Logitech G HUB Lua API
V2023.5

Overview and Reference

Contents

Overview

The G-series Lua API is a set of functions using the Lua programming language that provides advanced scripting functionality for the G-series line of gaming keyboards and mice.

This document assumes a working knowledge of the Lua programming language. Further information can be obtained from www.lua.org.

Every G-series Profile has a default Lua script bound to it which can be edited and customized. The script is invoked using an event handler: OnEvent. Users may examine the various events exposed in this handler to perform their desired actions.

Reference

Functions

OnEvent

The **OnEvent**() function serves as the event handler for the script. You will need to implement this function.

```
function OnEvent(event, arg [family])

end
```

Parameters

event

String containing the event identifier.

arg

Argument correlating to the appropriate identifier.

family

Family of device creating the hardware event. Empty if event is not hardware specific. Use this if you need to distinguish input from multiple devices.

Family	Devices
“kb”	Supported keyboard devices
“lhc”	Supported left-handed controllers
“mouse”	Supported gaming mouse

Return Values

None

Remarks

The following is the list of identifiers and their arguments:

Event	arg	Description
"PROFILE_ACTIVATED"	None	Profile has been activated. This is the first event seen.
"PROFILE_DEACTIVATED"	None	Profile has been deactivated. This is the last event seen.
"G_PRESSED"	1=G1 18=G18 n = G_n	G Key pressed
"G_RELEASED"	1=G1 18=G18 n = G_n	G Key released
"M_PRESSED"	1=M1 2=M2 3=M3	M Key pressed
"M_RELEASED"	1=M1 2=M2 3=M3	M Key released

“MOUSE_BUTTON_PRESSED”	2=Mouse Button 2 3=Mouse Button 3 4=Mouse Button 4 …	Mouse Button Pressed NOTE: Left Mouse Button (1) is not reported by default. Use ‘EnablePrimaryMouseButtonEvents’ to override this.
“MOUSE_BUTTON_RELEASED”	2=Mouse Button 2 3=Mouse Button 3 4=Mouse Button 4 ..	NOTE: Left Mouse Button (1) is not reported by default. Use ‘EnablePrimaryMouseButtonEvents’ to override this.

Example

```
-- This is the primary event handler. You must implement this function

function OnEvent(event, arg)
	if (event == "PROFILE_ACTIVATED") then
		-- Profile has been activated
	end

	if (event == "PROFILE_DEACTIVATED") then
		-- Profile has been deactivated
	end

	if (event == "G_PRESSED" and arg == 1) then
		-- G1 has been pressed
	end

	if (event == "G_RELEASED" and arg == 1) then
		-- G1 has been released
	end

	if (event == "M_PRESSED" and arg == 1) then
		-- M1 has been pressed
	end

	if (event == "M_RELEASED" and arg == 1) then
		-- M1 has been released
	end

	if (event == "MOUSE_BUTTON_PRESSED" and arg == 6) then
		-- Mouse Button 6 has been pressed
	end

	if (event == "MOUSE_BUTTON_RELEASED" and arg == 6) then
		-- Mouse Button 6 has been released
	end
end
```

GetMKeyState

GetMKeyState() returns the current state of the M keys.

```
mkey GetMKeyState([family])
```

Parameters

family

Optional family name of device if you want to distinguish between multiple attached devices. Default is “kb”.

Family	Devices
“kb”	Supported keyboard devices
“lhc”	Supported left-handed controllers

Return Values

mkey

1 = M1, 2 = M2, 3 = M3

Remarks

Example

```
-- Get the current M Key state
current_mkey = GetMKeyState()
```

SetMKeyState

SetMKeyState() sets the current state of the M keys.

NOTE: Calling **GetMKeyState** immediately afterwards will likely return the previous state. Use the OnEvent handler to determine when the operation has completed.

```
mkey SetMKeyState(mkey, [family])
```

Parameters

mkey

1 = M1, 2 = M2, 3 = M3

family

Optional family name of device if you want to distinguish between multiple attached devices. Default is “kb”.

Family	Devices
“kb”	Supported keyboard devices
“lhc”	Supported left-handed controllers

Return Values

None

Remarks

Example

```
-- Set the current M Key state to M1 when G1 is pressed

function OnEvent(event, arg)
	if (event == "G_PRESSED" and arg == 1) then
		SetMkeyState(1)
	end
end
```

Sleep

Sleep() will cause the script to pause for the desired amount of time.

```
Sleep( timeout )
```

Parameters

timeout

Total time to sleep in milliseconds.

