Lua Debugger Manual

A simple debugger for use with a Lua scripting system.

By David Lannan

version 1.0

Contents

Table of Contents

Overview

The following section is a broad overview of what Lua Debugger is and why you would probably want to use it. In its initial state the Lua Debugger is fairly rudimentary, but quite useful for most purposes.

What is it

Lua Debugger is a tool that can be used with any Lua based system to be able to assess the current values of a Lua State. The most common use will be to check a value of a property or variable in the Lua State, or find out the properties of a table.

Lua Debugger is intended to be friendly, and usable by almost anyone with some understanding of the Lua scripting system. For the casual user of a Lua application, Lua Debugger will also be very useful for supplying feedback to Lua developers that have embedded the debugger in their application.

Why use it

The main reason you would use Lua Debugger is to debug errors and problems occuring in a Lua application. However, there are other considerations. Lua Debugger is an extremely lightweight and user friendly debugger (although there are other great debuggers available for Lua – this should not be considered the only one), and it is able to be embedded in any Lua system. There is no restriction on the use of the Lua Debugger.

Do I have to use it with LuaEng?

No. Not at all. If you have a Lua application or system that you'd like to have Lua Debugger attached to, then you need only include the project, provide the appropriate hooks and build against your Lua library. You will then have a dll, that you can include with your application to be able to provide debug facilities.

The required hooks for the dll are as follows:

DebuggerExecCommand, and Debugger Pause must be provided to be able to pause the application and execute commands within the applications running Lua state. These commands are actually not required to run the debugger, but of course are needed for specific debugger functionality.

```
__declspec( dllexport ) void DebuggerExecCommand( const char *command );
__declspec( dllexport ) void DebuggerPause( bool Pause );
```

OutputToDebugger and AttachToDebugger are required functions that the application must call to send information to the Debugger and to launch the Debugger respectively.The UpdateDebugger function ensures that Watches are updated per frame.

```
__declspec( dllimport ) void OutputToDebugger( const char *Output );
__declspec( dllimport ) bool AttachToDebugger( lua_State *iL );
__declspec( dllimport ) bool UpdateDebugger( lua_State *iL ) ;
```

For further details, refer to the project source in the LuaEng source tree. The implementation is extremely trivial, and it should take only a few minutes for an application to have Lua Debugger added to its build.

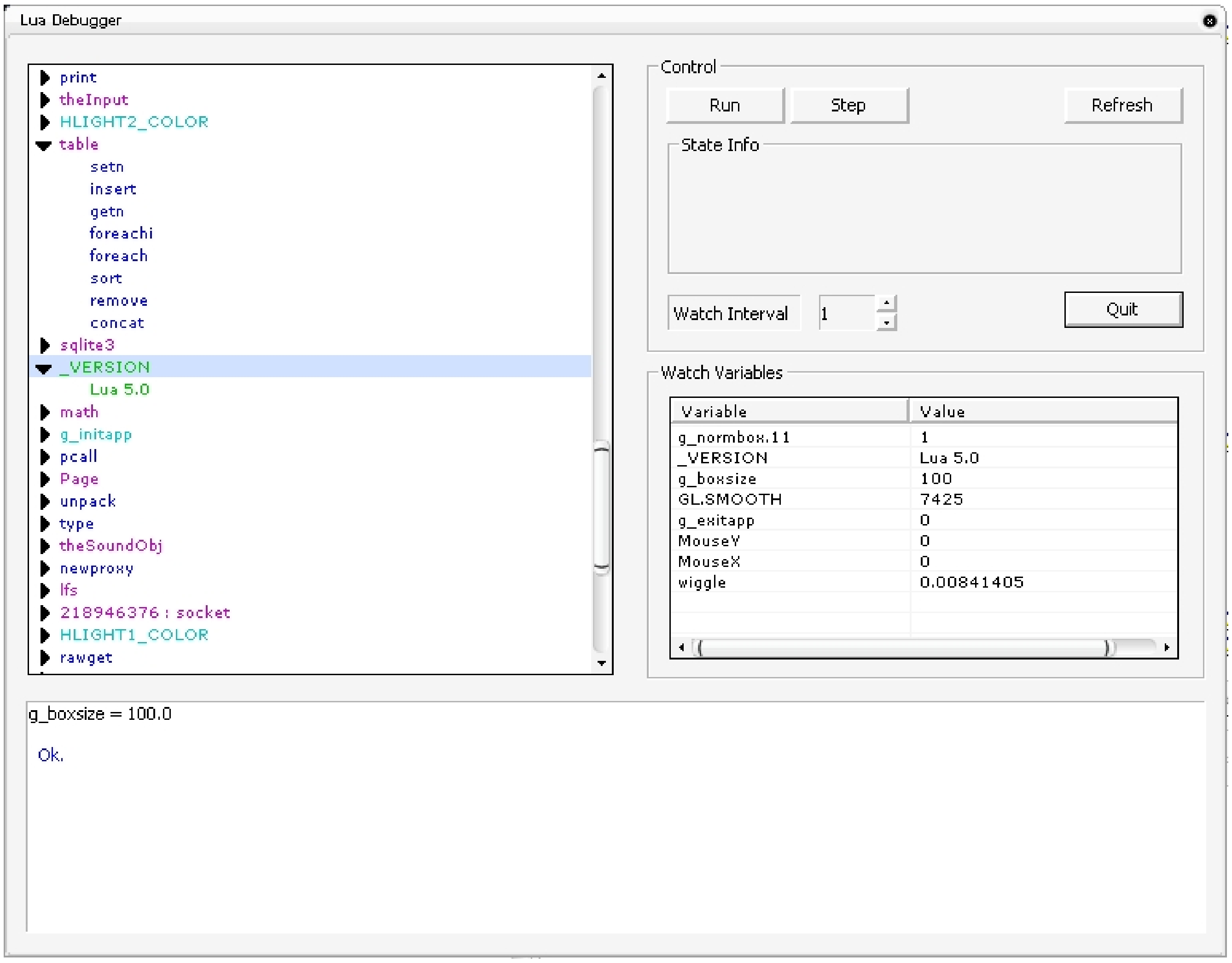
Usage

Lua Debugger is very simple to use, but if you are having trouble the following section should help you learn how to operate this tool.

