mod_lua

About
Lua is the preferred scripting language for custom applications based on FreeSWITCH.

Installation
On Centos 6.x:

```
yum install lua-devel
```

Features
Write IVR scripts in Lua
It has a very easy to use syntax, see the Hello Lua script.

Add event hooks in Lua
You can define a Lua script to be executed each time a specific event is fired. See: Mod_lua#Event_Hooks

Serve configs
Lua can serve configurations for xml_curl, without requiring a web server, this works in a similar way to mod_xml_curl.
See: Mod_lua/Serving_Configuration

Make API calls directly from Lua code
Examples

Lightweight
Stripped mod_lua.so is 272k

Highly Embeddable
As far as embeddability goes - Python ranks a 2, Perl ranks a 4, JavaScript is a 5, and Lua is a 10!

Learning Lua
Here's a comparison on some salient points to Javascript: [1]
CLI Usage: lua and luarun

You can issue a "luarun /path/to/script.lua" and launch a thread which your lua script will run in. The "lua" command is for inline lua like from the dialplan i. e. ${lua(codehere)}. "luarun" will spawn a thread while "lua" will block until the code is complete. A "~" in front of the argument will run a single line lua command. Note that lua scripts executed with luarun cannot write to the console through stream:write API as there is no stream.

Passing Arguments

Arguments are passed as space-separated values:

```
luarun arg1 arg2 arg3
```

Arguments are accessed with "argv" like this:

```
my_first_var = argv[1];
my_next_var = argv[2];
```

And so on...

```
freeswitch@DVORAK> lua ~print(string.find("1234#5678", "(%d+)#(%d+)"))
1 9 1234 5678
freeswitch@DVORAK> luarun ~print(string.find("1234#5678", "(%d+)#(%d+)"))
+OK
1 9 1234 5678
freeswitch@DVORAK> luarun ~stream:write("1234#5678")
+OK
2011-06-20 13:35:35.274782 [ERR] mod_lua.cpp:191 [string "line"]:1: attempt to index global 'stream' (a nil value)
stack traceback:
  [string "line"]:1: in main chunk
freeswitch@DVORAK> lua ~stream:write("1234#5678")
1234#5678
```

Configuring

Event Hooks

The following configuration shows how to have the tone_event.lua script executed each time the DETECTED_TONE event is fired.

```
<configuration name="lua.conf" description="LUA Configuration">
  <settings>
    <param name="script-directory" value="$${base_dir}/scripts/?.lua"/>
    <hook event="DETECTED_TONE" script="tone_event.lua"/>
    <hook event="CUSTOM" subclass="conference::maintenance" script="conference.lua"/>
  </settings>
</configuration>
```

Event Hook Script

This is an example event hook script. getHeader() can be called on the event to get information about the event.

```
local uuid = event:getHeader("Unique-ID")
local tone = event:getHeader("Detected-Tone")
freeswitch.consoleLog("info", uuid .. " detected tone: ", tone .. "\n")
```

For IVR use

Nothing should be needed here.
For making API calls

```lua
api = freeswitch.API();
digits = api:execute("regex", "testing1234|/(\d+)/|$1");
-- The returned output of the API call is stored in the variable if you need it.
freeswitch.consoleLog("info", "Extracted digits: " .. digits .. "\n")
```

**Note:** Please take care to escape the arguments that you pass, as has been done to the regex string above.

To call another Lua script

```lua
api = freeswitch.API();
reply = api:executeString("luarun another.lua");
```

For serving configuration

**mod_lua** allows you to replace request for configuration data from a lookup in the static XML to your script.

See [Mod_lua/Serving_Configuration](#) for more information.

Lua scripts at startup

Here is a minimum configuration file:

```xml
<configuration name="lua.conf" description="LUA Configuration">
  <settings>
    <!-- The following options identifies a lua script that is launched at startup and may live forever in the background. You can define multiple lines, one for each script you need to run. -->
    <param name="startup-script" value="startup_script_1.lua"/>
    <param name="startup-script" value="startup_script_2.lua"/>
  </settings>
</configuration>
```

The start-up script values represent lua scripts (located inside the scripts/ directory) that are launched when FreeSWITCH is started. The scripts live in their own thread. You can use them to run simple tasks (and then let them finish) or looping forever, watching (for example) for events, generating calls or whatever you like.