Return Values

nil

Remarks

Scripting runs on a separate thread than the main Profiler, thus pausing the script will not affect it.

You can use this function to simulate delays.

Deactivation of the profiler will wait 1 second for the script to finish, after which the script will be forcefully aborted. Take precaution if using a long timeout.

Example

```
-- Sleeping for 20 milliseconds
Sleep(20)
```

OutputLogMessage

OutputLogMessage() will send log messages into the script editor's console.

```
OutputLogMessage( ... )
```

Parameters

message

Printf-style formatted string containing the message.

Return Values

nil

Remarks

Mirror of string.format().

You must manually insert a carriage return "\n" to denote end of line.

Example

```
-- Send out "Hello World"
OutputLogMessage("Hello World %d\n", 2007)
```

GetRunningTime

GetRunningTime() returns the total number of milliseconds elapsed since the script has been running.

```
elapsed GetRunningTime()
```

Parameters

None

Return Values

elapsed

Integer value containing the elapsed time in milliseconds.

Remarks

You can use this to calculate timing in your script.

Example

```
-- Display the script running time
OutputLogMessage("This script has been running for: %d ms", GetRunningTime())
```

GetDate

Use **GetDate()** to retrieve the formatted date

```
date GetDate ([format [, time]])
```

Parameters

format

Optional date format string.

time

Optional time table.

Return Values

date

A string or table containing the user's machine's current date and time (or the time represented by time), formatted according to the given string format. If one wishes to supply their own format string it uses the same rules as strftime(). The special string *t tells the date() function to return a table.

Remarks

Mirror of os.date().

Example

```
-- Display the current date/time
OutputLogMessage("Today’s date/time is: %s\n", GetDate())
```

ClearLog

The **ClearLog**() function clears the output window of the script editor.

```
ClearLog()
```

Parameters

None.

Return Values

nil

Remarks

None.

Example

```
-- Clear the script editor log
OutputLogMessage("This message will self destruct in 2 seconds\n")
Sleep(2000)
ClearLog()
```

PressKey

The **PressKey**() function is used to simulate a keyboard key press.

NOTE: Calling **IsModifierPressed** or **IsKeyLockOn** immediately afterwards for a simulated modifier or lock key will likely return the previous state. It will take a few milliseconds for the operation to complete.

```
PressKey( scancode [,scancode] )

PressKey( keyname [,keyname] )
```

Parameters

scancode

Specifies the numerical scancode of the key to be pressed.

keyname

Specifies the predefined keyname of the key to be pressed.

Return Values

nil

Remarks

If multiple keys are provided as arguments, all keys will be simulated with a press.

For scancode and keyname values, refer to Appendix A.

Example

```
-- Simulate "a" pressed using the scancode
PressKey(30)

-- Simulate "a" pressed using the keyname
PressKey("a")

-- Simulate "a" and "b" being pressed
PressKey("a", "b")
```

ReleaseKey

The **ReleaseKey**() function is used to simulate a keyboard key release.

```
ReleaseKey( scancode [,scancode] )

ReleaseKey( keyname [,keyname] )
```

Parameters

scancode

Specifies the numerical scancode of the key to be pressed.

keyname

Specifies the predefined keyname of the key to be pressed.

Return Values

nil

Remarks

If multiple keys are provided as arguments, all keys will be simulated with a release.

For scancode and keyname values, refer to Appendix A.

Example

```
-- Simulate "a" released using the scancode
ReleaseKey(30)

-- Simulate "a" released using the keyname
ReleaseKey("a")

-- Simulate "a" and "b" being released
ReleaseKey("a", "b")
```

PressAndReleaseKey

The **PressAndReleaseKey**() function is used to simulate a keyboard key press followed by a release.

NOTE: Calling **IsModifierPressed** or **IsKeyLockOn** immediately afterwards for a simulated modifier or lock key will likely return the previous state. It will take a few milliseconds for the operation to complete.

```
PressAndReleaseKey( scancode [,scancode] )

PressAndReleaseKey( keyname [,keyname] )
```

Parameters

scancode

Specifies the numerical scancode of the key to be pressed.

keyname

Specifies the predefined keyname of the key to be pressed.

