bitmask, repeats, ccf )
```

© 2015 Promixis, LLC

```
      if not self.dev then
            return false, "No Device"
      end

    local ccfParts = string.split(ccf," ")
    local firstPacket = true
    local data = '';
    for i, v in ipairs(ccfParts) do

      if i == 1 then
          data = data .. string.char( repeats ) .. string.char(bitmask)
      end

      local value = math.hextodecimal(v)
      local vl = bit.band(value,0xff)
      local vh = bit.rshift(value,8)

      data = data .. string.char(vh) .. string.char(vl)

      local lastPacket = i == #ccfParts;

      if string.len(data) >= 60 or lastPacket then

        local flags = 0;
        if firstPacket then
            flags = bit.bor(flags,1)
        end
        firstPacket = false;
        if lastPacket then
            flags = bit.bor(flags,2)
        end

        local reportHeader = '\000\014' .. string.char(flags) ..
string.char( string.len(data))
        local packet = reportHeader .. data

        if not self.dev:write(packet) then
                  return false, "No Device"
            end
        data = ''

      end

    end

      return true

end


function init ( self )
```

```
        Base.init(self)
        self.terminated = true
        self.dev = openFirstPIR1()
        self.terminated = false

        thread.newthread( function()
                self.threadRunning = true
                pcall(recv, self)
                self.threadRunning = false
        end, {})

end

function deinit( self )
        Base.deinit(self)

        self.terminated = true
        while self.threadRunning do
                -- loop loop loop
        end

        if self.dev then
                self.dev:close()
                self.dev = nil
        end
end
```

18.10 json

Json provides function to do JSON encoding and decoding.

Encode

```
jsonString = json.encode ( tableToEncode )
```

Name	Type	Description
tableToEncode	table	Table to encode to json
jsonString	string	JSON representation of tableToEncode

Example

```
json = require('json')
print( json.encode ( { a=10; b=20; c= { sub1= 2; sub2= 3 } } ) )
```

© 2015 Promixis, LLC

This will print {"a":10,"c":{"sub2":3,"sub1":2},"b":20}

Decode

```
resultTable = json.decode ( jsonString )
```

Name	Type	Description
resultTable	table	Table decoded from JSON string
jsonString	string	JSON representation of resultTable

Example

```
json = require('json')
table.print( json.decode ( '{"a":10,"c":{"sub2":3,"sub1":2},"b":20}' ))
```

This will print:

```
{ -- #0
    ["a"] = 10,
    ["c"] = { -- #1
        ["sub2"] = 3,
        ["sub1"] = 2,
    } -- #1,
    ["b"] = 20,
} -- #0
```

18.11 kv

The kv namespace allows you to store and share key value string pairs. These values are persisted between reboots. Note that the get function accepts wildcards using the "*" symbol. It is suggested that you create 'namespaces' for the keys. For example if you are working on a plugin that handles the toaster name your variables "toaster.burnTemperature".

Functions

Return Value	Signature	Description
	get(key, getCallback)	Gets the values matching key and calls the callback function once complete.
	set(keyValueTable,	sets the values in

	setCallback)	keyValueTable in the KV storage. Calls callback when done.
	delete (keyTable, deleteCallback)	deletes the keys in keyTable and calls callback when done.
number	register(eventCallback)	registers for KV events
	unregister(number)	unregisters for KV events

getCallback function signature

```
function ( table, success, timedout )
```

table contains the key-value table that match the query.

setCallback function signature

```
function ( success, timedout )
```

deleteCallback function signature

```
function ( success, timedout )
```

eventCallback function signature

```
function ( eventType, ... )
```

eventType can be "UPDATED", "DELETED"

eventCallback - UPDATED function signature

```
function ( eventType, keyValueTable )
```

eventCallback - DELETED function signature

```
function ( eventType, keyTable )
```

Example

The first example registers as a listener and prints all updated/new values on the lua console.

© 2015 Promixis, LLC

```
kv.register( function( event, kvs )

  print(event)
  if event == "UPDATED" then

      for key,value in pairs(kvs) do
          print(key, value)
      end

  end

end)
```

This example sets a few values in the kv-store.

```
local kvs = {}

kvs["myvalues.hello"] = "dooodaa";
kvs["myvalues.bye"] = "yeehaaw";

kv.set(kvs, function( success )
        print("Set KV: " , success )
end)
```

This example shows how to retrieve values.

```
kv.get("myvalues.*", function( values, success )

    for key, value in pairs(values) do
        print(key, value)
    end

    print("Done")

end)
```

Availability

Lua

See Also

Javascript KV

18.12 lfs

Lua File System.

© 2015 Promixis, LLC

18.13 lxp

The LXP namespace contains a XML parser. Full documentation can be found here:

http://promixis.com/lua/luaexpat/manual.html

Example

```
local lom = require('lxp.lom')
local io = require('io')
local table = require('table')

local f = io.open("c:/devel/test.xml")
if not f then
    print("Could not open input file")
    return
end

-- read the whole file:
local xml = f:read("*a")
f:close()

local xmlTable, err = lom.parse(xml)
if not xmlTable then
    print(err)
    return
end

table.print(xmlTable)
```

This example code will read all data from the file "C:\devel\test.xml", which in this example contained:

```
<?xml version="1.0" encoding="UTF-8"?>
<node1 p="asdf">
      Text in tag node1
      <node2>cool stuff</node2>
</node1>
```

If all goes well it will print the following output:

```
Wed Mar 12 11:22:01 2014      { -- #0
Wed Mar 12 11:22:01 2014         [1] = "\
Wed Mar 12 11:22:01 2014            Text in tag node1\
Wed Mar 12 11:22:01 2014            ",
Wed Mar 12 11:22:01 2014         [2] = { -- #1
Wed Mar 12 11:22:01 2014            [1] = "cool stuff",
Wed Mar 12 11:22:01 2014            ["attr"] = { -- #2
```

© 2015 Promixis, LLC

```
Wed Mar 12 11:22:01 2014            } -- #2,
Wed Mar 12 11:22:01 2014            ["tag"] = "node2",
Wed Mar 12 11:22:01 2014         } -- #1,
Wed Mar 12 11:22:01 2014         [3] = "  \
Wed Mar 12 11:22:01 2014      ",
Wed Mar 12 11:22:01 2014         ["attr"] = { -- #3
Wed Mar 12 11:22:01 2014            [1] = "p",
Wed Mar 12 11:22:01 2014            ["p"] = "asdf",
Wed Mar 12 11:22:01 2014         } -- #3,
Wed Mar 12 11:22:01 2014         ["tag"] = "node1",
Wed Mar 12 11:22:01 2014      } -- #0
```

18.14 math

Girder adds a few extra functions to the math library besides the usual Lua Math functions.

18.14.1 crc

Girder has a set of CRC function built in. These include: crc16, crc32, crcCCITT, crcDNP, crcKermit and crcSick. All are nested in the math table and have the same functions: init, update, finish and process.

Definitions for string input

```
crc = math.crc16.process( value )
crc = math.crc32.process( value )
crc = math.crcCCITT.process( value )
crc = math.crcDNP.process( value )
crc = math.crcKermit.process( value )
crc = math.crcSick.process( value )
```

Name	Type	Description
crc	number	CRC value
value	string	String to CRC

Definitions for incremental input

```
crcTemp = math.crc16.init()
crcTemp = math.crc32.init()
crcTemp = math.crcCCITT.init()
crcTemp = math.crcDNP.init()
crcTemp = math.crcKermit.init()
crcTemp = math.crcSick.init()
```

© 2015 Promixis, LLC

Name	Type	Description
crcTemp	number	CRC temporary calculation value

```
crcTemp = math.crc16.update( previousCrcTemp, byte )
crcTemp = math.crc32.update( previousCrcTemp, byte )
crcTemp = math.crcCCITT.update( previousCrcTemp, byte )
crcTemp = math.crcDNP.update( previousCrcTemp, byte )
crcTemp = math.crcKermit.update( previousCrcTemp, byte )
crcTemp = math.crcSick.update( previousCrcTemp, byte, previousByte )
```

Name	Type	Description
crcTemp	number	CRC temporary calculation value
previousCrcTemp	number	CRC temporary calculation value of previous cycle (or value from init call)
byte	number	Byte value (0-255) of value to add to calculation
previousByte	number	Byte value of previous cycle.

```
crc = math.crc16.finish( previousCrcTemp )
crc = math.crc32.finish( previousCrcTemp )
crc = math.crcCCITT.finish( previousCrcTemp )
crc = math.crcDNP.finish( previousCrcTemp )
crc = math.crcKermit.finish( previousCrcTemp )
crc = math.crcSick.finish( previousCrcTemp )
```

Name	Type	Description
crcTemp	number	CRC value
previousCrcTemp	number	CRC temporary calculation value of previous cycle (or value from init call)

Examples

Here is an example of how to use the incremental input CRC functions.

```
function process_crc_sick ( s, init )
      if not init then
            init = math.crcSick.init()
```

© 2015 Promixis, LLC

```
        end
        local last = 0
        for i=1, string.len(s) do
                local b = string.byte(s,i)
                init = math.crcSick.update(init,b,last)
                last = b
        end
        return math.crcSick.finish(init)
end
print( process_crc_sick( "Hello World" ) )
```

or an example to directly calculate the CRC:

```
print( math.crc32.process( "Hello World" ) )
```

18.14.2 hexStringToBinary

Definition

```
binary = math.hexStringToBinary ( hex )
```

Name	Type	Description
binary	string	Binary representation of hex
hex	string	Hex string for example "0A A0 BF 23"

Example

```
print( math.hexStringToBinary("48 65 6C 6C 6F 20 57 6F 72 6C 64") )
```

This will print "Hello World"

18.14.3 binaryToHexString

Definition

```
hex = math.binaryToHexString ( binary )
```

Name	Type	Description

binary	string	Binary representation of hex
hex	string	Hex string for example "0A A0 BF 23"

Example

```
print( math.binaryToHexString( "Hello World" ) )
```

This will print "48 65 6C 6C 6F 20 57 6F 72 6C 64"

18.14.4 formatbytes

Definition

```
formatted = math.formatbytes ( binary )
```

Name	Type	Description
binary	string	Binary representation of hex
formatted	string	Hex string for example "0A A0 BF 23"

Example

```
print( math.formatbytes( "Hello World" ) )
```

This will print "48 65 6C 6C 6F 20 57 6F 72 6C 64   Hello World"

18.14.5 hextodecimal

Definition

```
value = math.hextodecimal ( hex )
```

Name	Type	Description
value	number	Number represented by hex
hex	string	Hex value to convert to number

© 2015 Promixis, LLC

Example

```
print( math.hextodecimal( "48" ) )
```

This will print 72

18.14.6 decimaltohex

Definition

```
hex = math.hextodecimal ( value )
```

Name	Type	Description
value	number	Number represented by hex
hex	string	Hex value to convert to number

Example

```
print( math.decimaltohex( 78 ) )
```

This will print "48"

18.14.7 decimaltobyte

Definition

```
byte = math.decimaltobyte ( value )
```

Name	Type	Description
value	number	number of byte
byte	character	binary value of value

Example

```
print( math.decimaltobyte( 72 ) )
```

This will print "H"

© 2015 Promixis, LLC

18.14.8 binarytohex

Definition