Sample Dialplan

```xml
<action application="lua" data="helloworld.lua arg1 arg2"/>
```

**NOTE:** arguments can be accessed by using argv[1] argv[2] in your script

**NOTE:** for looking up the location of the helloworld.lua file, it looks in prefix/scripts by default

In-line expansion
Sample IVR's

This is a basic IVR example, in which we answer the call, wait 1 second and play an wav audio file.

```lua
-- answer the call
session:answer();

-- sleep a second
session:sleep(1000);

-- play a file
session:streamFile("/path/to/blah.wav");

-- hangup
session:hangup();
```

Pattern matching (regular expressions)

Regex API Example

Using this method, you can execute regex conditions from inside your lua scripts.

```lua
session:execute("set", "some_chan_variable=${regex(" .. destination .. ")}|([0-9]{10})$\")
```

If destination is a lua variable with your destination number in it, and your regex was `^[0-9]{10}$`
the result will be put in "some_chan_variable" that you can get thorough a `session:getVariable`

You can also do:

```lua
session:execute("set", "some_chan_variable=${regex(" .. destination .. ")}|(061)([2,3,7,8][0-9]{8})|2$2")
```

To match and return parts of your regex.

Native Lua Pattern Matching

Lua supports a simple but powerful pattern matching syntax. It is less powerful than PCRE but it handles the majority of pattern matching cases will ever need in a simple script.

The following is a simple script that can be run with "luarun" from fs_cli and demonstrates the capturing of two values from a data string:
Misc. Samples

Run a shell command

When using `session:execute()` and `api:execute()` to execute a shell command (ie: bash script) it will only return the error code integer.

To return the output of a shell command use `io.popen()`. The following example Lua function is from [http://lua-users.org/wiki/ShellAccess](http://lua-users.org/wiki/ShellAccess).

```lua
function shell(c)
  local o, h
  h = assert(io.popen(c,"r"))
  o = h:read("*all")
  h:close()
  return o
end
```

FAQ

Why is mod_lua not unloadable?

```bash
freeswitch@baremetal> reload mod_lua
+OK Reloading XML
-ERR unloading module [Module is not unloadable]
-ERR loading module [Module already loaded]
```

Some threads explaining why the decision was made [http://freeswitch-users.2379917.n2.nabble.com/Reload-mod-lua-td7586568.html](http://freeswitch-users.2379917.n2.nabble.com/Reload-mod-lua-td7586568.html)

mod_lua was reloadable but when I found that long-running scripts cannot be easily stopped the developers changed it to un-reloadable.


{...truncated thread...}

The reason being if the module unloads while scripts are still running you're going to be almost guaranteed a segmentation fault – a well meaning admin running a simple command could crash the entire server dropping all calls.

Where is my debug information?
Q: When I use static XML or xml_curl, I see the commands executed in the fs_cli and in the "application log" section of the xml_cdr file. However, when I run commands via my lua script, I don't see that information anywhere?

A: When you have an actual call session, you can use session:execute("$application","$data") just like in the static XML - it will then show up in the fs_cli and the xml_cdr application log. If you don't have a call session - e.g. if you are running lua as a background application or from the CLI, then you have to use other commands which may not be as easily logged.

How can I make it use the "system lua"

Q: I have Lua installed, but mod_lua seems to ignore the Lua binary.

A: Lua is so small that the whole ball of wax is statically linked into the module!

How can I get Lua to see my own libraries using "require"

Q: Can I use the require mechanism for including libraries with the Lua in FreeSWITCH?

A: You may need to alter the LUA_PATH variable to instruct the embedded Lua inside FreeSWITCH to find your libraries. A simple startup script to do this is:

```
#!/bin/bash
export LUA_PATH=/usr/local/freeswitch/scripts/?.lua;\;
ulimit -s 240
screen /usr/local/freeswitch/bin/freeswitch
```

The default path is something like /usr/local/share/lua/5.1/

Another option for common code similar to the "include" directive in many languages is to use dofile

eg. dofile("/home/fred/scripts/fredLuaFunctions.lua")

Note that this will just execute the code contained in the file as if it were inline - this is not the same as creating a lua module

How can I find useful undocumented Functions?