Return Values

nil

Remarks

If multiple keys are provided as arguments, all keys will be simulated with a press and a release.

For scancode and keyname values, refer to Appendix A.

Example

```
-- Simulate "a" pressed and released using the scancode
PressAndReleaseKey(30)

-- Simulate "a" pressed and released using the keyname
PressAndReleaseKey("a")

-- Simulate "a" and "b" being pressed and released
PressAndReleaseKey("a", "b")
```

IsModifierPressed

The **IsModifierPressed**() function is used to determine if a particular modifier key is currently in a pressed state.

```
boolean IsModifierPressed ( keyname )
```

Parameters

keyname

Specifies the predefined keyname of the modifier key to be pressed. The name must be one of the following:

Modifier	Description
"lalt", "ralt", "alt"	Left, right, or either Alt key
"lshift", "rshift", "shift"	Left, right, or either Shift key
"lctrl", "rctrl", "ctrl"	Left, right, or either Ctrl key

Return Values

True if the modifier key is currently pressed, false otherwise.

Remarks

None.

Example

```
-- Press a specific modifier
PressKey("lshift")
-- Sleep for 100 ms to allow IsModifierPressed() to get an accurate result
Sleep(100)

if IsModifierPressed("shift") then
   OutputLogMessage("shift is pressed.\n")
end

-- Release the key so it is no longer pressed
ReleaseKey("lshift")
-- Sleep for 100 ms to allow IsModifierPressed() to get an accurate result
Sleep(100)

if not IsModifierPressed("shift") then
   OutputLogMessage("shift is not pressed.\n")
end
```

PressMouseButton

The **PressMouseButton**() function is used to simulate a mouse button press.

NOTE: Calling **IsMouseButtonPressed** immediately afterwards, will likely return the previous state. It will take a few milliseconds for the operation to complete.

PressMouseButton(button)

Parameters

button

Button identifier. Use the following table:

Button value	Location
1	Left Mouse Button
2	Middle Mouse Button
3	Right Mouse Button
4	X1 Mouse Button
5	X2 Mouse Button

Return Values

nil

Remarks

None

Example

```
-- Simulate left mouse button press
PressMouseButton(1)

-- Simulate right mouse button press
PressMouseButton(3)
```

ReleaseMouseButton

The **ReleaseMouseButton**() function is used to simulate a mouse button release.

```
ReleaseMouseButton( button )
```

Parameters

button

Button identifier. Use the following table:

Button value	Location
1	Left Mouse Button
2	Middle Mouse Button
3	Right Mouse Button
4	X1 Mouse Button
5	X2 Mouse Button

Return Values

nil

Remarks

None

Example

```
-- Simulate a left mouse button click (press and release)
PressMouseButton(1)
ReleaseMouseButton(1)
```

PressAndReleaseMouseButton

The **PressAndReleaseMouseButton**() function is used to simulate a mouse button press followed by a release.

NOTE: Calling **IsMouseButtonPressed** immediately afterwards will likely return the previous state. It will take a few milliseconds for the operation to complete.

PressAndReleaseMouseButton(button)

Parameters

button

Button identifier. Use the following table:

Button value	Location
1	Left Mouse Button
2	Middle Mouse Button
3	Right Mouse Button
4	X1 Mouse Button
5	X2 Mouse Button

Return Values

nil

Remarks

None

Example

```
-- Simulate a left mouse button click (press and release)
PressAndReleaseMouseButton(1)
```

IsMouseButtonPressed

The **IsMouseButtonPressed**() function is used to determine if a particular mouse button is currently in a pressed state.

```
boolean IsMouseButtonPressed( button )
```

Parameters

button

Button identifier. Use the following table:

Button value	Location
1	Left Mouse Button
2	Middle Mouse Button
3	Right Mouse Button
4	X1 Mouse Button
5	X2 Mouse Button

Return Values

True if the button is currently pressed, false otherwise.