```
hex = math.binarytohex ( byte )
```

Name	Type	Description
hex	string	hex value of byte
byte	character	binary value of value

Example

```
print( math.binarytohex( "H" ) )
```

This will print "48"

18.15 mime

Mime is part of the lua socket library. You can find it's documentation here: http://www.promixis.com/lua/luasocket/mime.html

18.16 network

Girder provides a few standard network operations. HTTP Get, HTTP Post and Email send.

18.16.1 get

This function does a standard HTTP get call on the URL provided optionally with a username and password. The result is passed back into the callback function.

Definition

```
network.get( url, callback, timeout, username, password, headers )
```

Name	Type	Description
url	string	Fully qualified URL. For example HTTP://www.promixis.com
callback	function	The function to be called with the result.
timeout	number	Timeout for operation in ms

© 2015 Promixis, LLC

username [optional]	string	Username to use in authentication.
password [optional]	string	Password to use in authentication.
headers [optional]	table of strings	Headers to send with request. Each string must be of the form HEADER: Value. For example {"Agent: Girder", "FakeHeader": "FakeValue"}

Callback function signature

```
function ( success, status, body )
```

Name	Type	Description
success	boolean	Returns true if the HTTP call was successful. Note that this could still mean the status code is not 200 (= HTTP OK).
status	number	HTTP Status code. 200 = OK.
body	string	The body of the response.

Example

```
network.get( "http://www.promixis.com", function ( success, status,
body )

if not success then
      print("Sorry could not connect with server")
      return
end

if status ~= 200 then
      print("Sorry server returned", status)
      return
end

print(body)

end, 3000)
```

Returns the content of the Promixis home page, which starts with something like

<html><head>...

Related

post

Availability

Lua

18.16.2 post

This function does a standard HTTP get call on the URL provided optionally with a username and password. The result is passed back into the callback function.

Definition

```
network.post( url, postData, mimeType, callback, timeout, username,
password, headers )
```

Name	Type	Description
url	string	Fully qualified URL. For example HTTP://www.promixis.com
postData	string	Data to post.
mimeType	string	The mime type of the postData.
callback	function	The function to be called with the result.
timeout	number	Timeout for operation in ms
username [optional]	string	Username to use in authentication.
password [optional]	string	Password to use in authentication.
headers	table of strings	Headers to send with request. Each string must be of the form HEADER: Value. For example {"Agent: Girder", "FakeHeader": "FakeValue"}

Callback function signature

```
function ( success, status, body )
```

© 2015 Promixis, LLC

Name	Type	Description
success	boolean	Returns true if the HTTP call was successful. Note that this could still mean the status code is not 200 (= HTTP OK).
status	number	HTTP Status code. 200 = OK.
body	string	The body of the response.

Example

```
network.post( "http://www.yoururl.com/form.php", 'name=20&id=100', 'data/
urlencoded', function ( success, status, body )

      if not success then
            print("Sorry could not connect with server")
            return
      end

      if status ~= 200 then
            print("Sorry server returned", status)
            return
      end

      print(body)

end)
```

Submits the url encoded form data to www.yoururl.com/form.php and prints the returned body.

Related

get

Availability

Lua

18.16.3 sendEmail

sendEmail requires that the email settings (SMTP server etc) are filled out on the settings dialog.

© 2015 Promixis, LLC

Definition

```
network.sendEmail( to, from, subject, plainBody, htmlBody, callback )
```

Name	Type	Description
to	string	recipient for example sales@promixis.com.
from	string	sender santa@clause.com
subject	string	Subject of the email.
plainBody	string	The plain text email body. Note that either plainBody or htmlBody or both must be provided. If you wish not to provide either plain or html place an empty string in place.
htmlBody	string	The HTML text email body.
callback	function	The callback is called after email was successfully sent. Note this does not mean the email was successfully received.

Callback function signature

```
function ( success )
```

Name	Type	Description
success	boolean	Set to true if email was sent successfully, false otherwise.

Example

```
local body = [[
Dear Promixis,

Can we get another good deal on the 432,312,364
Girder Pro licenses we need this year? Can you
deliver these as CD's so we can put them under
```

© 2015 Promixis, LLC

```
the trees of all the good home automation fans?

Sincerely

  Mr. Claus
]]

network.sendEmail( "sales@promixis.com", "santa@clause.com", "Licenses
for Christmas", body, "", function(success)
      if success then
            print("Email sent")
      else
            print("Email send failed")
      end
end )
```

This will print

```
Email sent
```

Related

get
post

Availability

Lua
Actions

18.16.4 Wake On Lan

sendEmail requires that the email settings (SMTP server etc) are filled out on the settings dialog.

Definition

```
network.wol( ip, mac )
```

Name	Type	Description
ip	string	broadcast ip address. You can use 255.255.255.255
mac	string	MAC address of target (00-50-56-C0-00-01) sometimes called physical address.

© 2015 Promixis, LLC

Example

```
network.wol( "255.255.255.255", "00-50-56-C0-00-01" )
```

Availability

Lua

18.17 onewire

The onewire table supplies support for the Maxim Integrated line of 1-Wire hardware. The support here is built with the 1-Wire public domain kit as such you can find example on how to use these function by following the examples in the kit. This is most definitely an advanced topic.

Examples

Opening a 1-Wire connection

The first examples shows how to open a 1-Wire connection using a DS2490 (USB connected 1-Wire hardware)

```
if portnum then
    print("Already opened")
    return
end

local devices = onewire.DS2490.names()

if #devices == 0 then
    print("No 1-Wire devices.")
    return
end

table.print(devices)

portnum = onewire.DS2490.acquire( devices[1] )
print(portnum)
if portnum <= 0 then
    portnum = nil
    print("Could not open.")
end
```

Once this script runs a global variable called portnum will be set. This is your key to using the rest of the onewire functions.

Listing 1-Wire devices

This code prints a list of serial numbers of connected 1-Wire devices.

```
if not portnum then
    print("Port not opened.")
    return
end


local result = onewire.owFirst(portnum, true, false)
print("reuslt", result)
while ( result ) do
    print( onewire.owSerialNum( portnum ) )
    result = onewire.owNext(portnum, true, false)
end
```

Read temperature

The code below reads the temperature of the 1-Wire device with serial number DA000802A0AB5410.

```
print(onewire.DS1920.getTemperature(portnum, "DA000802A0AB5410", 1000))
```

Read humidity

While 1-Wire doesn't directly support humidity measurements there is hardware out there that uses a DS2438 together with a humidity sensor hooked to it's A2D port. (For example the HT3-R1-A from Hobby Boards does this).The formula below will provide you with humidity.

```
local snum = "BD00000121AF4826"
local vdd = onewire.DS2438.readAtoD(portnum, true, snum)
local add = onewire.DS2438.readAtoD(portnum, false, snum)

print(vdd, add)
temp = onewire.DS2438.getTemperature(portnum, snum)

humid = (((add/vdd) - 0.16)/0.0062)/(1.0546 - 0.00216 * temp);
if(humid > 100)then
  humid = 100;
elseif(humid < 0) then
  humid = 0;
end
```

18.18 os

The os table is mostly the standard lua os functions with one addition, the File System Watcher.

18.18.1 newFileSystemWatcher

Creates a new FileSystemWatcher object.

Definition

```
FileSystemWatcher = os.newFileSystemWatcher( )
```

Name	Type	Description
FileSystemWatcher	FileSystemWatcher Object	FileSystemWatcher return value object.

Example

```
fw = os.newFileSystemWatcher()

fw:add("c:\\")
fw:callback( function( event, path )

  print("File System Watcher ", event, path)

end)
```

Related

FileSystemWatcher Object

Availability

Lua

18.18.2 FileSystemWatcher Object

File FileSystemWatcher object allows lua to receive notifications when a directory or file changes.

Methods

Return	Signature	Flags

© 2015 Promixis, LLC

	add(string)	
	remove(string)	
table of strings	files()	**[const]**
table of strings	paths()	**[const]**
	callback(function(string, string))	
	close()	

FileSystemWatcher::add(path)

Adds the path (either a directory or a file) to the list of watched items.

FileSystemWatcher::remove(path)

Removes the path from the list of watched items.

FileSystemWatcher::files()

Returns the list of watched files as a table.

FileSystemWatcher::paths()

Returns the list of watched folders as a table.

FileSystemWatcher::callback(function (event, path))

Sets the function to call when a change occurs. The event parameter will either be "fileChanged" or "directoryChanged". Note only one callback can be active at a time. Setting a new callback on the object will remove the old callback.

FileSystemWatcher::close()

Stops watching for changes and unregisters the callback.

Example

```
fw = os.newFileSystemWatcher()

fw:add("c:\\")
fw:callback( function( event, path )

  print("File System Watcher ", event, path)

end)
```

© 2015 Promixis, LLC

18.19 pir

The pir namespace allows you to control the PIR-1 and PIR-4 from Lua.

Functions

Return Value	Signature	Description
serials	listpir1()	returns a list of pir1 serials
serials	listpir4()	returns a list of pir4 serials
	transmit(serial, ccf, bitmask, repeats)	sends an IR signal
	stopTransmit(serial)	stops sending an IR signal
	keyboard(serial, key1, key2, key3, key4)	sets the keyboard emulation keys

pir.transmit(serial, ccf, bitmask, repeats)

serial should be the serial of the PIR (you can find it on the back of the PIR or use listpir1/listpir4. If you pass an empty string it will send out all PIR's attached to the machine.
bitmask depends on the hardware

PIR-1 Bitmask

1 = Back
2 = Front
3 = Both

PIR-4 Bitmask

1 = Port 1
2 = Port 2
4 = Port 3
8 = Port 4
Note that the PIR-4 will only send out one port at time.

Repeats

Repeats dictates how many times the repeat part of the CCF code will be sent. Note that if the CCF code only consists out of repeat codes and you pass 0 for repeat nothing will be sent. So it's generally a good idea to pass at least 1 here.

keyboard(serial, key1, key2, key3, key4)

© 2015 Promixis, LLC

PIR-1 units have a built-in keyboard emulator and can send one of 4 keys depending on which of the contacts was closed. To set what keypress, if any, should be generated use this function. Note that serial must match a connected PIR-1 otherwise this function will fail.

Key	Code	Key	Code
a	4	Return	40
b	5	Escape	41
c	6	Backspace	42
d	7	Tab	43
e	8	Space	44
f	9	-	45
g	10	=	46
h	11	[	47
i	12	]	48
j	13	\	49
k	14	;	51
l	15	'	52
m	16	`	53
n	17	,	54
o	18	.	55
p	19	/	56
q	20	Capslock	57
r	21	F1	58
s	22	F2	59
t	23	F3	60
u	24	F4	61
v	25	F5	62
w	26	F6	63
x	27	F7	64
y	28	F8	65
z	29	F9	66
1	30	F10	67
2	31	F11	68
3	32	F12	69
4	33	Right Arrow	79
5	34	Left Arrow	80

© 2015 Promixis, LLC

6	35	Down Arrow	81
7	36	Up Arrow	82
8	37	Num Lock	83
9	38	Page Up	75
0	39	Page Down	78

This is only an excerpt of the scan code table. The full version can be found here (http://download.microsoft.com/download/1/6/1/161ba512-40e2-4cc9-843a-923143f3456c/translate.pdf)

Setting the key values to 0 will suppress any key presses from being generated.

Example