Invoke debugger

From a running LuaEng application you can invoke the debugger at any time by pressing the 'Pause' key on your keyboard. If you are using Lua Debugger in your own application, then you will need to respond to the user and call the AttachToDebugger function to initiate the debugger.

Once displayed, you should see a similar window to the following:



```
print
theInput
HLIGHT2_COLOR
table
    setn
    insert
    getn
    foreachi
    foreach
    sort
    remove
    concat
sqlite3
_VERSION
    Lua 5.0
math
g_initapp
pcall
Page
unpack
type
theSoundObj
newproxy
lfs
218946376 : socket
HLIGHT1_COLOR
rawget
```

Variable	Value
g_normbox.11	1
_VERSION	Lua 5.0
g_boxsize	100
GL.SMOOTH	7425
g_exitapp	0
MouseY	0
MouseX	0
wiggle	0.00841405

```
g_boxsize = 100.0

 Ok.
```

The main left hand treeview is a view of your variables in Lua. These are all the global variables, tables and functions in the Lua State, with their values at the time you initiated the debugger. These values **do not** update continuously – see the section on Watch for more on this. It is important to note also, that the values are valid only at the instant that the Lua Debugger is invoked.

The bottom window is the Command Line and output window. On the right of the window is the Control pane and the Watch pane. Each of these panes are described below.

The colours of the variables in the treeview are as follows:

Purple : All Tables. Tables with a long number preceding them, are referenced tables – that is, the variable points to another already declared table in the list.

Green : Strings – all global string values.

Cyan : Integer variables or booleans.

Dark Blue : Functions

Red : Nil values – note, you should NEVER see this. If you do, you know there is something very wrong with your Lua garbage collector, nils should always be cleared!

From the treeview a user can right click on a selected variable name, and use the "Add Watch" action which will send the variable to the Watch window for constant updates.

Refresh

Refresh is a special operation, and really shouldnt be used too often, especially with large programs or Lua States. Refresh rebuilds the treeview of all globals, and also updates their entered values. This is only to update changes that a user would possibly create while executing some commands from the Command Line. An example is the creation of new variables, or the modification of tables.

Run, Step and Quit (Control Pane)

Control of the debugger is managed from the Control Pane. Once invoked, the Debugger can Pause execution, continue execution and step execution. To exit the debugger select the Quit button.

For applications that are embedding the Lua Debugger, some controls may not be available. Run and Pause can only operate if the approriate functions are exported to the Debugger.

During the use of a LuaEng application, the Pause will halt the cycle of calling the Main Game State loop, before the initial PreDraw function is called. Because the execution process is held in a loop, there will be no graphical updates done on the LuaEng window – this may change in future releases.

Once a Lua application is Paused, the Step button will issue a single LuaEng loop step. This allows very finite, frame by frame assessment of the Lua State. Watch values will update only for the entire stepped frame.

Pressing Run will resume normal execution. And pressing Step while in Run, will drop the application into Pause mode.

Watch Pane

The Watch Pane is used to examine variables during runtime with the capability to select how often the values are updated in the watch pane.

Too add a variable to the watch pane, select it in the treeview (making sure you select the correct level) and then select right click on the mouse, which will display two options "Add Watch" or "Add break". Select "Add Watch" and the variable will appear in the watch pane.

Currently the time interval features for the watch updates is not working – but soon to be implemented.

See 'Future' for more information on the implementation of this feature.

Command Line

Once a debugger is invoked, a user may execute any Lua command they wish. Clicking in the Command Line and issuing Lua commands will result in error returns (displayed in red after a command is executed) or an Ok return (Ok in blue after a successful command). Using the LuaEng system, all stdout and stderr streams are sent to the log files, so the Lua Debugger does not intercept these. If you are using print commands for example, the output will be sent to the log files.

The Command Line is keyboard sensitive. Everytime a 'enter' is entered while the Command Line window has focus, a command will be attempted to be executed. The line previous to the 'enter' will be used as the command string.

WARNING: I have found one very serious problem with the Command Line at this time of writing. You cannot set watch variables to nil, while they are in the watch pane. To solve this problem, simply quit from the debugger and reopen the debugger, which will empty the Watch pane. Then you can set any variable to nil safely, and refresh the treeview.

Lua State info

General Lua State information will be displayed in the Control Pane. This feature is not yet implemented – see Future for further notes.

Future

This section describes the features soon to be included in the Lua Debugger Tool.

Watch

A user will be able to add variables to watch within the watch window. The Watch interval will determine how often to update the watch values. A default value of 20 milliseconds is set as the base update watch interval. System performance will greatly effect the ability to lower this value and still be accurate.

Lua State info

Lua State info will be added to the Control Pane. This will include items like garbage to be collected (memory), current execution stack(probably in a separate window that can be opened), and various other important State information (stack depth, closures etc).

Refreshing and Live updates.

This is yet to be decided upon, because of the serious performance implications. The idea is to be able to refresh the Treeview control when browsing the treeview. While this sounds fairly innocuous, there is a lot of information to retrieve in some circumstances, and will impede the applications performance.