

21 Lua script runner

Introduction

Lua is a powerful, efficient, lightweight scripting language. It has been used in many industrial applications with an emphasis on embedded systems. Lua has a deserved reputation for performance. To claim to be "as fast as Lua" is an aspiration of other scripting languages. Several benchmarks show Lua as the fastest language in the realm of interpreted scripting languages. Lua is fast not only in fine-tuned benchmark programs but in real life too. Substantial fractions of large applications have been written in Lua.

In the WCC Lite system, Lua is used for extending the functionality of Excel configuration adding an interface to the existing signal-linking engine. Provided functions enable to recreate of PLC functionality and modify any value with ease.

Overview

Execution types:

Lua runner provides 3 execution modes: interval, date, and signal, which can be specified in **execution_type**.

Interval: executes provided script based on provided time interval in **execution_parameter**. It uses milliseconds, meaning if a 500 value is provided, then the script is executed every 500 milliseconds.

Date: schedules a script execution based on the provided cron expression in **execution_parameter**. The format consists of 6-7 fields separated by space.

Name	Required	Allowed Values	Allowed Special Characters
Seconds	Y	0-59	, - * /
Minutes	Y	0-59	, - * /
Hours	Y	0-23	, - * /
Day of month	Y	1-31	, - * /
Month	Y	0-11 or JAN-DEC	, - * /
Day of week	Y	1-7 or SUN-SAT	, - * /
Year	N	empty or 1970-2099	, - * /

For example, 0 0 * * * * will execute the script at every hour mark. There are a lot of online cron expression parsers or generators to convert this expression to a more understandable sentence. <https://crontab.cronhub.io/>

Signal: uses source signal provided in signal sheet to trigger script execution. Another non-example signal can be provided in the **execution_parameter** attribute. As an example, if a signal in Excel configuration has an attribute **execute** with value 1, and a source signal specified in **source_device_alias**, **source_signal_alias**, then if a value event happens to the source signal, the script will be executed. More than one signal can have an **execute** attribute. The signal that executes the script can be accessed through the **execution_signal** variable that has a table inside of it with variables: {tag = {device_alias = "script", signal_alias = "tag1"}, value = {value = 60, time = "123456789", attributes = "iv, nt, sb"}}

Additional functions:

To interface with existing signals and extend the available Lua functions some extra functions were added:

get, **set**, **publish**, and **subscribe** functions are used to get the values configured in Excel configuration or to communicate with other scripts.

A signal value in the WCC Lite system consists of 3 sub-values: **value**, **time**, and **attributes**. **get** function is used to get all the sub-values or **get_value**, **get_time** and **get_attributes** to get only one of the sub-values. To execute this command, a signal has to be specified: **get(signal.value1)**. A signal is specified by a string "tag/<device_alias>/<signal_alias>" or a table that is created from an **iterator** parameter in Excel configuration. For example:

```
-- the default iterator is signals, that means there is a table 'signals' generated from excel signals
-- signals["tag1"] = "tag/device_alias/tag1" tag1 would be the signal_alias
-- to get the value of this signal:
local variable = get_value(signals.tag1)
local variable = get_value(signals["tag1"]) -- both methods are viable
-- and the 'variable' will have the value of tag1 signal
-- get function will return the value of source signal that was sent to this signal
```

Function **publish** is used to send a value to other signals:

```
publish(signals.tag2, 90) -- this function will send 90 with current time to the signal tag2
publish(signals.tag2, {value = 60, time = "123456789", attributes = "iv,nt,sb"})
-- above command is used when other sub values are needed to be specified
-- and the signal-linker will send it to another signal
-- if this signal was specified as source signal in another protocol signal
```

To provide different times and attributes to the publish function, a table of these 3 values has to be specified, time can be omitted. An example of the table is returned by the **get** function.

These two functions will be used the most, others are included for communication between different scripts in a more complex system.

set function sets the value of the signal, without sending an event of change:

```
set(signals.tag3, 50) -- this command will set the value of the signal tag3
-- because it is a set function, other protocols will not see a change
-- and the value will not be accessible
-- it is only used with non excel signals
set("signal1", 60) -- now the signal1 tag will have a value
-- and another script will be able to use get to get this value
local value1 = get_value("signal1") -- value1 will be equal to 60 with current time
```

Function **subscribe** is used to wait (blocks code execution while waiting) for a value and get it. It is used the same as the **get** function (does not have separate functions for sub-values). It should only be used if more complex communication between scripts is needed.

 These functions are equivalent to REDIS functions.