Remarks

None

Example

```
-- Press a mouse button
PressMouseButton(1)

if IsMouseButtonPressed(1) then
   OutputLogMessage("Left mouse button is pressed.\n")
end

-- Release the button so it is no longer pressed
ReleaseMouseButton(1)

if not IsMouseButtonPressed(1) then
   OutputLogMessage("Left mouse button is not pressed.\n")
end
```

MoveMouseTo

The **MoveMouseTo**() function is used to move the mouse cursor to an absolute position on the screen.

NOTE: Calling **GetMousePosition** immediately afterwards, will likely return the previous state. It will take a few milliseconds for the operation to complete.

MoveMouseTo(x, y,)

Parameters

x

Normalized X coordinate between 0 (farthest left) and 65535 (farthest right)

Y

Normalized y coordinate between 0 (farthest top) and 65535 (farthest bottom)

Return Values

nil

Remarks

If multiple monitors are present, use MoveMouseToVirtual.

Example

```
-- Move mouse to upper, left corner
MoveMouseTo(0, 0)

-- Move mouse to center of screen
MoveMouseTo(32767, 32767)

-- Move mouse to lower, right corner
MoveMouseTo(65535, 65535)
```

MoveMouseWheel

The **MoveMouseWheel**() function is used to simulate mouse wheel movement.

MoveMouseWheel(click)

Parameters

click

Number of mouse wheel clicks.

Return Values

nil

Remarks

Positive values denote wheel movement upwards (away from user).

Negative values denote wheel movement downwards (towards user).

Example

```
-- Simulate mouse wheel 3 clicks up
MoveMouseWheel(3)

-- Simulate mouse wheel 1 click down
MoveMouseWheel(-1)
```

MoveMouseRelative

The **MoveMouseRelative**() function is used to simulate relative mouse movement.

NOTE: Calling **GetMousePosition** immediately afterwards, will likely return the previous state. It will take a few milliseconds for the operation to complete.

MoveMouseRelative(x, y)

Parameters

x

Movement along the x-axis

Y

Movement along the y-axis

Return Values

nil

Remarks

Positive x values simulate movement to right.

Negative x values simulate movement to left.

Positive y values simulate movement downwards.

Negative y values simulate movement upwards.

Example

```
-- Simulate relative mouse movement upwards in 1 pixel increments
for i = 0, 50 do
	MoveMouseRelative(0, -1)
	Sleep(8)
end
```

MoveMouseToVirtual

The **MoveMouseToVirtual**() function is used to move the mouse cursor to an absolute position on a multi-monitor screen layout.

NOTE: Calling **GetMousePosition** immediately afterwards, will likely return the previous state. It will take a few milliseconds for the operation to complete.

MoveMouseToVirtual(x, y)

Parameters

x

Normalized X coordinate between 0 (farthest left) and 65535 (farthest right)

Y

Normalized y coordinate between 0 (farthest top) and 65535 (farthest bottom)

Return Values

nil

Remarks

If multiple monitors are present, use MoveMouseToVirtual.

Example

```
-- Move mouse to upper, left corner of virtual desktop
MoveMouseToVirtual(0, 0)

-- Move mouse to center of virtual desktop
MoveMouseToVirtual(32767, 32767)

-- Move mouse to lower, right corner of virtual desktop
MoveMouseToVirtual(65535, 65535)
```

GetMousePosition

The **GetMousePosition**() function returns the normalized coordinates of the current mouse cursor location.

```
x,y GetMousePosition()
```

Parameters

None

Return Values

x

Normalized X coordinate between 0 (farthest left) and 65535 (farthest right)

Y

Normalized y coordinate between 0 (farthest top) and 65535 (farthest bottom)

Remarks

Example

```
-- Get the current mouse cursor position
x, y = GetMousePosition()
OutputLogMessage("Mouse is at %d, %d\n", x, y)
```

OutputLCDMessage

The **OutputLCDMessage**() function is used to add a line of text on to the LCD.

OutputLCDMessage(text [,timeout])

Parameters

text

String to display

timeout

Timeout in milliseconds, after which the message will disappear

Return Values

nil

Remarks

Up to 4 lines of text can be displayed at once. The default timeout is 1 second.

This function is not implemented in G HUB.

Example

```
-- Display some text with default timeout
OutputLCDMessage("Hello world")

-- Display some text for 2 seconds
OutputLCDMessage("Hello world", 2000)
```

ClearLCD

The **ClearLCD**() function clears the script display on the LCD.

```
ClearLCD( )
```

Parameters

none

Return Values

nil

Remarks

This function is not implemented in G HUB.

