GameGuru

the easy game maker

SCRIPTING WITH LUA

DRAFT VERSION 0.1

Lua 5.0 license

For a full explanation of the LUA language you can visit ***http://www.lua.org***, however a brief overview of the language and how to create simple scripts will help you get started with creating your own game logic in GameGuru.

You can create a simple global variable like so:

```
Attack = 10
```

Or you can create a complex data structure, like so:

```
Database = {
 Weapons = {
  Sword =
  {
   Attack = 10
  }
 }
}
```

Or you can declare and fill arrays this way:

```
Data = {}
Data[1] = "Hello World"
```

Functions are declared as follows:

```
Function MyFunc(myvar)
 -- And comments look like this
end
```

You can make IF THEN statements inside your functions which look like this:

```
Function MyFunc(myvar)
 -- And comments look like this
 if myvar < 10 then
  myvar = 0; -- and assigning values to variables like so
 end
end
```

When creating LUA scripts for Reloaded, you must follow a strict set of rules for perfect compatibility with the engine. You should prefix all your global members and function names with the lower case name of the script file you have created, with the postfix _main tagged to the end of all function names, so a file called markb_explode.lua may look like this:

```
markb_myglobal = 123

Function markb_explode_main(myvar)
 -- my comment goes here
 if myvar < markb_myglobal then
  markb_myglobal = markb_myglobal + 1;
 end
end
```

By following these rules, you are ensuring that your script (which is placed in a global collection of scripts under a single LUA scope) do not conflict with each other, and your global variables are not overwritten by neighbouring scripts. It also means that in a dangerous and useful way, you can 'tap into' the global of neighbouring scripts if you so wished. It is highly recommended you avoid this however as it would make your script dependent on other script values and much less modular or predictable to re-use in future projects.

GameGuru pre-loads a series of LUA libraries for you:

- The maths library
- The string manipulation functions
- Array (table) manipulation functions
- IO (file handling) functions
- The Base library (core functions)

Below is a summary of all the functions in the different libraries. Full information can be found at ***http://www.lua.org/manual/5.0/manual.html#libraries***.

Maths library

math.abs
math.acos
math.asin
math.atan
math.atan2
math.ceil
math.cos
math.deg
math.exp
math.floor
math.log
math.log10
math.max
math.min
math.mod
math.pow
math.rad
math.sin
math.sqrt
math.tan
math.frexp
math.ldexp
math.random
math.randomseed

And the variable **math.pi**.

String Library

string.byte (s [, i])
string.char (i1, i2, ...)
string.dump (function)
string.find (s, pattern [, init [, plain]])
string.len (s)
string.lower (s)
string.rep (s, n)
string.sub (s, i [, j])
string.upper (s)

string.format (formatstring, e1, e2, ...)
string.gfind (s, pat)
string.gsub (s, pat, repl [, n])

Table library

table.concat (table [, sep [, i [, j]]])
table.foreach (table, f)
table.foreachi (table, f)
table.getn (table)
table.sort (table [, comp])
table.insert (table, [pos,] value)
table.remove (table [, pos])
table.setn (table, n)

I/O File manipulation library

io.close ([file])
io.flush ()
io.input ([file])
io.lines ([filename])
io.open (filename [, mode])
io.output ([file])
io.read (format1, ...)
io.tmpfile ()
io.type (obj)
io.write (value1, ...)
file:close ()
file:flush ()
file:lines ()
file:read (format1, ...)
file:seek ([whence] [, offset])
file:write (value1, ...)