```
local pir1s = pir.listpir1()

table.print(pir1s)
for i,serial in pairs(pir1s) do
    print(serial, pir.keyboard(serial, 0,0,0,0))
end
```

18.20 plugin

It is possible to build fully function plugins using Lua and Javascript. The backend code is written in Lua and has full access to the lua state. The front-end is written in Javascript. There are a few examples on how to do this in the Lua directory.

Plugin Description files

To define a script plugin you need to create a plugin description file. These have the extension "plugin". For example the file UPB/upb.plugin. These files are parsed using Lua but do not contain the full lua state. There are a few fields that should be in there.

name

The name of the plugin

description

The description of the plugin

backEndComponentManager

This is a boolean that indicates if the backend has a component manager.

backEndScriptName

The name of the script that contains the backend code.

frontEndComponentManager

This is a boolean indicating if the front-end has a component manager

frontEndIncludes

This is an array of strings that list the javascript files that should be loaded before the front-end script. This allows you to keep your code organized over several files instead of piling everything into one javascript file.

frontEndScriptName

This is the filename of the script containing the front-end script, relative to the lua directory.

id

The ID of the plugin. Make sure you do not use duplicate IDs.

Front end scripts

For now we'll refer you to the pio1\plugin.js and UPB\plugin.js files as examples of how to do this. The examples use .ui files, these can be created with the QtCreator.

Back end scripts

For now we'll refer you to the pio1\plugin.lua and UPB\plugin.lua files as examples of how to do this.

18.21 Promixis

18.21.1 Event

18.21.1.1 modifiers

These are the constants that go with events.

Name	Description
EVENT_MOD_NONE	No key modifier used.
EVENT_MOD_ON	This modifier should be used when the event goes into the ON or PRESSED state.
EVENT_MOD_OFF	This modifier should be used when the even goes into the OFF or UNPRESSED state.
EVENT_MOD_REPEAT	This modifier should be used when the ON event repeats.

Related

triggerEvent

Availability

Lua
Plugins
Javascript

18.21.2 EventNode

18.21.2.1 Modifiers

These are the constants that go with events.

Name	Description
EVENT_MOD_ON	This modifier should be used when the event goes into the ON or PRESSED state.
EVENT_MOD_OFF	This modifier should be used when the even goes into the OFF or UNPRESSED state.
EVENT_MOD_REPEAT	This modifier should be used when the ON event repeats.

Related

triggerEvent

Availability

Lua

© 2015 Promixis, LLC

Plugins
Javascript

18.21.3 IEmailSettings

18.21.3.1 ConnectionType

Constants for the email settings object.

Name	Description
CT_PLAIN	Plain connection
CT_SSL	SSL connection
CT_TLS	TLS connection

Related

Email Settings

Availability

Lua
Plugins
Javascript

18.21.4 IProxySettings

18.21.4.1 ProxyType

Constants used with the Proxy settings object.

Name	Description
PT_HTTPPROXY	HTTP Proxy
PT_NOPROXY	No Proxy
PT_SOCKS5	Socks 5 Proxy

Related

Proxy Settings

Availability

Lua
Plugins
Javascript

18.21.5 Transport

18.21.5.1 Connection

18.21.5.1.1 Type

The connection type for the transport system

Name	Description
CON_SERIAL	Serial connection
CON_TCP	TCP connection
CON_SSL	SSL connection (Secure connection)

Related

Connection Object

Availability

Lua

18.21.5.2 IConnectionCallback

18.21.5.2.1 Status

The status reported by the connection object.

Name	Description
CONNECTION_CLOSED	The connection was closed
CONNECTION_ESTABLISHED	The connection was established
CONNECTION_FAILED	The connection was not able to connect

Related

© 2015 Promixis, LLC

Connection Object

Availability

Lua

18.21.5.3 IParser

18.21.5.3.1 Type

Name	Parameters	Description
PARSER_LENGTH	length (number)	Length based parser.
PARSER_PASSTHROUGH	none	Passes data through as it comes in. This could be in single bytes or blocks of bytes.
PARSER_TERMINATED	single terminator string for example "\n"	Splits incoming data at each terminator without returning terminator.
PARSER_TERMINATED_LIST	array of strings	Splits incoming data based on a list of terminators and keeps the terminator on the data.
FIRST_BYTE_LENGTH	none	Splits the incoming data by using the first byte of the incoming data as the length field. The length does not include the first byte. For example 02 00 01 03 01 02 03 holds 2 messages Message 1: 00 01 Message 2: 01 02 03

Related

Connection Object

Availability

Lua

© 2015 Promixis, LLC

18.21.5.4 ITransactionCallback

18.21.5.4.1 Result

Name	Description
TX_KEEP	Keep the transaction but do not pass the data to the next transaction.
TX_CONTINUE	Pass the data to the next transaction.
TX_RECONNECT	Disconnect and reconnect the connection.
TX_REMOVE	Remove the transaction
TX_REQUEUE	Re-queue the transaction at the end of the transaction queue.
TX_REQUEUE_FIRST	Re-queue the transaction at the beginning of the transaction queue.
TX_RESET_TIMEOUT	Reset the transaction time. This is good for multi packet transactions.

These are flags and as such can be combined by the plus (+) operator.

Related

Connection Object
Transaction Object

Availability

Lua

18.21.5.5 SerialConnection

18.21.5.5.1 Flow Control

Name	Description
FLOW_HARDWARE	Hardware flow control
FLOW_SOFTWARE	Software flow control
FLOW_NONE	No flow control

Related

Serial Connection

© 2015 Promixis, LLC

Availability

Lua

18.21.5.5.2 StopBits

Name	Description
STOP_ONE	One stopbit.
STOP_ONEFIVE	One and a half stop bit.
STOP_TWO	Two stopbits.

Related

Serial Connection

Availability

Lua

18.21.5.5.3 Parity

Name	Description
PARITY_EVEN	Even parity.
PARITY_ODD	Odd parity.
PARITY_NONE	No parity.

Related

Serial Connection

Availability

Lua

18.21.6 Control

18.21.6.1 DType

The type of control this is.

Name	Description

BUTTON	A Button control with no value changes
CCF	A CCF control. This control will accept plain CCF or a JSON IR structure.
EDIT	An edit control DConfig will contain a "formatString". This could be for example "Temperature %1 F" will will then be displayed as "Temperature 85 F"
LABEL	A label not editable by user but changeable from code.
LIST	A List of options to choose from DConfig contains a list of key value pairs {"values": [["Off", "0"], ["50%", "50"], ["On", "100"]]}
RANGLE	A range selector DConfig will contain maximum, mininum and step configures the behaviour of this control.
TOGGLE	A Toggle Button
URL	Displays the webpage at url

Related

Control Object

Availability

Lua

18.21.7 Device

18.21.7.1 Status

The status reported by the device object.

Name	Description
STATUS_ERROR	The device has reported an error and is not working properly
STATUS_OK	The device is working properly
STATUS_UNKNOWN	The device is offline or disabled.

© 2015 Promixis, LLC

Related

Device Object

Availability

Lua

18.21.8 Log

The type of control this is.

Name	Description
OK	Not an error just a informative message.
WARNING	Something went wrong. It's been handled but you should know about it.
ERROR	Something went wrong. It was not automatically fixed but things can keep going. You should investigate.
CRITICAL	Something really bad went wrong. Whatever you tried to do did not work and might have more consequences.

Related

gir.log

Availability

Lua

18.22 publisher

Pubsub is a thread safe publish and subscriber class. This can be very useful to decouple classes from each other. Which is considered good programming practice.

```
publisher = publisher.new()
```

The publisher object has the following methods:

Return	Signature	Description

number	subscribe(callback)	subscribes the function to this publisher. Returns the ID of this subscription. This can be used to unsubscribe at a later date.
	unsubscribe(id)	Unsubscribe from the publisher.
	publish(arguments)	This function will call all the subscribers with the arguments pass.

Example

```
local publisher = require('publisher')

local p = publisher.new()

p:subscribe( function( aa )
print("Called 1!", aa)
end)

p:subscribe( function( aa )
print("Called 2!", aa)
end)

p:publish ( "cool" )
```

this will print

```
Thu May 2 15:21:12 2013 Called 1!   cool
Thu May 2 15:21:12 2013 Called 2!   cool
```

18.23 raspi

When Girder is running on a Raspberry Pi, you can modify the GPIO pins from Lua with these commands. The commands take pin numbers as input. To find the right pin you can consult the documentation available here:

http://elinux.org/RPi_BCM2835_GPIOs

To use these functions don't forget to add to the top of your script.

```
local raspi = require("raspi")
```

© 2015 Promixis, LLC

18.23.1 export

To be able to use the GPIO pins you must first export them.

Definition

```
result, error = raspi.export( pin )
```

Name	Type	Description
pin	number	The pin number to export
result	nil or true	Returns true if success or nil + error on error.
error	nil or string	The error string if applicable.

Example

```
print(raspi.export( 24 ))
```

Availability

Lua only on Raspberry Pi

18.23.2 unexport

If a GPIO pin is no longer needed unexport it.

Definition

```
result, error = raspi.unexport( pin )
```

Name	Type	Description
pin	number	The pin number to export
result	nil or true	Returns true if success or nil + error on error.
error	nil or string	The error string if applicable.

Example

```
print(raspi.unexport( 24 ))
```

Availability

Lua only on Raspberry Pi

18.23.3 direction

The GPIO pins can be configured to be input or output.

Definition

```
result, error = raspi.direction( pin, dirOut )
```

Name	Type	Description
pin	number	The pin number
dirOut	boolean	true to make pin output, false to make pin input.
result	nil or true	Returns true if success or nil + error on error.
error	nil or string	The error string if applicable.

Example

```
print(raspi.direction( 24, true ))
```

Availability

Lua only on Raspberry Pi

18.23.4 write

The GPIO pins can be set to logic level 0 or 1.

Definition

```
result, error = raspi.write( pin, value )
```

© 2015 Promixis, LLC

Name	Type	Description
pin	number	The pin number
value	number	0 or 1.
result	nil or true	Returns true if success or nil + error on error.
error	nil or string	The error string if applicable.

Example

```
print(raspi.write( 24, 1 ))
```

Availability

Lua only on Raspberry Pi

18.23.5 read

The GPIO pins can be logic level 0 or 1, use read to get the value.

Definition

```
result, error = raspi.read( pin )
```

Name	Type	Description
pin	number	The pin number
result	nil, 0 or 1	Returns 0 or 1 if success or nil + error on error.
error	nil or string	The error string if applicable.

Example

```
print(raspi.read( 24 ))
```

Availability

Lua only on Raspberry Pi

© 2015 Promixis, LLC

18.24 scheduler

The scheduler functionality allows you to schedule events at specific intervals defined by tasks. A scheduler consists of the main scheduler object with tasks attached to it. The scheduler defines the eventstring and event device. The tasks define when to trigger the eventstring.

18.24.1 create

Creates a new scheduler object.

Definition

