Using Lua as an Embedded Language

Eric Lewis

April 2, 2018

1 What is Lua?

Lua is a powerful programming language with a focus on being flexible and lightweight. The name Lua comes from the Portuguese word for Moon. Lua allows for multiple programming paradigms and can be used as a regular language, a scripting language, an embedded language, a language for embedded systems, or for configuration files. Lua gains its flexlibility from it's usage of metatables, a mechanism that gives the programmer a large amount of control over the language. Lua provides three types to the programmer: numbers, strings, and tables. Metatables are tables that contain special functions which allow the programmer to modify behavior of a table such as by preventing the addition of new values to it.

2 Why use Lua?

Lua can be embedded into existing software to act as a scripting language or configuration system. Developers are free to decide how far they want to go in integrating Lua into their software. The developer also has the ability to restrict what Lua can do by limiting access to certain libraries and functions. This allows for tight control over scripts to prevent users from creating malicious code for example.

Lua is licensed under the MIT license. The MIT license allows the programmer to use Lua without restriction as long as Lua's software and documentation

is distributed with a copy of its license. As a result, Lua can be used in both closed source and open source projects.

3 Embedding Lua

When it comes to embedding the usage of Lua usually fits into two categories: configuration and as an embedded language. Lua can of course be used for both at the same time. Using Lua for configuration is typically easier as the programmer usually will not have to expose any interfaces to Lua. Examples for both styles include C code as it is the simplest way to use it.[1] It's important to understand certain concepts before attempting to embed Lua. Lua is a stack and register based language and interaction with Lua is done through both of these. Data must be put on and popped off the stack in order to move data between Lua and the host program. In addition the entire Lua environment is stored in a structure called the Lua state. It is possible to create multiple states, but only one is used for the examples.

It's important to note that the examples below are for Lua 5.3. Lua's version by running `lua -v` in a terminal. If Lua's version is below 5.3 then it's recommended that Lua is updated or the Lua manual is referenced for the Lua manual on the differences between versions.

3.1 Using Lua for configuration

Using Lua for configuration boils down to a few key steps. Include Lua in a program, create the Lua state, load a Lua script, and run it. The standard Lua libraries can be optionally loaded, but for basic configuration this won't be necessary.

The below example contains two files, `config.lua` and `main.c`. `config.lua` contains some simple client variables. `main.c` gets and uses these variables.[2]

[1] It's assumed that the reader knows how to compile C code and include libraries.

[2] This code is unsafe and contains no sanity checks or validation. Refer to the Lua manual on how to safely embed Lua.

Listing 1: config.lua

```
address = '1.1.1.1'
port = 8000
timeout = 3000
```

Listing 2: main.c

```
#include <lua.h>
#include <lualib.h>
#include <lauxlib.h>

#include <stdio.h>

int main(void) {
  lua_State *L = luaL_newstate(); //Create a new Lua state

  luaL_dofile(L, "config.lua"); //Open and run our config file

  //Put the variables on the Lua stack
  lua_getglobal(L, "address");
  lua_getglobal(L, "port");
  lua_getglobal(L, "timeout");

  //Get the variables from the Lua stack
  char *address = lua_tostring(L, -3);
  int port = lua_tointeger(L, -2);
  float timeout = lua_tonumber(L, -1);

  printf("Connecting to %s:%d with timeout %.1f",
         address, port, timeout);

  //...

  lua_close(L); //Close the Lua state
  return 0;
}
```