There may be times when a function gets added, but not documented. This simple Lua script may help.

```
-- This function simply tells us what function are available in Session
-- It just prints a list of all functions. We may be able to find functions
-- that have not yet been documented but are useful. I did :
function printSessionFunctions( session )
   metatbl = getmetatable(session)
   if not metatbl then return nil end
   local f=metatable['.fn'] -- gets the functions table
   if not f then return nil end
   for k,v in pairs(f) do stream:write(k.."\n") end
   stream:write("\nEND\n")
end
new_session = freeswitch.Session() -- create a blank session
stream:write("\n***Session Functions***\n")
printSessionFunctions(new_session)
new_event = freeswitch.Event("message_waiting");
stream:write("\n***Event Functions***\n")
printSessionFunctions(new_event)
new_api = freeswitch.API();
stream:write("\n***API Functions***\n")
printSessionFunctions(new_api)
```

At freeswitch version FreeSWITCH Version 1.6.17-34-0fc0946-~64bit (-34-0fc0946 64bit) this is the output of the script:

```
***Session Functions***
playAndGetDigits
speak
setDTMFCallback
answer
mediaReady
```
**API Functions**

- `__disown`
- `getVariable`
- `collectDigits`
- `streamFile`
- `setEventData`
- `sleep`
- `flushDigits`
- `setLUA`
- `setHangupHook`
- `setInputCallback`
- `unsetInputCallback`
- `preAnswer`
- `consoleLog2`
- `consoleLog`
- `run_dtmf_callback`
- `say`
- `process_callback_result`
- `execute`
- `get_cb_args`
- `waitForAnswer`
- `getXMLCDR`
- `flushEvents`
- `set_tts_parms`
- `end_allow_threads`
- `sendEvent`
- `getState`
- `begin_allow_threads`
- `get_uuid`
- `setAutoHangup`
- `getDigits`
- `getPrivate`
- `bridged`
- `recordFile`
- `destroy`
- `set_tts_params`
- `ready`
- `transfer`
- `hangupCause`
- `insertFile`
- `originate`
- `sayPhrase`
- `answered`
- `read`
- `hangupState`
- `hangup`
- `check_hangup_hook`
- `setPrivate`
- `setVariable`

**Event Functions**

- `setPriority`
- `chat_send`
- `serialize`
- `getHeader`
- `getBody`
- `fire`
- `delHeader`
- `__disown`
- `getType`
- `addBody`
- `chat_execute`
- `addHeader`

---

**API Functions**

- `__disown`
- `executeString`

---

**Event Functions**

- `setPriority`
- `chat_send`
- `serialize`
- `getHeader`
- `getBody`
- `fire`
- `delHeader`
- `__disown`
- `getType`
- `addBody`
- `chat_execute`
- `addHeader`

---

**API Functions**

- `__disown`
- `execute`

---

**Event Functions**

- `setPriority`
- `chat_send`
- `serialize`
- `getHeader`
- `getBody`
- `fire`
- `delHeader`
- `__disown`
- `getType`
- `addBody`
- `chat_execute`
- `addHeader`

---

**API Functions**

- `__disown`
- `execute`

---

**Event Functions**

- `setPriority`
- `chat_send`
- `serialize`
- `getHeader`
- `getBody`
- `fire`
- `delHeader`
- `__disown`
- `getType`
- `addBody`
- `chat_execute`
- `addHeader`

---

**API Functions**

- `__disown`
- `executeString`
Jester Lua Toolkit

Jester is a scripting lua toolkit for FreeSWITCH :Wiki Jester

See Also

- FreeSWITCH Lua API Reference
- Lua.org Web site
- Lua Install dependencies
- Script Language Choice
- FreeSWITCH book pages 149-151
- http://www.lua.org/manual/5.1/manual.html#5.4.1