```
scheduler = scheduler.create( eventstring, eventdevice, keepBeyondLua )
```

Name	Type	Description
scheduler	Scheduler Object	The newly created scheduler object
eventstring	string	The eventstring to trigger
eventdevice	number	The event device number.
keepBeyondLua	boolean	Set this to true if this scheduler should persist beyond a lua reset.

Example

```
local s = scheduler.create( "hello", 18, true )
```

Availability

Lua

18.24.2 getScheduler

Gets a scheduler object.

Definition

```
scheduler = scheduler.getScheduler( idOrUuid )
```

Name	Type	Description

© 2015 Promixis, LLC

scheduler	Scheduler Object	The scheduler object requested or nil if not found.
idOrUuid	number of string	Either the id or the uuid of the scheduler

Example

```
local s = scheduler.getScheduler( 1 )
```

Availability

Lua

18.24.3 getSchedulers

Lists all scheduler objects.

Definition

```
schedulers = scheduler.getSchedulers( )
```

Name	Type	Description
schedulers	table	Table of scheduler details indexed by scheduler id

Scheduler Details Record

Name	Type	Description
eventString	string	The eventstring for the scheduler
device	number	The event device number
id	number	The scheduler id
uuid	string	The scheduler uuid

Example

© 2015 Promixis, LLC

```
local schedulers = scheduler.getSchedulers( )

for id, schedule in pairs(schedulers) do
      print(schedule.eventString)
      print(schedule.device)
      print(schedule.uuid)
end
```

Availability

Lua

18.24.4 sunrise

Calculate the sunrise time.

Definition

```
sunrise = scheduler.sunrise( latitude, longitude, [date], [timezone] )
```

Name	Type	Description
sunrise	Date Object	Date and Time of sunrise.
date	Date Object	Optional date object specifying the day you wish to get the sunrise time for.
latitude	number	latitude to calculate for
longitude	number	longitude to calculate for
timezone	number	the timezone the lat/long is in.

Example

```
local sunrise = scheduler.sunrise( gir.settings().latitude,
gir.settings().longitude, date.now(),date.utcOffset( ) )
print(sunrise)
```

prints:
Fri Mar 1 06:44:00 2013

Availability

Lua

© 2015 Promixis, LLC

18.24.5 sunset

Calculate the sunset time.

Definition

```
sunset = scheduler.sunset( latitude, longitude, [date], [timezone] )
```

Name	Type	Description
sunset	Date Object	Date and Time of sunset.
date	Date Object	Optional date object specifying the day you wish to get the sunset time for.
latitude	number	latitude to calculate for
longitude	number	longitude to calculate for
timezone	number	the timezone the lat/long is in.

Example

```
local sunset = scheduler.sunset( gir.settings().latitude,
gir.settings().longitude, date.now(),date.utcOffset( ) )
print(sunset)
```

prints:
Fri Mar 1 06:44:00 2013

Availability

Lua

18.24.6 Scheduler Object

The schedules object:

Functions

Return	Signature	

boolean	start()	starts the scheduler
boolean	stop()	stops the scheduler
	destroy()	destroys the sched

© 2015 Promixis, LLC

		ulen
taskId	minuteTask(every, repeat, begin, end)	adds a minute task.
taskId	hourTask(every, minute, repeat, begin, end)	adds an hour task.
taskId	dayTask(every, hour, minute, repeat, begin, end)	adds a day task.
taskId	dayOfWeekTask(every, dow, hour, minute, repeat, begin, end)	ad

© 2015 Promixis, LLC

		c s a d a y o f w e e k t a s k .
taskId	dayOfMonthTask(every, dom, hour, minute, repeat, begin, end)	a c c s a d a y o f m o n t h t a s k .
taskId	sunsetTask(every, latitude, longitude, repeat, begin, end, offset, timezone)	a c c s a s u n n i s e t a

© 2015 Promixis, LLC

		sk.	
taskId	sunriseTask(every, latitude, longitude, repeat, begin, end, offset, timezone)	adds a sunset task.	
boolean	removeTask(taskId)	removes a task.	
boolean	clear()	removes all tasks.	
taskList	getTasks()	re	

© 2015 Promixis, LLC

		turns a table with all task properties.
task	getTask(taskId)	returns a table with al

		l properties for task.
boolean	setRandomizer(taskId, min, max, dist)	sets the randomizer for task.
boolean	keepBeyondLua()	returns i

		f this scheduler lives beyond lua.
boolean	isRunning()	returns if this scheduler i

		s running.
string	getEventString()	returns the event string.
number	getDevice()	returns the event devic

		e.
number	getID	returns the scheduler id.
string	getUUID	returns the scheduler uuid.

Task manipulation

Tasks are the parts of a scheduler that dictate when the scheduler triggers it's event. All the functions have a few parameters in common.

Name	Type	Description
every	number	Determines if the task should skip any of it's trigger points. For example. every = 1 means that each trigger point should actually trigger. every = 3 means that after one actual trigger, the next two trigger points are skipped.
repeat	number	how many times the task may trigger between begin and end times.
begin	Date Object	Date to start triggering from.
end	Date Object	Date to end triggering on.

Task Randomization

To prevent tasks from running at exactly the same interval setRandomizer can be used. The dist parameter selects the distribution. Currently we have:
- dist = 0, returns minimum at all times.
- dist = 1, returns a linear distribution of random values between min and max.

The min and max parameters are task offsets in minutes.

18.25 socket

The socket namespace provides various methods of communicating over tcp or udp. Full documentation can be found here: http://www.promixis.com/lua/luasocket

18.25.1 imap

IMAP is an extension by Promixis to the socket namespace. This class is fairly low level an understanding of the IMAP protocol will certainly help using this class.

Some resources on IMAP can be found here:

- http://tools.ietf.org/html/rfc3501
- http://www.skytale.net/blog/archives/23-Manual-IMAP.html
- http://www.imapwiki.org/ClientImplementation
- http://networking.ringofsaturn.com/Protocols/imap.php

© 2015 Promixis, LLC

Methods

Return	Signature	Description
IMAP object	new (server, port, sslProtocol, timeout)	constructor
result, err	connect()	connect to server
result, err	login(username, password)	login to the server
result, err	logout()	logout from the server
result, boxes	getMailBoxes()	Gets a list of mailboxes
result, imapResult	examine(mailBox)	Runs an EXAMINE command on the mailbox
result, imapResult	select(mailBox)	Selects the mailbox as current, required for functions below
result, messageIds	unseen()	Gets a list of ids for messages that have not yet been read.
result, message	fetchMessage(messageId, peek)	Gets the message.
result, err	addFlags(messageId, flags)	Adds flags to the message flag list
result, err	removeFlags(messageId, flags)	Removes flags from the message flag list
result, imapResult	query(query)	Advanced function to run your own IMAP queries.

new

Creates the new object

```
imap = socket.imap.new( server, port, sslProtocol, timeout )
```

Name	Type	Description
imap	IMAP Object	The IMAP object
server	string	server name
port	number	port number to connect to
sslProtocol	string	ssl to use or empty (sslv3, tlsv1 or sslv23)

© 2015 Promixis, LLC

timeout	number	timeout to use for all operations

connect

Connects the object to the server. Returns true on success.

```
result, err = imap:connect()
```

Name	Type	Description
result	boolean	True on success, nil on error
err	string	Reason for error

login

logs into the server

```
result, err = imap:login( username, password )
```

Name	Type	Description
result	boolean	True on success, nil on error
err	string	Reason for error
username	string	username for login
password	string	password for login

logout

logs out from the server

```
result, err = imap:logout( )
```

Name	Type	Description
result	boolean	True on success, nil on error
err	string	Reason for error

getMailBoxes

gets a list of available mailboxes

```
result, mailboxes = imap:getMailBoxes( )
```

© 2015 Promixis, LLC

Name	Type	Description
result	boolean	True on success, nil on error
mailboxes	string or table of mailbox details (mailbox.flags, mailbox.path, mailbox.name)	Reason for error or mailbox details

examine

Performs the IMAP EXAMINE command on the mailbox.

```
result, msg = imap:examine( mailbox )
```

Name	Type	Description
result	boolean	True on success, nil on error
msg	string or table of IMAP results see http://tools.ietf.org/html/rfc3501#section-6.3.2	Reason for error or details

select

Sets the mailbox as the currently selected box. This is necessary before you can do unseen fetchMessage, addFlags or removeFlags

```
result, msg = imap:select( mailbox )
```

Name	Type	Description
result	boolean	True on success, nil on error
msg	string or table of IMAP results see http://tools.ietf.org/html/rfc3501#section-6.3.1	Reason for error or details

unseen

gets a list of messages that have not yet been marked unseen (aka UNREAD)

```
result, msg = imap:unseen( )
```

Name	Type	Description

© 2015 Promixis, LLC

result	boolean	True on success, nil on error
msg	string or table or message ids	Reason for error or list of message ids that can be passed to fetchMessage

fetchMessage

Fetches a message from the server

```
result, message = imap:fetchMessage( messageId, peek )
```

Name	Type	Description
result	boolean	True on success, nil on error
message	string	reason for error or message header + body
messageId	number	id of message obtained from unseen
peek	boolean	if set to true the message will not be marked as \Seen upon fetch

addFlags, removeFlags

Adjusts the flags on a message.

```
result, err = imap:addFlags( messageId, flags )
```

Name	Type	Description
result	boolean	True on success, nil on error
err	string	reason for error
messageId	number	id of message obtained from unseen
flags	string	Message flags to add or remove see http://tools.ietf.org/html/rfc3501#section-2.3.2

Running custom requests

© 2015 Promixis, LLC

To run custom requests study the imap.lua file and the use the imap:request(requestString) function to perform any operation you like.

Example

The following example show how you can get the oldest unseen message from the gmail IMAP server. Depending on the content of the message you could trigger events. This would allow you to make Girder do certain actions from email.

