LuaPLC

Introduction

Raritan Inc.

October 18, 2016

Contents

1 LuaPLC briefly

Lua - A powerful and fast scripting language, simple to learn and use, easy to embed into an application

PLC - Programmable Logic Controller, a system that has a firmware and this one can execute PLC application

The LuaPLC contains two parts. One part is a service to manage Lua scripts on a PDU. It is called LuaService. With this service you can up/download, start/terminate a script and get the output if a script produces one. The other part is the Lua-Binding. The glue to combine Lua and PDU libraries. With this binding you can control the whole PDU and also a remote PDU.

2 Quickstart

2.1 Requirements

- A firmware with LuaPLC support
- Our latest JSON-RPC SDK
- Python 2.7 to run luaservice_client.py, a python script to handle the LuaService

2.2 Upload and Start a Lua Script

Download the JSON-RPC SDK and unpack it. Open a console and change to the JSON-RPC SDK directory.

Next we are going to upload a Lua script with name *get_info.lua*. It's located at the folder *LuaPLC_Examples/basic/*. To upload and start the script the option *-r* is used. The whole command looks like this: **./luaservice_client.py -a 10.0.42.10 -u username -p password -r LuaPLC_Examples/basic/get_info.lua**. Correct the ip-address and username + password. Listing 1 shows the generated ouput. Line 1 is the command to execute, lines 2 to 7 are output from the luaservice_client.py script and starting at line 8 is the output of the Lua get_info.lua script. To exit from the luaservice_client.py script press **Ctrl + c**.

Listing 1: Run Lua Script Example get_info.lua

```
1 $ ./luaservice_client.py -a 10.0.42.10 -u admin -p raritan -
    r LuaPLC_Examples/basic/get_info.lua
```

```
2 agent created: url https://10.0.42.10:443
3 name get_info
4   starting get_info returns: no error
5   print output for script get_info
6     refresh interval is every second
7     to abort press Crtl-C
8 -- pdu metadata
9 ...
```

3 LuaService Daemon

LuaService is a daemon that runs on the PDU. The service provides following:

- Upload a Lua script
- Download a Lua scirpt
- Delete or overwrite an uploaded script
- Start a Lua script
- Terminate a Lua script
- Set diffrent options
  - To start a Lua script after system start
  - To restart a script after termination or crash
  - Set command line arguments to control the script
- To get the output from a script if it puts something out
- Get the last state from a script (new, terminated, running)

The LuaService has an IDL file named *LuaService.idl*. There are JSON-RPC and Lua-Bidings.

The script luaservice_client.py uses the Python JSON-RPC-Bindings. It's a command line tool to control the LuaService daemon.

3.1 API Documentation

The full documentation for LuaService can be found at our *JSON-RPC SDK*. To find the documentation download the sdk (software development kit) from our homepage. After downloading open the file *index.html* from the folder *html*. Click on the site *Namespace* the entry *luaservice* to get the LuaService documentation site.

4 USB Flash Device and Tftp Server

There is an additional way to start a Lua script. It's possible to run a script from USB mass storage or tftp server. The **fwupdate.cfg** key is **execute_lua_script** and the value is the script name. USB flash device examples are located at LuaPLC_Examples/usb_fwupdate.

Lua script output from USB flash device or tftp server will be stored on an extra log file (limited size on dhcp/tftp). A dhcp/tftp located script has a timeout of 60 seconds. After that duration the script will be killed. If you unplug the USB flash device and the script is still running then the Lua script will be killed. An exit handler can be used but the execution time is limited to three seconds.

5 Lua

Lua is easy to learn and simple to use scripting language. See the links section to find about more the language itself.

The Lua-Binding is just another binding like the Python binding or Perl binding. All bindings are generated from IDL files. That means you can controll more or less the whole pdu with this binding. In addition there is a build-in JSON-RPC client, to controll a remote PDU.

5.1 Limitation

There are some Lua script limitations:

- Memory per script
- Script size
- Amount of scripts

- Script speed

You can get the limitations from the LuaService daemon. Look for the function **getEnvironment()** at API documentation.

To limit cpu usage all Lua scripts run with low priority.

5.2 Implementation Details

5.2.1 Mapping

Table 1 lists the IDL Lua type mapping.

The code listing 2 shows how access an enumeration in Lua.

Lua can return multiple values. If there more than one return/out value the IDL Lua method mapping returns all. Listing 3 shows the method mapping. In parameters can be used as usual.