Example

```
-- Clear the LCD and then display 2 lines of text
ClearLCD()
OutputLCDMessage("Hello world1")
OutputLCDMessage("Hello world2")
```

PlayMacro

The **PlayMacro**() function is used to play an existing macro.

```
PlayMacro( macroname )
```

Parameters

macroname

Name of existing macro belonging to the current profile.

Return Values

nil

Remarks

If the function is called while another script macro is playing, no action is taken. In other words, only one script macro may be playing at any given time.

If the function is called while the same script macro is playing, the macro is queued.

The macro will simulate a key press down (on a virtual macro key), wait for 100ms, then simulate a key release. This way macros will on press and on release will be played properly. For granular control on macro playback, take a look at PressMacro and ReleaseMacro functions.

Example

```
-- Play an existing macro
PlayMacro("my macro")
```

PressMacro

The **PressMacro**() function is used to play an existing macro by simulating a key press down.

```
PressMacro( macroname )
```

Parameters

macroname

Name of existing macro belonging to the current profile.

Return Values

nil

Remarks

If the function is called while another script macro is playing, no action is taken. In other words, only one script macro may be playing at any given time.

If the function is called while the same script macro is playing, the macro is queued.

Example

```
-- Play an existing macro
PressMacro("my macro")
```

ReleaseMacro

The **ReleaseMacro**() function is used to play an existing macro by simulating a key release.

```
ReleaseMacro( macroname )
```

Parameters

macroname

Name of existing macro belonging to the current profile.

Return Values

nil

Remarks

If the function is called while another script macro is playing, no action is taken. In other words, only one script macro may be playing at any given time.

If the function is called while the same script macro is playing, the macro is queued.

Example

```
-- Play an existing macro
ReleaseMacro("my macro")
```

AbortMacro

The **AbortMacro**() function is used to abort any macro started from a script.

AbortMacro( )

Parameters

None

Return Values

nil

Remarks

Any keys still pressed after a call to PlayMacro will be released. Macros playing outside the script will continue to play.

Example

```
-- Start a macro
PlayMacro("my macro")

-- Wait for 100ms and then abort any playing macro
Sleep(100)
AbortMacro()
```

IsKeyLockOn

The **IsKeyLockOn**() function used to determine if a particular lock button is currently in an enabled state.

```
IsKeyLockOn( key )
```

Parameters

key

key name. Use the following table:

Key name	Location
"scrolllock"	Scroll Lock
"capslock"	Caps Lock
"numlock"	Number Lock

Return Values

True if the lock is currently enabled, false otherwise.

Remarks

None.

Example

```
-- Check if the numlock is on and turn it off if it is
if IsKeyLockOn("numlock") then
	PressAndReleaseKey("numlock")
end
```

SetBacklightColor

The **SetBacklightColor**() function is used to set the custom backlight color of the device (if the device supports custom backlighting).

SetBacklightColor(red, green, blue, [family])

Parameters

red

Red intensity (0 – 255)

green

Green intensity (0 – 255)

blue

Blue intensity (0 – 255)

family

Optional family name of device if you want to distinguish between multiple attached devices. Default is “kb”.

Family	Devices
“kb”	Keyboard devices (G15, G11, G19, etc)
“lhc”	Left handed controllers (G13, etc)

Return Values

nil

Remarks

This function is not implemented in G HUB.

Example

```
-- Set the backlight to red
SetBacklightColor(255, 0, 0)

-- Set the backlight color for all left handed controllers to blue
SetBacklightColor(0, 0, 255, "lhc")
```

OutputDebugMessage

OutputDebugMessage() will send log messages to the Windows debugger.

```
OutputDebugMessage( ... )
```

Parameters

Message

Printf style, formatted string containing the message.

Return Values

nil

Remarks

Mirror of string.format().

You must manually insert a carriage return "\n" to denote end of line.

Use tools like Dbg View for viewing these messages.

Example

```
-- Send out "Hello World"

OutputDebugMessage("Hello World %d\n", 2007)
```

SetMouseDPITable

SetMouseDPITable() sets the current DPI table for a supported gaming mouse

SetMouseDPITable({*value1, value2, value3}, [index]*)

Parameters

DPI Array

Array of DPI values

DPI Index

Optional 1-Based index to DPI to apply as the current DPI.