```
imap = require('socket.imap')
i = imap.new( "imap.gmail.com", 993, "tlsv1", 1000)

if not i:connect() then
      print("Failed to connected")
      return
end

if not i:login("youremail@gmail.com", "yourpassword") then
      print("Login failed.")
      return
end

-- Select your desired mailbox
if not i:select("Inbox") then
      print("Could not select mailbox")
      return
end

-- get a list of unseen messages
local status, unseenMessageList = i:unseen()
if not status then
      print("Could not get list of unseen messages")
      return
end

if #unseenMessageList == 0 then
      print("No unseen messages")
      return
end

-- get the first unseen message marking it as seen.
local status, message = i:fetchMessage( unseenMessageList[ 1 ], false )
if not status then
      print("Could not get message")
      return
end


print( message )
```

```
i:deinit()
i = nil
```

18.26 speech

Interfaces with the text to speech engine.

18.26.1 speak

This function will speak text.

Definition

```
speech.speak( text [, voice] )
```

Name	Type	Description
text	string	The text to speak
voice	string	The voice to use (can be nil for default voice)

Example

```
speech.speak('Hello I am Girder insert Bender')
```

Availability

Lua
Actions

18.26.2 listVoices

This function will speak text.

Definition

```
voices = speech.listVoices( )
```

Name	Type	Description
voices	table of strings	Table filled with available voices.

Example

```
local voices = speech.listVoices()
table.print(voices)
```

Availability

Lua

18.26.3 setVolume

This function will set the volume of the voice.

Definition

```
success = speech.setVolume( vol )
```

Name	Type	Description
success	boolean	return value, indicating success
vol	integer	Volume level, 0 - 100. Where 0 is silent and 100 is max.

Example

```
speech.setVolume(50)
```

Availability

Lua

© 2015 Promixis, LLC

18.26.4 listOutputs

This function will list the available outputs.

Definition

```
outputs = speech.listOutputs( )
```

Name	Type	Description
outputs	table of strings	Table filled with available output devices.

Example

```
local outputs = speech.listOutputs()
table.print(outputs)
```

Availability

Lua

18.26.5 setOutput

This function will set the output device.

Definition

```
success = speech.setOutput( device )
```

Name	Type	Description
success	boolean	return value, indicating success
device	string	Name of device selected from list returned by listOutputs.

Example

```
speech.setOutputs("headset") -- adjust headset to your actual hardware
name as obtained by listOutputs.
```

Availability

Lua

18.27 sql

the SQL namespace documentation can be found here http://www.promixis.com/lua/luasql

The implemented drivers are sqlite and ODBC. ODBC is only available on Windows.

18.28 ssl

The SSL namespace is an extension to the socket namespace which provides SSL (sslv3, tlsv1 or sslv23) to the socket library.

Client example:

```
require("socket")
require("ssl")

local params = {
  mode = "client",
  protocol = "tlsv1",
  key = "/etc/certs/clientkey.pem",
  certificate = "/etc/certs/client.pem",
  cafile = "/etc/certs/CA.pem",
  verify = "peer",
  options = "all",
}

local conn = socket.tcp()
conn:connect("127.0.0.1", 8888)

-- TLS/SSL initialization
conn = ssl.wrap(conn, params)
conn:dohandshake()
--
print(conn:receive("*l"))
conn:close()
```

The key, certificate, cafile fields of params are optional for clients. After dohandshake you can use the socket as per lua sockets documentation.

© 2015 Promixis, LLC

Server example

```
require("socket")
require("ssl")

local params = {
  mode = "server",
  protocol = "tlsv1",
  key = "/etc/certs/serverkey.pem",
  certificate = "/etc/certs/server.pem",
  cafile = "/etc/certs/CA.pem",
  verify = {"peer", "fail_if_no_peer_cert"},
  options = {"all", "no_sslv2"},
  ciphers = "ALL:!ADH:@STRENGTH",
}

local server = socket.tcp()
server:bind("127.0.0.1", 8888)
server:listen()
local conn = server:accept()

-- TLS/SSL initialization
conn = ssl.wrap(conn, params)
conn:dohandshake()
--
conn:send("one line\n")
conn:close()
```

18.29 string

Girder adds a few extra functions to the string library besides the usual Lua String functions.

© 2015 Promixis, LLC

18.29.1 latin1ToUtf8

Definition

```
utf8Str = string.latin1ToUtf8 ( latin8Str )
```

Name	Type	Description
latin8Str	string	Latin-1 (ISO8859-1) encoded string.
utf8Str	string	UTF-8 encoded string.

Example

```
local s = 'René'
print(s)
s = string.latin1ToUtf8(s)
print(s)
```

This will print:
Tue Dec 31 14:18:44 2013   Ren?
Tue Dec 31 14:18:44 2013   René

18.29.2 local8BitToUtf8

Definition

```
utf8Str = string.local8BitToUtf8 ( local8BitStr )
```

Name	Type	Description
local8BitStr	string	Uses encoding most suitable for the current locale.
utf8Str	string	UTF-8 encoded string.

Example

```
local s = 'René'
print(s)
s = string.local8BitToUtf8(s)
print(s)
```

© 2015 Promixis, LLC

This will print:
Tue Dec 31 14:18:44 2013   Ren?
Tue Dec 31 14:18:44 2013   René

18.29.3 ltrim

Definition

```
trimmedStr = string.ltrim ( str )
```

Name	Type	Description
str	string	String with possible white space at beginning and end
trimmedStr	string	String without any white space at beginning

Example

```
print( string.ltrim("     hello     world     ") )
```

This will print "hello   world   "

18.29.4 rtrim

Definition

```
trimmedStr = string.rtrim ( str )
```

Name	Type	Description
str	string	String with possible white space at beginning and end
trimmedStr	string	String without any white space at end

Example

```
print( string.trim("     hello     world     ") )
```

This will print "   hello   world"

18.29.5 split

Definition

```
results = string.split ( str, pattern, resultsTable )
```

Name	Type	Description
results	table of strings	A table of strings
pattern	string	Pattern upon which to split can be regexp
resultsTable	table (optional)	A table to hold the result, will be passed back as results

```
table.print( string.split("hello world there", " ") )
```

This will print

```
{ -- #0
    [1] = "hello",
    [2] = "world",
    [3] = "there",
} -- #0
```

18.29.6 trim

Definition

```
trimmedStr = string.trim ( str )
```

Name	Type	Description
str	string	String with possible white space at beginning and end
trimmedStr	string	String without any white space at beginning or end

Example

```
print( string.trim("     hello     world      ") )
```

© 2015 Promixis, LLC

This will print "`hello    world`"

18.30 table

Girder adds a few extra functions to the string library besides the usual Lua Table functions.

18.30.1 copy

Make a deep copy of a table

Definition

```
copiedT = table.copy( t )
```

Name	Type	Description
t	table	Table to copy
copiedT	table	Deep copy of t

18.30.2 isEmpty

Check if a table has no keys or indexes.

Definition

```
empty = table.isEmpty( t )
```

Name	Type	Description
t	table	Table to check
empty	boolean	true if table was completely empty.

18.30.3 print

Definition

```
table.print( t )
```

Name	Type	Description
t	table	Table to print

18.30.4 tostring

Definition

```
str = table.tostring( t )
```

Name	Type	Description
str	string	Table displayed as string
t	table	Table to format as string

Example

```
local t = {
        a=10;
        b=20;
        c= {
                sub1= 2;
                sub2= 3
        }
}

print ( table.tostring( t ) )
```

This will print:

```
{ -- #0
   ["a"] = 10,
   ["c"] = { -- #1
      ["sub2"] = 3,
      ["sub1"] = 2,
   } -- #1,
```

© 2015 Promixis, LLC

```
    ["b"] = 20,
} -- #0
```

18.31 thread

Threading is a tricky subject and should be consider an advanced feature of Girder. It is very easy to hang Girder up by improper use of the functions given below. Consider yourself warned!

Functions

Returns	Signature	Description
	newthread(function, parameters)	creates and runs a new thread.
Mutex Object	newmutex()	creates a new mutex.
Condition Object	newcond()	creates a new condition.

Example

```
thread.newthread( function( a,b )
    print(a,b)
end, {"hello", "there" })
```

This example simply prints "hello there". But it does so from a new thread. The next example is a bit more complicated. It protects a shared variable called "result" with a mutex and signals the main thread about a change to it by using a condition.

```
local m = thread.newmutex()
local c = thread.newcond()
local result = 0

thread.newthread( function( mutex, cond )

    mutex:lock()
    for i=0, 50000 do
        result =  i
    end

    cond:broadcast()
    mutex:unlock()

end, { m,c })


m:lock()
```

```
c:wait(m)
print("Thread result:" , result)
m:unlock()
```

As you'll be able to see is that it will print "Thread result: 50000" correctly.

18.31.1 Mutex

Mutex object

Functions

Returns	Signature	Description
	lock()	locks the mutex
	unlock()	unlocks the mutex

Example

See thread

18.31.2 Condition

Condition object allows you to signal other threads.

Functions

Returns	Signature	Description
	wait(Mutex)	wait for the condition to become signalled.
	signal()	signals one waiting thread
	broadcast()	signals all waiting threads

Example

See thread

18.32 timer

© 2015 Promixis, LLC

18.32.1 new

timer.new creates a new timer object.

Definition

```
timerObj = timer.new( timeout, callback )
```

Name	Type	Description
timerObj	Timer Object	The timer object.
timeout	number	Number of miliseconds till callback is called.
callback	function(Timer Object)	The function to call.

Example

```
local counter = 0
local timerObj = timer.new(500, function(t)
  counter = counter + 1
  if (counter>= 50) then
        t:deinit()
  end
  print("hello", counter)
end)

timerObj:start()
```

This prints "hello 1" through "hello 20" 500 milliseconds apart into the Lua console.

Related

Timer Object

Availability

Lua

18.32.2 Timer Object

The timer object.

Methods

© 2015 Promixis, LLC

Return	Signature	Flags
	start()	
	stop()	
	deinit()	

Timer::start()

Starts the timer. If already running the timer will be restarted.

Timer::stop()

Stops the timer.

Timer::deinit()

Stops and destroys the timer. After this call the timer object is invalid. Do not keep a reference to it any longer.

Example

```
local counter = 0
local timerObj = timer.new(500, function(t)
  counter = counter + 1
  if (counter>= 50) then
        t:deinit()
  end
  print("hello", counter)
end)

timerObj:start()
```

18.33 transport

Girder 6 like Girder 5 has a wire independent script based driver infrastructure built in. Driver independent means that no matter if the hardware is connected through RS232, RS485 or Ethernet if the hardware API is the same you'll only need to develop the driver once. Moving to a different wire is only a matter of changing the initialization parameters. Since the transport library is script based it allows for driver development without the need for a C++ compiler by using the built-in Lua scripting. The implementation is slightly different from the Girder 5 transport API. This was done so that drivers can run on both Girder's and PEAC's scripting engine. For most applications these drivers can be very simple. More complex hardware APIs will take a bit more effort.

Structure

© 2015 Promixis, LLC

The transport system consist of out 4 different parts. These are, the parser, the connection, the transaction queue, transactions and the callbacks into your code.

The connection

The connection object abstracts the actual wire protocol (RS232, RS485, ethernet) away from your code. You get 3 events. "Connected" and "Disconnected" and "Failed". The connection automatically tries to re-establish a broken connection.

The Parser

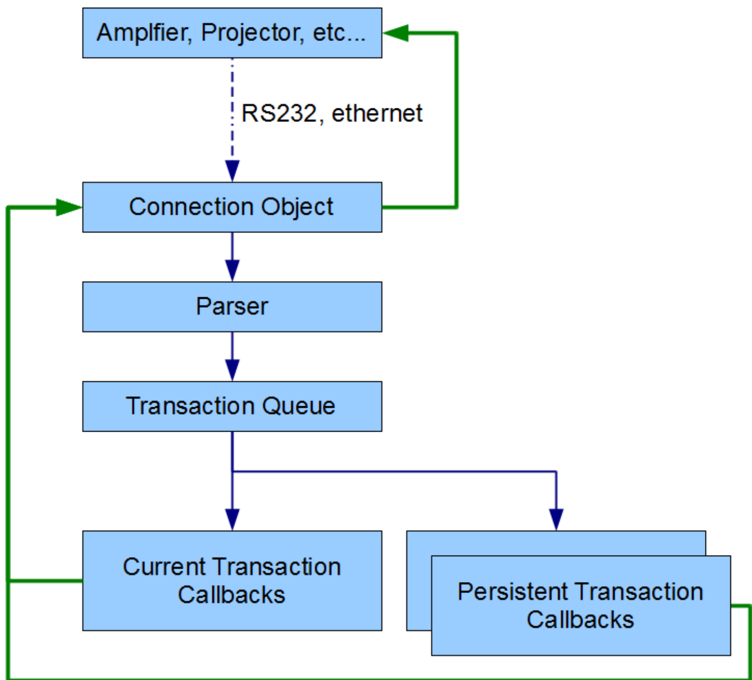
Protocols will all have some way of delimiting different messages form each other. Often this is based on a special marker like and end-line ("\n") or based on number of characters. The parser takes this worry from your hands and snips in the incoming data into packets.

The queue

This object holds the transaction that get queued up for processing. Some transaction do not need to wait for a response before the next transaction is processed while others list sit and listen to the packets coming by. By far the most common is the transaction that sends some data and waits for a response.

The transaction

Communications with hardware is usually request-response based. We call a pair of these a transaction. Typical transaction send a command to the hardware and wait a certain amount of time for the response from the hardware. If the response does not arrive on time it is sent again if so desired. Other transaction simply sit around listening for incoming data, these are called persistent transactions. While others again and fire and forget sending a command but not expecting any response. Transactions have 3 different events they generate for your scripts. "OnData", "OnSent" and "OnTimeout". The first is called when a response arrives, the second when the data for this transaction has been sent and the last if no response is received before the timeout. Note that multiple transaction can be queued up and sit in the queue until the connection is established.



At any one time there can only be one 'current' transaction. That is a transaction that blocks sending of other transactions until it has either timed-out or received a response. There can be any number of transactions in the "persistent" queue. What happens after a timeout or data received events depends completely on the script and the return value it sends back to it's transaction.

The transport table only has a few functions. The new call, udp and resolve. The new function creates the connection object for you.