Base library

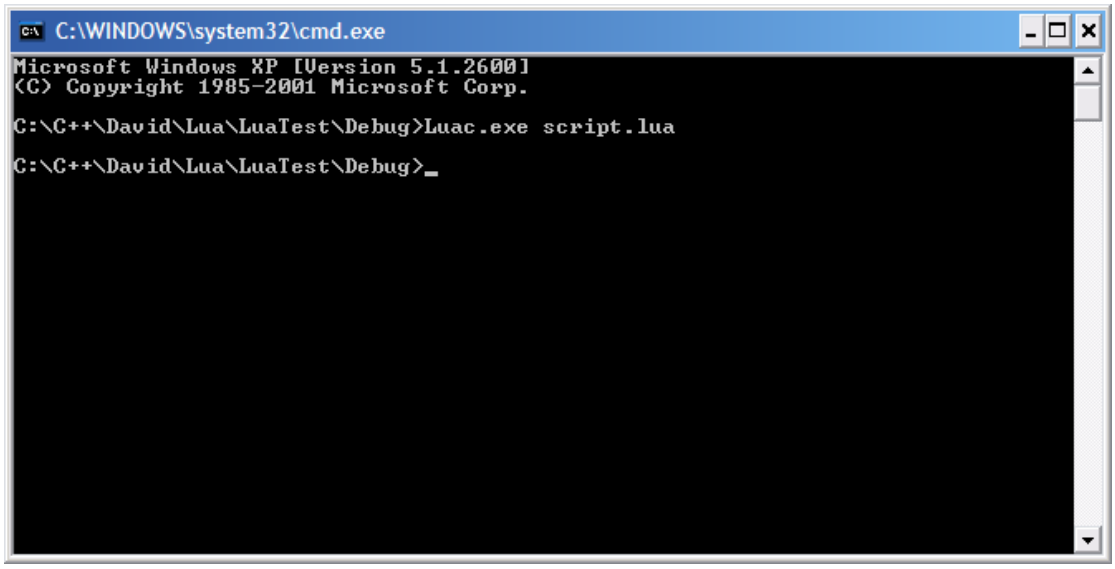
assert (v [, message])
collectgarbage ([limit])
dofile (filename)
error (message [, level])
_G
getfenv (f)
getmetatable (object)
gcinfo ()
ipairs (t)
loadfile (filename)
loadlib (libname, funcname)
loadstring (string [, chunkname])
next (table [, index])
pairs (t)
pcall (f, arg1, arg2, ...)
print (e1, e2, ...)
rawequal (v1, v2)
rawget (table, index)
rawset (table, index, value)
require (packagename)
setfenv (f, table)
setmetatable (table, metatable)
tonumber (e [, base])
tostring (e)
type (v)
unpack (list)
_VERSION
xpcall (f, err)

GameGuru operates within one Lua state.

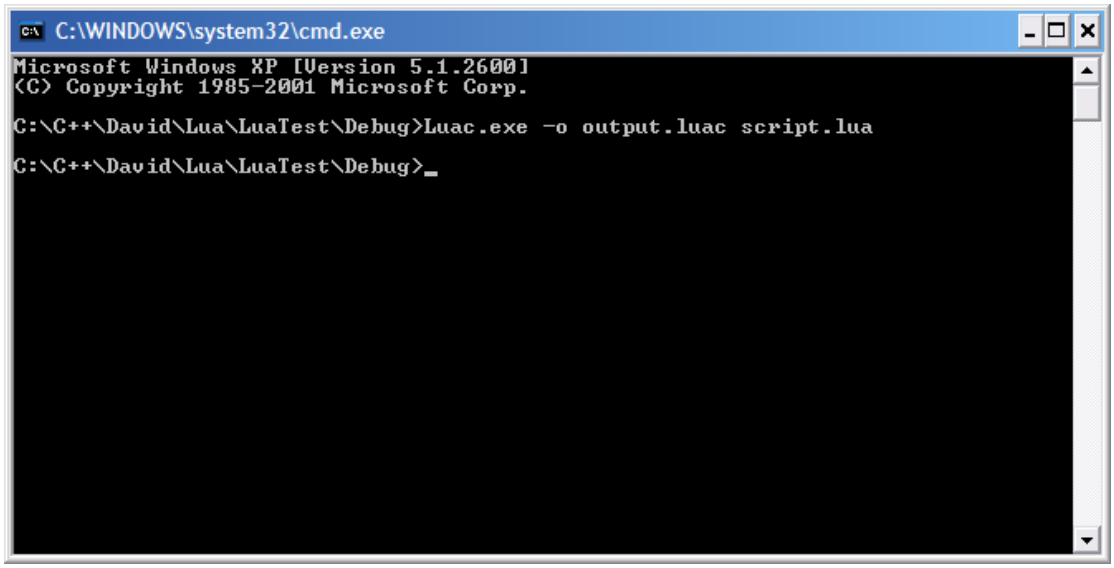
It is often useful to have your scripts in the form of text files. However, when you distribute your game, it is preferable to have all scripts in a compiled, non-human readable form. Using the Lua Compiler, “Luac”, it is possible to precompile your scripts.

The main advantages of precompiling are: faster loading, protecting source code from user changes, and off-line syntax error detection (so no errors are detected at run time).

To compile a script, simply run Luac.exe with a single argument – the file name of the script to compile:



If you don't want to call your output file output.luac, you can use the –o flag:



More information about the Lua compiler can be found on the Lua web site:
http://www.lua.org/manual/5.0/luac.html

This document is currently in draft format and will be expanded as questions about scripting in LUA are fed back from the community, and an FAQ will be added to the document once we have a sufficient number of questions relating to this topic.