Return Values

nil

Remarks

If the index is not specified, the first entry is used as the current DPI.

A maximum of 16 entries are allowed.

Activating a profile with per-profile DPI settings will override any previously applied DPI.

This function is not implemented in G HUB.

Example

```
-- Set our DPI values to {500, 1000, 1500, 2000, 2500}
-- By default, 500 DPI will be set as the current DPI
SetMouseDPITable({500, 1000, 1500, 2000, 2500})

-- Set our DPI values to {500, 2500} and set the second value as the current DPI
SetMouseDPITable({500, 2500}, 2)
```

SetMouseDPITableIndex

SetMouseDPITableIndex() sets the current DPI table index for a supported gaming mouse

```
SetMouseDPITableIndex(index)
```

Parameters

Index

1-Based index into the DPI Table

Return Values

nil

Remarks

If SetMouseDPITable was not called prior to this, the current DPI table for the mouse is used.

A maximum of 16 entries are allowed.

Activating a profile with per-profile DPI settings will override any previously applied DPI.

This function is not implemented in G HUB.

Example

```
-- Set our initial DPI values to {500, 1000, 1500, 2000, 2500}
SetMouseDPITable({500, 1000, 1500, 2000, 2500})

-- Set the current DPI to the 3rd item in the table (1500 DPI)
SetMouseDPITableIndex(3)
```

EnablePrimaryMouseButtonEvents

EnablePrimaryMouseButtonEvents() enables event reporting for mouse button 1.

EnablePrimaryMouseButtonEvents(enable)

Parameters

enable

1 or true to enable event reporting for mouse button 1

0 or false to disable event reporting for mouse button 1

Return Values

nil

Remarks

The primary mouse button is not reported by default for performance issues.

Example

```
-- Enable event reporting for mouse button 1
EnablePrimaryMouseButtonEvents(true)

-- Disable event reporting for mouse button 1
EnablePrimaryMouseButtonEvents(false)
```

SetSteeringWheelProperty

SetSteeringWheelProperty() sets a steering wheel property.

```
SetSteeringWheelProperty(device, property, value)
```

Parameters

device

Device	Description
“G29”	Logitech G29 Steering Wheel
“G920”	Logitech G920 Steering Wheel

property

Property	Description
“operatingRange”	Operating range of wheel from 40 to 900. Default is 900.
“combinedPedals”	Combines the brake and accelerator into a single axis. The accelerator is on the + axis, and the brake is on the – axis. Default is false.
“defaultCenteringSpring”	Plays a persistent spring on top of any game forces. Default is false.
“defaultCenteringSpringStrength”	Sets the strength of the default centering spring from 0-100.

Return Values

nil

Remarks

This function is not implemented in G HUB.

Example

```
-- Set the operating range to 200 degrees for the G29
SetSteeringWheelProperty("G29", "operatingRange", 200)

-- Enable combined pedals on the G920
SetSteeringWheelProperty("G920", "combinedPedals", true)
```

G13 Programming

The G13 game panel has an analog joystick that can have a mouse function assigned to it. The speed of the mouse can be adjusted through either the profiler options panel in the settings window, or through the Lua scripting language. The following are the new Lua functions for mouse speed control:

SetMouseSpeed ()

Parameters

New mouse speed

Absolute mouse speed 32 to 255.

Return Values

nil

Remarks

This function is not implemented in G HUB.

Example

```
--Set Mouse speed to 128
SetMouseSpeed(128)
```

GetMouseSpeed()

Parameters

Current mouse speed

Absolute mouse speed 32 to 255.

Return Values

Current emulated mouse speed.

Remarks

This function is not implemented in G HUB.

Example

```
--Get Mouse speed
OutputLogMessage("The Mouse Speeed is: %d\n", GetMouseSpeed())
```

IncrementMouseSpeed()

Parameters

Mouse speed increment

Return Values

nil

Remarks

Resultant mouse speed will be clamped to a maximum of 255

This function is not implemented in G HUB.

Example

```
--Increase Mouse speed by 10
IncrementMouseSpeed(10)
```

DecrementMouseSpeed()

Parameters

Mouse speed decrement

Return Values

nil

Remarks

Resultant mouse speed will be clamped to a minimum of 32

This function is not implemented in G HUB.