```
connection = transport.new( connectionType, ... )
```

There are 4 different transport types

Serial connection

The serial transport uses a RS232 or RS485 connection using a COM port on your computer. Since comports are rare on computers these days this most likely will be a USB to RS232 dongle.

```
serialConnection =
transport.new( Promixis.Transport.Connection.Type.CON_SERIAL, serialPort,
callback )
```

Name	Type	Description

© 2015 Promixis, LLC

serialPort	string	The name of the serial port. For example on Windows this could be "COM1" or "COM2". On Linux this could be "\dev \ttyS1"
callback	function (event, reason)	The callback for the connection object. Reason is filled with a string if the event has any additional information. Typically this will be on a failed connection attempt.

TCP Connection

The TCP connection is a reliable connections in the sense that any loss of data should be corrected by the underlying system. Data should always arrive and if it does not the system should generate a disconnected event.

```
tcpConnection =
transport.new( Promixis.Transport.Connection.Type.CON_TCP, hostname,
port, callback )
```

Name	Type	Description
hostname	string	The hostname of the device to connect to. This can be a IPV4 or IPV6 ip address or a hostname.
port	number	The port number to connect to. Read the API documentation of the hardware to find this.
callback	function (event, reason)	The callback for the connection object. Reason is filled with a string if the event has any additional information. Typically this will be on a failed connection attempt.

SSL Connection

The SSL connection has most of the same characteristics of the TCP connection with the added benefit that it encrypts the data traveling over the wire. Note that currently we do not have a way of verifying the remote certificate and this could in theory lead to man in

the middle attacks.

```
tcpConnection =
transport.new( Promixis.Transport.Connection.Type.CON_SSL, hostname,
port, callback )
```

See the TCP connection for parameter details.

18.33.1 Connection Object

The connection object holds the connection to the remote end being controlled. For example an amplifier. It's created by the transport.new function. It automatically tries to reconnect if the connection has dropped and dishes out data in transaction based packets. The TCP Object, SSL Object and Serial Object subclass this object.

Methods

Return	Signature	Description
	connect()	Connects to the remote end.
	close()	Disconnect from the remote end.
	reconnect(timeout)	Disconnects and reconnects to the remote end.
boolean, string	parser(parserType, opt...)	Sets the parser to the type specified. All data in the old parser is transferred over.
transaction	newTransaction(sent, received, timeout)	Create a new transaction with 3 callback functions.
	send(transaction, boolean)	Queues a transaction.

Connection::connect()

Attempts to establish a connection to the remote end. If successful the connection callback is called on the Connection Object with status parameter "Connected".

Connection::close()

Closes the connection to the remote end. If successful (if not closed already) the connection callback is called with the status "Disconnected".

Connection::Reconnect()

Closes and reopens the connection calling the connection callback as appropriate.

Connection::Parser(parserType, opt...)

When data comes in, Girder can chop it into blocks of data before it reaches your code. For example if you know that all responses are delimited by an endline (\n) you can set up a terminated parser with parameter "\n". Each parser has different options. See the parser type page for the details.

Connection::newTransaction(sent, received, timeout)

Creates a new transaction object with the callbacks as specified in the call. The signatures of the callback functions are:

Return	Signature	Description
	sent()	called when data is put on the wire.
ITransactionCallback::Result	received(string)	called when data arrived and was parsed.
ITransactionCallback::Result	timeout( )	called when the transaction times out.

Send(transaction, queueFirst)

Queues the transaction up for sending out. If the connection was not yet established it is simply queued up until the connection is open. If queueFirst is true the transaction is queued at the start of the transaction queue.

Related

Serial Connection
TCP Connection
SSL Connection

18.33.1.1 Serial Connection Object

The Serial Connection Object inherits all methods from the Connection Object.

Methods

Return	Signature	Description
boolean	baud (number)	Sets the baud rate.
boolean	flow (flowControl)	Sets the flow control.
boolean	parity (parityType)	Sets the parity.
boolean	stopBits (stopBitsType)	Sets the number of stop bits.
boolean	characterSize (number)	Sets the number of bits in a character, allowed values are 5,6,7 and 8.

© 2015 Promixis, LLC

SerialConnection::baud (number)

Sets the baud rate for the connection. Call before actually connecting.

SerialConnection::flow (flowControl)

Sets the flow control for the connection.

SerialConnection::parity(parityType)

Sets the parity for the connection.

SerialConnection::stopBits(stopBitsType)

Sets the number of stop bits for the connection.

SerialConnection::characterSize(number)

Sets the number of bits in a character.

Related

Connection Object

18.33.1.2 SSL Connection

The SSL Connection object inherits all the methods from the Connection Object. It exposes no additional methods.

Related

Connection Object

18.33.1.3 TCP Connection

The TCP Connection object inherits all the methods from the Connection Object. It exposes no additional methods.

Related

Connection Object

18.33.2 Transaction Object

The transaction object holds all data related to the transaction. Note that lua holds the ownership over this object so make sure it's not garbage collected before the transaction finished or you'll have a hard to find bug in your code.

© 2015 Promixis, LLC

Methods

Return	Signature	Description
number	timeout(number)	Sets the timeout in milliseconds.
string	data (string)	Sets the data to send.
boolean	persistent(boolean)	Sets the persistent flag
	removeCallbacks()	Removes the internal hooks to the callback functions to prevent cyclic references which could prevent garbage collection of the transaction object and eventually lead to an out of memory error on Girder.

TransactionObject::timeout([number])

Set or query the timeout for this object. You can reset the timeout from the timeout callback or the data arrived callback with the TX_RESET_TIMEOUT result status. (Default = 0)

TransactionObject::data([string])

Sets or queries the data to send

TransactionObject::persistent ([boolean])

Sets or queries the persistent flag. This flag determines if a transaction goes into the persistent queue or not. (Default = false)

TransactionObject::removeCallbacks ( )

Removes the callbacks for this transaction. After this call the callsbacks will no longer be called, make sure you pass TX_REMOVE in the result set.

Related

Connection Object

18.33.3 UDP Connections

UDP Connections are different from the TCP, SSL and Serial connections. The main difference is that UDP is connectionless and that it's not stream oriented. Data always arrives in blocks or not at all. Thus there is no need for a parser as with the other connections. This means the API is a little different.

© 2015 Promixis, LLC

To create a UDP connection use:

```
connection = transport.udp( callback )
```

Callback is a lua function that will receive the data, address and port for the remote device.

```
function callback( data, address, port )
      print(data,address,port)
end
```

the UDP connection object has a few methods.

Return	Signature	Description
number	send(data, address, port)	sends the data to the remote device
	close	closes the socket
boolean	listen(port)	listens on the port for data.
boolean	joinMulticastGroup(group)	joins a multicast group. Note this must be called after the socket is listening.

please note that address should be an IP address, not a hostname. You can get the IP address by using transport.resolve.

Example

The example below is based upon the PIO-1 broadcast presence API. Basically the PIO-1's send out a broadcast UDP packet every so often on port 5998. You can also request the PIO-1's to announce themselves. This is done by sending a specially crafted broadcast message to port 5999.

```
function callback( connection, data, address, port )
      print(data,address,port)
end

pioListener = transport.udp( function ( data, address, port )
      callback(c, data, address, port)
end)

print(pioListener:listen( 5998 ))

bp = 'PIO-1\002'
pioListener:send(bp, '255.255.255.255', 5999)
```

© 2015 Promixis, LLC

18.33.4 resolve

transport.resolve allows you to resolve hostnames into IP addresses.

```
transport.resolve( hostname, callback )
```

The callback function will receive a table with ip addresses for the requested hostname or nil if there was an error resolving.

Example

```
transport.resolve("google.com", function ( ips )

 if not ips then
     print("No IP found")
     return
 end

 for i,ip in ipairs(ips) do
   print(ip)
 end

end)
```

This function printed

Thu May 2 14:28:11 2013   74.125.229.227
Thu May 2 14:28:11 2013   74.125.229.233
Thu May 2 14:28:11 2013   74.125.229.232
Thu May 2 14:28:11 2013   74.125.229.226
Thu May 2 14:28:11 2013   74.125.229.229
Thu May 2 14:28:11 2013   74.125.229.231
Thu May 2 14:28:11 2013   74.125.229.230
Thu May 2 14:28:11 2013   74.125.229.228
Thu May 2 14:28:11 2013   74.125.229.224
Thu May 2 14:28:11 2013   74.125.229.225
Thu May 2 14:28:11 2013   74.125.229.238

18.33.5 PIO-1 Example

Let's put all the stuff from the previous pages together and actually develop a transport class! We'll build upon the Promixis PIO-1 hardware. This is a network connected multi purpose input output device. For example it has 4 IR outputs that you can use to control a TV, DVD player, cable box or anything using an IR remote. On top of that it has 3 relay outputs for things like door openers and automatic locks. The PIO-1 also has 2 serial input/outputs.

© 2015 Promixis, LLC

You can find the API documentation here.

First off we see that the PIO-1 employs an easy to use text based interface terminated with an end line character "\n". Next we see that the PIO-1 requires a password to be sent before it allows you to change anything. But first things first. Let's create a lua file in the examples folder called "pio1a.lua". We'll create this new file using the class structure of Lua.

```
local Base = require('Class') -- the base class for this object.

-- Includes that we'll need
local Promixis = Promixis
local transport = require('transport')
local print = print
local timer = require("timer")
local table = require("table")
local ipairs = ipairs

module (...)