Table 1: IDL Lua Type Mapping

IDL	Lua
boolean	boolean
int	number
long	number
float	number
double	number
string	string
time	number
enumeration	variable defined at the global table
structure	table
vector	table
map	table
interface	table

Listing 2: Enumeration Type Matching, Example

```
-- IDL file snippet:
--   module foo {
--     enumeration bar { zero, one };
--   };
```

```
-- enumeration in Lua
myEnum0 = foo.bar.zero
myEnum1 = foo.bar.one
```

Listing 3: Lua Method Mapping

```
-- defined at luaservice.idl:
--   int luaservice::Manager::getScript(
--     in string name,
--     out string script
--   )

-- on in parameter ('in string name')
-- multiple returns in Lua ('int' and 'out string script')
status, script = myManager:getScript("name of the script")
```

5.2.2 Load a Lua Library

The **require()** function is used to load a library. Typicall you type **require "Pdu"** to load the pdu library and all libraries that PDU library depends on.

If you want to use the event service you have to load the event service library with **require "EventService"**. If you want to use LuaService you have to require **LuaService**.

5.2.3 Command Line Arguments

LuaPLC supports a feature called command line arguments. You can pass arguments to your script if you start a script with the **startScriptWithArgs** json-rpc-command or use the script option **defaultArgs**. The command line arguments are *key - value* based. You get the arguments in Lua from the global table **ARGS**. Listing 4 shows you how to get the values for the keys *outlet* and *delay*. Both, key and value, are strings.

Listing 4: Get Command Line Arguments

```
paramOutlet = ARGS["outlet"]   -- value for key outlet
paramDelay = ARGS["delay"]     -- value for key delay
```

5.2.4 Exit Handler

The function ExitHandler() (see listing 5) will executed when the script unexpected stops/crashes. That can happen in some cases (script will be terminated from

outside, programming failure, ...). Best practice is to write this function at the top (after require statement) of the script and do not use global variables.

Listing 5: Exit Handler Usage

```
require "Pdu"

-- my special exit handler
function ExitHandler()
  print("i'm going to exit")
end
```

5.2.5 Sleep

The build-in sleep function let the script pause for a moment or more (Listing 6).

Listing 6: Simple Lua Sleep Function Demonstration

```
sleep(2) -- a 2 second pause
```

5.2.6 Exception Handling

To execute a function with the given arguments in procteced mode use the function **pcall(f [, arg1, ...])**. You can use this function to catch exceptions. Listing 7 shows you two examples how this works and example how it works not. To get more about **pcall()** take a look at Lua reference.

Listing 7: Correct Use of pcall() Example

```
-- catching works correct
err, errMsg = pcall(function () o3:setSettings(z) end)
err, errMsg = pcall(o3.setSettings, o3, z)

-- catching works NOT correct
err, errMsg = pcall(o3:setSettings(z))

-- test if error happens
if err == false then
  print("error catched: " ..  errMsg)
else
  print "no failure catched"
end
```

5.2.7 Local and Remote PDU

Listing 8 shows you three ways to create a PDU object. **pdu1** and **pdu2** reprensents a local PDU and **pdu3** represents a remote PDU.

getDefault wraps the method **new(tfwSocketPath, "pdu.0")**. **tfwSocketPath** is a path and **pdu.0** means the first pdu. If you have a PDU with more 'PDU's inside than **pdu.1** points to the second pdu. Typically you use the **getDefault()** method to create a PDU object.

If you want to create a remote PDU object (like **pdu3**) you need to create a http agent object first. The signature to create a new agent is **agent.HttpAgent:new (host, user, password [, port [, useTls]])**. **host**, **user** and **password** are strings. The fourth argument is **port**. It's optional and a number. Default is the port 80. The last argument **useTls** is also optional and a boolean. Default is false and if set to true a secure http call (https) will be used.

The method **newRemote(Well-know URI, agent)** creates a new remote pdu object. All **Well-know URI**'s are documented at our JSON-RPC-SDK.

Listing 8: Three Ways to Create a PDU Object

```
pdu1 = pdumodel.Pdu:getDefault()
pdu2 = pdumodel.Pdu:new(tfwSocketPath, "pdu.0")

agent = agent.HttpAgent:new("myAddress", "user", "password")
pdu3  = pdumodel.Pdu:newRemote("/model/pdu/0", agent)
```

5.2.8 Out of Memory

Memory per script is limited. If a script allocates to much memory the script will be killed by the system. Then a message like **LuaSvcScript: Out of Memory. Aborting ...** is to read on the output.

Lua has an garbage collector. You don't have to worry about allocation and deallocation memory. To perform a full garbage-collection cycle call **collectgarbage("collect")** in your Lua script. If your script creates to much new objects in a short time then calling a gargabe-collection cycle could prevent a *out of memory* situation.

5.3 Links

Some Lua links:

http://www.lua.org
: The offical Lua homepage. You can find there the reference manual.

http://www.lua.org/pil/contents.html
: The online version of the book Programming in Lua (third edition).

http://tylerneylon.com/a/learn-lua/
: Lern Lua in 15 minutes - more or less. To program with LuaPLC you need to know sections 1, 2 and 3. Sections 3.1 and 3.2 are nice to know.

https://en.wikipedia.org/wiki/Lua_(programming_language)
: Lua article at the Wikipedia site.