Example

```
-- Decrease Mouse speed by 10
DecrementMouseSpeed(10)
```

The G13 mouse functionality does not support any native buttons, e.g. left button, center button, etc. Mouse buttons must be programmed via Lua. Here is an example of generic Lua code to effect mouse button operation:

```
if event=="G_PRESSED" and arg==x then
   PressMouseButton( y )
end

if event=="G_RELEASED" and arg==x then
   ReleaseMouseButton( y )
```

Standard Lua 5.4 Libraries

The following standard library functions are supported:

Math functions	String functions	Table functions
math.abs	string.byte	table.concat
math.acos	string.char	table.insert
math.asin	string.dump	table.move
math.atan	string.find	table.pack
math.ceil	string.format	table.remove
math.cos	string.gmatch	table.sort
math.deg	string.gsub	table.unpack
math.exp	string.len	
math.floor	string.lower	
math.fmod	string.match	
math.huge	string.pack	
math.log	string.packsize	
math.max	string.rep	
math.maxinteger	string.reverse	
math.min	string.sub	
math.mininteger	string.unpack	
math.modf	string.upper	
math.pi		
math.rad		
math.random		
math.randomseed		
math.sin		
math.sqrt		
math.tan		
math.tointeger		
math.type		
math.ult		

Appendix A

Table of scancodes and keynames used in **PressKey()**, **ReleaseKey()**, **IsModifierPressed()**.

Keyname	Scancode (hex)
"escape"	0x01
"f1"	0x3b
"f2"	0x3c
"f3"	0x3d
"f4"	0x3e
"f5"	0x3f
"f6"	0x40
"f7"	0x41
"f8"	0x42
"f9"	0x43
"f10"	0x44
"f11"	0x57
"f12"	0x58
"f13"	0x64
"f14"	0x65
"f15"	0x66
"f16"	0x67
"f17"	0x68
"f18"	0x69
"f19"	0x6a
"f20"	0x6b
"f21"	0x6c
"f22"	0x6d
"f23"	0x6e
"f24"	0x76
"printscreen"	0x137
"scrolllock"	0x46
"pause"	0x146
"tilde"	0x29
"1"	0x02
"2"	0x03
"3"	0x04
"4"	0x05
"5"	0x06
"6"	0x07
"7"	0x08
"8"	0x09
"9"	0x0a
"0"	0x0b
"minus"	0x0c
"equal"	0x0d
"backspace"	0x0e
"tab"	0x0f
"q"	0x10
"w"	0x11
"e"	0x12
"r"	0x13
"t"	0x14
"y"	0x15

"u"	0x16
"l"	0x17
"o"	0x18
"p"	0x19
"lbracket"	0x1a
"rbracket"	0x1b
"backslash"	0x2b
"capslock"	0x3a
"a"	0x1e
"s"	0x1f
"d"	0x20
"f"	0x21
"g"	0x22
"h"	0x23
"j"	0x24
"k"	0x25
"l"	0x26
"semicolon"	0x27
"quote"	0x28
"enter"	0x1c
"lshift"	0x2a
"non_us_slash"	0x56
"z"	0x2c
"x"	0x2d
"c"	0x2e
"v"	0x2f
"b"	0x30
"n"	0x31
"m"	0x32
"comma"	0x33
"period"	0x34
"slash"	0x35
"rshift"	0x36
"lctrl"	0x1d
"lgui"	0x15b
"lalt"	0x38
"spacebar"	0x39
"ralt"	0x138
"rgui"	0x15c
"appkey"	0x15d
"rctrl"	0x11d
"insert"	0x152
"home"	0x147
"pageup"	0x149
"delete"	0x153
"end"	0x14f
"pagedown"	0x151
"up"	0x148
"left"	0x14b
"down"	0x150
"right"	0x14d
"numlock"	0x45
"numslash"	0x135

"numminus"	0x4a
"num7"	0x47
"num8"	0x48
"num9"	0x49
"numplus"	0x4e
"num4"	0x4b
"num5"	0x4c
"num6"	0x4d
"num1"	0x4f
"num2"	0x50
"num3"	0x51
"numenter"	0x11c
"num0"	0x52
"numperiod"	0x53