Base:subclass( _M)

function init ( self, address, port, password )

	Base.init(self)

	self.password = password
	self.address = address
	self.port = port

	if not self.port then
		self.port = 6000
	end
	if not self.password then
		self.password = "cookie"
	end

end

function deinit( self )
	Base.deinit(self)

end
```

The code above will provide the framework where all the functionality will be placed. Note that the code above doesn't deal with the transport functionality yet. All it does is store the address and port information and provide the class framework. Now let's add the connect function. It will create a connection object and setup it's parser and callback.

```
local connectionCallback = function(self, event, reason )
```

© 2015 Promixis, LLC

```
        print(event,reason)
end

function connect( self )

        self.connection = transport.new
( Promixis.Transport.Connection.Type.CON_TCP, self.address, self.port,
function ( event, reason )
                connectionCallback(self, event,reason)
        end)


self.connection
:parser( Promixis.Transport.IParser.Type.PARSER_TERMINATED, "\r" )
        self.connection:connect()

end
```

Place this code above the init function. The first line we should look at is the transport.new line. This create a TCP connection object connecting to address self.address and port self.port. It also supplies the callback for the connection events. This will allow use to handle connecting, disconnect and connect failure events.

To actually use this code we'll create a script action on the Girder tree with this code:

```
local PIO1a = require("pio.pio1a")

local address = '192.168.1.107'

pio1 = PIO1a.new( address )
pio1:connect()
```

Running this if the IP address is correct will print a single lone "0". Looking up the status codes we find this means Promixis.Transport.IConnectionCallback.Status.CONNECTION_ESTABLISHED. So great! We have a connected object. Now we'll need to send some data over the link. This is done using transactions.

sendCCF

Let's implement the sendccf command of the PIO-1. The API states this to be sendccf bitmask,repeats,ccf\n Easy enough! The PIO-1 will respond with ok if the command is accepted and irdone when the IR code has been completed.

```
function sendCCF ( self, bitmask, repeats, ccf, callback )

        if not self.connection then
                return false
        end
```

```
        local retries = 0
        local devices = {}

        local tx = self.connection:newTransaction(
                function()
                        -- sent
                        print("sendIR / Data was sent...")
                end,
                function( data )
                        -- data
                        print("sendIR / Received: ", data)

                        if data == "irdone" then
                                callback( true )
                        elseif data == "ok" then
                                print("CCF code sending...")
                                return TXR.TX_RESET_TIMEOUT
                        elseif data == "error 2,0" then
                                print("PIO was busy try again")
                                return TXR.TX_REQUEUE
                        else
                                print("Error could not send")
                                callback(false)
                                return TXR.TX_REMOVE
                        end


                end,
                function ()
                        -- timeout
                        retries = retries + 1
                        if retries > 5 then
                                print("sendIR / Failed to get devices")
                                callback( false )
                                return TXR.TX_REMOVE
                        else
                                print("sendIR / Retrying...")
                                return TXR.TX_REQUEUE_FIRST
                        end
                end
        )

        tx:timeout(5000)
        tx:data("sendccf " .. bitmask .. "," .. repeats .. "," .. ccf ..
"\n")
        tx:persistent(false)

        self.connection:send(tx, true)
```

© 2015 Promixis, LLC

```
end
```

Easy as that! It's pretty much the same code as before with one significant change. When the PIO-1 returns "ok" to signal acceptance of IR code the transaction timeout is reset by returning `TXR.TX_RESET_TIMEOUT`.
To call this code simply add this to a lua scripting action.

```
pio1:sendCCF(8,20,'0000 006E 0022 0022 0156 00A9 0015 0012 0015 003F 0015
003F 0015 003F 0015 0012 0015 003F 0015 003F 0015 0012 0015 003F 0015
0012 0015 0012 0015 0012 0015 003F 0015 0012 0015 0012 0015 003F 0015
0012 0015 003F 0015 0015 0015 0012 0015 0015 0015 0012 0015 0015 0015
0015 0015 003F 0015 0015 0015 003F 0015 003F 0015 003F 0015 003F 0015
003F 0015 003F 0015 081F 0156 00A9 0015 0015 0015 003F 0015 003F 0015
003F 0015 0015 0015 003F 0015 003F 0015 0015 0015 003F 0015 0015 0015
0015 0015 0015 0015 003F 0015 0015 0015 0015 0015 003F 0015 0015 0015
003F 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015
003F 0015 0015 0015 003F 0015 003F 0015 003F 0015 003F 0015 003F 0015
003F 0015 081F', function( success)
    print("IR sent:", success)
end)
```

Persistent transactions

The PIR-1 supports asynchronous notifications of incoming IR codes or state changes on the digital pins. Let's add a callback for the IR code notification.

```
function listenForIR ( self, callback )

      if not self.connection then
            return false
      end

      local tx = self.connection:newTransaction(
            function()
            end,
            function( data )

                  if  string.find(data, "^nec") or
                        string.find(data, "^rc5") or
                        string.find(data, "^rc6") or
                        string.find(data, "^sirc")  then
                        callback(data)
                        return TXR.TX_KEEP
                  else
                        return TXR.TX_CONTINUE
                  end
```

```
            end,
            function ()
            end
      )

      tx:persistent(true)
      self.connection:send(tx, true)

end
```

This again looks similar to the previous functions. However we now do not need to set the timeout nor the data to send and set persistent to true. You can also see two new transaction return constants. TX_KEEP and TX_CONTINUE. TX_KEEP means keep the transaction in the list but do not pass the data to other transactions. TX_CONTINUE means keep the transaction yet pass the data to other transactions as well.

To use this code modify your startup lua action as follows:

```
local PIO1a = require("examples.pio1a")

local address = '192.168.1.107'

pio1 = PIO1a.new( address )
pio1:connect()
pio1:sendCCF(8,20,'0000 006E 0022 0022 0156 00A9 0015 0012 0015 003F 0015
003F 0015 003F 0015 0012 0015 003F 0015 003F 0015 0012 0015 003F 0015
0012 0015 0012 0015 0012 0015 003F 0015 0012 0015 0012 0015 003F 0015
0012 0015 003F 0015 0015 0015 0012 0015 0015 0015 0012 0015 0015 0015
0015 0015 003F 0015 0015 0015 003F 0015 003F 0015 003F 0015 003F 0015
003F 0015 003F 0015 081F 0156 00A9 0015 0015 0015 003F 0015 003F 0015
003F 0015 0015 0015 003F 0015 003F 0015 0015 0015 003F 0015 0015 0015
0015 0015 0015 0015 003F 0015 0015 0015 0015 0015 003F 0015 0015 0015
003F 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015 0015
003F 0015 0015 0015 003F 0015 003F 0015 003F 0015 003F 0015 003F 0015
003F 0015 081F', function( success)
    print("IR sent:", success)
end)
pio1:listenForIR(function(ir)
    print("IR Received:", ir)
end)
```

The full source

You can find the latest version of the full source code in the Lua\pio1\init.lua file. Below is a copy of the file with automatic reconnecting and a few more useful functions.

© 2015 Promixis, LLC

```
local Base = require('Class')

local Promixis = Promixis
local transport = require('transport')
local string = require('string')
local print = print
local timer = require("timer")
local table = require("table")
local ipairs = ipairs
local tonumber = tonumber
local gir = gir

module (...)

Base:subclass( _M)

local TXR = Promixis.Transport.ITransactionCallback.Results


function listenForIR ( self, callback )

      if not self.connection then
            return false
      end

      local tx = self.connection:newTransaction(
            function()
            end,
            function( data )

                  if  string.find(data, "^nec") or
                        string.find(data, "^rc5") or
                        string.find(data, "^rc6") or
                        string.find(data, "^sirc") then

                        callback(data)
                        return TXR.TX_KEEP
                  else
                        return TXR.TX_CONTINUE
                  end

            end,
            function ()
            end
      )

      tx:persistent(true)
      self.connection:send(tx, true)

end
```

```
function getmac( self, callback )

	if not self.connection then
		return false
	end

	if not callback then
		callback = function( ) end
	end

	local retries = 0

	local tx = self.connection:newTransaction(
		nil, function( data )
			-- data

			local _,_, mac = string.find(data, "^getmac (%x%x:%x%x:
%x%x:%x%x:%x%x:%x%x)")
			if mac then
				callback(mac)
			end

		end,
		function ()
			-- timeout
			retries = retries + 1
			if retries > 5 then
				callback( false )
				return TXR.TX_REMOVE
			else
				return TXR.TX_REQUEUE_FIRST
			end
		end
	)

	tx:timeout(5000)
	tx:data("getmac\n")
	tx:persistent(false)

	self.connection:send(tx, true)

end

function setrelay( self, relay, close, callback )

	if not self.connection then
		return false
	end

	relay = tonumber(relay)
```

© 2015 Promixis, LLC

```
        if close then
                close = 1
        else
                close = 0
        end
        if relay < 1 or relay > 3 then
                return false
        end

        if not callback then
                callback = function( ) end
        end

        local retries = 0
        local devices = {}

        local tx = self.connection:newTransaction(
                function()
                        -- sent
                end,
                function( data )
                        -- data
                        if data == "ok" then
                                callback(true)
                        end

                end,
                function ()
                        -- timeout
                        retries = retries + 1
                        if retries > 5 then
                                callback( false )
                                return TXR.TX_REMOVE
                        else
                                return TXR.TX_REQUEUE_FIRST
                        end
                end
        )

        tx:timeout(5000)
        tx:data("setrelay " .. relay .. "," .. close .. "\n")
        tx:persistent(false)

        self.connection:send(tx, true)

end


function sendCCF ( self, bitmap, repeats, ccf, callback )
```

```
	if not self.connection then
		return false
	end

	if not callback then
		callback = function( ) end
	end

	local retries = 0
	local devices = {}

	local tx = self.connection:newTransaction(
		function()
			-- sent
		end,
		function( data )
			-- data

			if data == "irdone" then
				callback( true )
			elseif data == "ok" then
				return TXR.TX_RESET_TIMEOUT
			elseif data == "error 2,0" then
				return TXR.TX_REQUEUE
			else
				callback(false)
				return TXR.TX_REMOVE
			end


		end,
		function ()
			-- timeout
			retries = retries + 1
			if retries > 5 then
				callback( false )
				return TXR.TX_REMOVE
			else
				return TXR.TX_REQUEUE_FIRST
			end
		end
	)

	tx:timeout(5000)
	tx:data("sendccf " .. bitmap .. "," .. repeats .. "," .. ccf ..
"\n")
	tx:persistent(false)

	self.connection:send(tx, true)
```

```
end


function getversion ( self, callback )

	if not self.connection then
		return false
	end

	if not callback then
		callback = function( ) end
	end

	local retries = 0
	local devices = {}

	local tx = self.connection:newTransaction(
		function()
			-- sent
		end,
		function( data )
			-- data
			if string.find(data, "^getversion Promixis PIO") then
				callback(data)
			end

		end,
		function ()
			-- timeout
			retries = retries + 1
			if retries > 5 then
				callback( false )
				return TXR.TX_REMOVE
			else
				return TXR.TX_REQUEUE_FIRST
			end
		end
	)

	tx:timeout(2000)
	tx:data("getversion\n")
	tx:persistent(false)
	self.connection:send(tx, true)
end

local testconnection = function(self)

	getversion(self, function( status )
```

```
		if not status then
			self.timer:deinit()
			self.timer = nil

			if not self.closing then

				self.connection:close()
			end
		end

	end)

end

local connectionCallback = function(self, event, reason )


	if event ==
Promixis.Transport.IConnectionCallback.Status.CONNECTION_ESTABLISHED then

		listenForIR(self, function( ir )
			gir.triggerEvent( ir .. " " .. self.mac, 18,
Promixis.Event.MOD_ON )
		end)

		getmac(self, function(mac)
			if mac then
				self.mac = mac
			end
		end)

		self.timer = timer.new( 30000, function(t)

			testconnection(self)

		end)

		self.timer:start()

	end

	if event ==
Promixis.Transport.IConnectionCallback.Status.CONNECTION_CLOSED then

		if self.timer then
			self.timer:deinit()
			self.timer = nil;
		end

		if not self.closing then
```

© 2015 Promixis, LLC

```
                        self.connection:reconnect()
                end

        end

        if event ==
Promixis.Transport.IConnectionCallback.Status.CONNECTION_FAILED then

                if not self.closing then

                        self.connection:reconnect()
                end
        end

end

function connect( self )

        self.connection = transport.new
( Promixis.Transport.Connection.Type.CON_TCP, self.address, self.port,
function ( event, reason )
                connectionCallback(self, event,reason)
        end)


self.connection
:parser( Promixis.Transport.IParser.Type.PARSER_TERMINATED, "\r\n" )
        self.connection:connect()

end

function init ( self, address, port )

        Base.init(self)

        self.mac = ''
        self.address = address
        self.port = port

        if not self.port then
                self.port = 6000
        end

end

function deinit( self )

        Base.deinit(self)

        self.closing = true
```

```
        self.connection:close()

        if self.timer then
                self.timer:deinit()
                self.timer = nil;
        end

end
```

18.34 twilio

The Twilio plugin allows the sending of text messages and make automated phone calls. For this functionality to work you must have an account with Twilio. Once you are registered with Twilio please enter the sid and auth token on the plugin settings page. Without these entered these functions will not work.

Please note that using this functionality will incur costs with Twilio.

18.34.1 callNumber

This function will call a regular phone (or cellphone) with a voice message. The text spoken is provided in this message. At the end of the message the receiving call will get the option to hear the message again.

Definition

```
twilio.callNumber( to, from, message, callback )
```

Name	Type	Description
to	string	Phone number to call.
from	string	Phone to call from. Must be one of the numbers returned by phoneNumbers.
message	string	The message to speak.
callback	function	Callback to be called after operation completes.

Callback function signature

```
function ( status, err )
```

Name	Type	Description
status	boolean	status of the phone call send, true if sent succesfully and false if failed. Note that

© 2015 Promixis, LLC

		a true value does not mean the call was successful.
err	string	Error description if any.

Example

```
twilio.callNumber('8055049741', '8055049741', 'Your house is on fire',
function ( status, err )
      print("Phone call sent: ", status, err )
end)
```

This will print

```
SMS Send: true
```

Related

Twilio
sendSMS
phoneNumbers

Availability

Lua
Actions

18.34.2 phoneNumbers

Twilio only allows to send text messages or call phones from a set of registered phone numbers. This function returns the list of numbers.

Definition

```
numbers = twilio.phoneNumbers()
```

Name	Type	Description
numbers	table of strings	Allowed source phone numbers.

Example

```
for _, number in ipairs(numbers) do
      print(number)
end
```

This will print a list of phone numbers.

Related

Twilio
sendSMS
callNumber

Availability

Lua

18.34.3 sendSMS

sendSMS allows text messages (also known as SMS, or short messages service) to mobile phones.

Definition

```
twilio.sendSMS( to, from, body, callback )
```

Name	Type	Description
to	string	The mobile phone number to send to.
from	string	A phone number from your phone number list registered with Twilio.
body	string	The message to send. Max 140 characters.
callback	function	The function that will be called once the message was sent (or not, check the status)

Callback function signature

```
function ( status, err )
```

© 2015 Promixis, LLC

Name	Type	Description
status	boolean	status of the SMS send, true if sent succesfully and false if failed. Note that SMS send does not mean it was received yet.
err	string	Error description if any.

Example

```
twilio.sendSMS('8055049741', '8055049741', 'Your house is on fire',
function ( status, err )
      print("SMS Send: ", status, err )
end)
```

This will print

```
SMS Send: true
```

Related

Twilio
callNumber
phoneNumbers

Availability

Lua
Actions

18.35 Twitter

Girder can send Tweet and read your direct messages. Make sure you authorize the Twitter plugin before using these functions as they will not work unauthorized.

18.35.1 tweet

This function will send a tweet on behalf of the authorized account.

Definition

```
twitter.tweet( message, callback )
```

© 2015 Promixis, LLC

Name	Type	Description
message	string	message to send, limited to 140 characters.
callback	function	Callback to be called after operation completes.

Callback function signature

```
function ( status, messageId )
```

Name	Type	Description
status	boolean	true if tweet was published false otherwise.
messageId	string	The id of the published tweet.

Example

```
twitter.tweet('Hello World', function ( status, messageId )
      print("Tweet sent: ", status, messageId )
end)
```

This will print

```
Tweet sent: 2342342342
```

Related

Twitter

Availability

Lua
Actions

18.35.2 directMessages

This function will query the direct message in the authorized account

Definition

© 2015 Promixis, LLC

```
twitter.directMessages( sinceId, callback )
```

Name	Type	Description
sinceId	string	ID of the last message received after which you want to retrieve messages.
callback	function	Callback to be called after operation completes.

Callback function signature

```
function ( status, messages )
```

Name	Type	Description
status	boolean	true if tweet was published false otherwise.
messages	array of message	The id of the published tweet.

Message Table

Name	Type	Description
recipientId	string	ID of the recipient
senderId	string	ID of the sender
id	string	ID of the message
senderName	string	Screen name of the sender
recipientName	string	Screen name of the recipient.

Example

```
twitter.directMessages( "", function( success, messages )

  for i, message in ipairs(messages) do
```

© 2015 Promixis, LLC

```
    table.print(message)

  end


end)
```

This will print

```
{ -- #0
   ["recipientId"] = "1432955723",
   ["senderId"] = "15643215723",
   ["id"] = "366978234159033360",
   ["senderName"] = "Promixis",
   ["text"] = "yow yow",
   ["recipientName"] = "Promixis",
} -- #0
```

Related

Twitter

Availability

Lua
Actions

18.36 usbuirt

This function will send a CCF code from the attached USB-UIRT.

Definition

```
usbuirt.transmit( ccf, repeats, zone )
```

Name	Type	Description
ccf	string	CCF code to send.
repeats	number	Number of times to repeat
zone	number	Zone to send from 0 = all 1 = zone 1 2 = zone 2 3 = zone 3

© 2015 Promixis, LLC

Example

```
usbuirt.transmit('0000 006B 000A 000A 00E7 002F 002E 002F 0017 0018 0017
0018 0017 002F 0017 0018 0017 002F 0017 002F 002E 0018 0017 0484 0074
002F 002E 002F 0017 0018 0017 0018 0017 002F 0017 0018 0017 002F 0017
002F 002E 0018 0017 0484', 2, 0)
```

Availability

Lua
Actions

18.37 zwave

18.37.1 poll

This function will attempt to poll the control.

Definition

```
requested = zwave.poll( controlId )
```

Name	Type	Description
requested	boolean	Returns true if the control allowed polling and polling was initiated.
controlId	number	The Id of the control to poll for. Note this might update other controls that source their information from the same source as well.

Availability

Lua
Actions

© 2015 Promixis, LLC

19 Regular Expressions

A good tutorial on regular expressions can be found here: http://www.regular-expressions.info/tutorial.html. Girder uses PCRE for regular expression in Events and gir.addEventHandler. Lua uses it's own regular expression, which can be found here: http://promixis.com/lua/manual.html#5.4.1

Cheat Sheet

This sheet applies to the PCRE regular expression used in Girder. PCRE is used in gir.addEventHandler and Event Matching. Note that when typing regular expression in Lua the backslash (\) must be escaped, thus "\\".

Anchors

Sequence	Description
^	Start of line
\A	Start of a string
$	End of a line
\Z	End of a string
\b	Word boundary
\B	Not a word boundary
\<	Start of a word
\>	End of a word

Assertions

Sequence	Description
?=	Look ahead
?!	Negative Look Ahead
?<=	Look behind
?!=	Negative look behind
?<!	Negative look behind
?>	Once only sub-expression
?()	An "If Then" condition
?()|	An "If then else" condition
?#	An comment

© 2015 Promixis, LLC

Quantifiers

Sequence	Description
*****	0 or more
***?**	Un-greedy 0 or more
+	1 or more
+?	Un-greedy 1 or more
?	0 or 1
??	Un-greedy 0 or 1
{2}	Exactly 2
{2,}	2 or more
{2,6}	2,3,4,5 or 6
{2,6}?	Ungreedy 2,3,4,5 or 6

Character Classes

Sequence	Description
\c	Control Character
\s	White space
\S	Not White space
\d	Digit (0-9)
\D	Not a digit
\w	A word
\W	Not a word
\x	Hexadecimal (lower case!)
\O	Octal

Special Characters

Sequence	Description
****	Escape Character
\n	Newline
\r	Carriage Return
\t	Tab
\v	Vertical Feed

© 2015 Promixis, LLC

\f	Form Feed
\a	Alarm
[\b]	Backspace
\e	Escape

Ranges

<table>
<tr><th>Sequence</th><th>Description</th></tr>
<tr><td>(a|b)</td><td>a or b</td></tr>
<tr><td>.</td><td>Any character except a newline</td></tr>
<tr><td>( )</td><td>Capture group</td></tr>
<tr><td>(?:)</td><td>Passive group</td></tr>
<tr><td>[abc]</td><td>a, b or c</td></tr>
<tr><td>[^abc]</td><td>Not a, b or c</td></tr>
<tr><td>[a-z]</td><td>Letters between a and z</td></tr>
<tr><td>[A-Z]</td><td>Letters between A and Z</td></tr>
<tr><td>[0-9]</td><td>Numbers between 0-9</td></tr>
<tr><td>[A-Fa-f0-9]</td><td>Case insensitive hexadecimal</td></tr>
</table>

20 Z-Wave

This product can be included and operated in any Z-Wave network with other Z-Wave certified devices from other manufacturers and/or other applications. All non-battery operated nodes within the network will act as repeaters regardless of vendor to increase reliability of the network.

20.1 How to add or remove a device

Enter topic text here.

20.2 How to copy the controller

Enter topic text here.

20.3 How to place controller in learn mode

Enter topic text here.

20.4 How to include controller into existing network

Enter topic text here.

© 2015 Promixis, LLC

20.5 Association Support

Grouping identifier

☐ Maximum number of devices that can be added to the group

☐ Description of how the association group is used and/or triggered by the product

© 2015 Promixis, LLC

Index

- A -

action 16
actions 16

- E -

event 16
events 16, 30

- I -

introduction 10
ir 30

- S -

server 30

- W -

web 30

© 2015 Promixis, LLC