Function **save** saves the specified value to flash memory for use after reboot. The same value sub-values apply to execution.

```
save(signals.tag1, 50) -- this will save a value to tag1
-- after reboot this value will be set to tag1
-- and will be accesible with get(signals.tag1)
-- this function will not set the value tag1 to 50
```

Function **time_ms** returns the current time in milliseconds and in UNIX format.


```
local t = time_ms()
-- t will be equal to 1665389490555
-- if the date right now is Mon Oct 10 2022 08:11:30:555 GMT+0000
```

Function **sleep** is the same as Lua socket module function `socket.sleep`.

```
sleep(1) -- will wait for 1 second
```

Additional functions for debug information are **emerg**, **alert**, **crit**, **err**, **warning**, **notice**, **info**, and **debug**. These functions correspond to levels from 0 to 7. The default level is 4, which means that the function from **emerg** to **warning** will be printed to syslog unless specified differently in file `/etc/lua-runner.conf`.

```
emerg("log message 0")
alert("log message 1")
crit("log message 2")
err("log message 3")
warning("log message 4")
notice("log message 5")
info("log message 6")
debug("log message 7")
```

 These functions will require significant CPU resources even if the message log level is higher than default and no message is printed.

Web interface

The web interface is used to see what scripts are running and, if there is a script provided, to stop/start a script. After

configuring a device with Lua runner as protocol, the script runner protocol hub tab will be populated with devices that were configured:

PROTOCOL HUB

STATUS

SYSTEM

SERVICES

NETWORK

USERS

LOGOUT (ROOT)

CONFIGURATIONIMPORTED SIGNALSEVENT LOGPROTOCOL CONNECTIONSSCRIPT-RUNNER

Script-Runner

LUA SCRIPT INSTANCE CONTROL

Script Configuration	Script process	Status	Script File		
Device_1	-	Stopped	No Script provided	Upload Script	Waiting for script
Device_2	-	Stopped	No Script provided	Upload Script	Waiting for script
Device_3	-	Stopped	No Script provided	Upload Script	Waiting for script
Device_4	-	Stopped	No Script provided	Upload Script	Waiting for script

SAVED VALUE CLEARING

Clear all saved values

Then by pressing the Upload Script button, a script will be available to be selected (the name of the script will be changed to match device_alias). When uploaded the script will not be started automatically, pressing start will be necessary.

PROTOCOL HUB

STATUS

SYSTEM

SERVICES

NETWORK

USERS

LOGOUT (ROOT)

CONFIGURATIONIMPORTED SIGNALSEVENT LOGPROTOCOL CONNECTIONSSCRIPT-RUNNER

Script-Runner

LUA SCRIPT INSTANCE CONTROL

Script Configuration	Script process	Status	Script File		
Device_1	-	Stopped	Device_1.lua	Upload Script	Start
Device_2	-	Stopped	No Script provided	Upload Script	Waiting for script
Device_3	-	Stopped	No Script provided	Upload Script	Waiting for script
Device_4	-	Stopped	No Script provided	Upload Script	Waiting for script

SAVED VALUE CLEARING

Clear all saved values

After pressing start, the script will be started, if there are errors it will try to start, but after a few attempts, it will stop.

PROTOCOL HUB

STATUS

SYSTEM

SERVICES

NETWORK

USERS

LOGOUT (ROOT)

CONFIGURATION

IMPORTED SIGNALS

EVENT LOG

PROTOCOL CONNECTIONS

SCRIPT-RUNNER

Script-Runner

LUA SCRIPT INSTANCE CONTROL

Script Configuration	Script process	Status	Script File		
Device_1	12136	Running	Device_1.lua	Upload Script	Stop
Device_2	-	Stopped	No Script provided	Upload Script	Waiting for script
Device_3	-	Stopped	No Script provided	Upload Script	Waiting for script
Device_4	-	Stopped	No Script provided	Upload Script	Waiting for script

SAVED VALUE CLEARING

Clear all saved values

The clear all saved values button is used to clear the memory of saved values. Having a lot of values saved is not healthy for the SD card and faults can happen. Also, the script runner process initialization is slowed down when a lot of saved values are used.

Configuration

Device configuration parameters:

Parameter	Type	Description	Required	Default value (when not specified)	Range	
					Min	Max
name	string	User-friendly name for a device	Yes			
device_alias	string	Alphanumeric string to identify a device	Yes			
description	string	Description of a device	No			
enable	boolean	Enabling/disabling of a device	No	1	0	1
protocol	string	Protocol to be used	Yes		lua runner	
execution_type	string	Execution type to be used	Yes		interval, date, signal	
execution_parameter	int, string	Parameters for execution	Yes	NULL	interval time in ms, date in cron format, additional signal	
queue_max	int	Maximum execution queue jobs	No	3	0 to disable the queue	
error_limit	int	Error limit before stopping	No	3	0 to disable	

keep_alive_time_ms	int	Time to keep the script alive in milliseconds	No	600000	0 to disable
--------------------	-----	---	----	--------	--------------

Signals configuration parameters:

Parameter	Type	Description	Required	Default value (when not specified)	Range	
					Min	Max
signal_name	string	User-friendly name for a signal	Yes			
device_alias	string	Alphanumeric string to identify a device	Yes		Must match device_alias in the device sheet	
signal_alias	string	Unique alphanumeric name of the signal to be Yes used	Yes			
iterator	string	Lua table name to which signal is added	No	signals		
default_value	string	Default value for a signal	No			
execute	int	Enable signal update trigger to execute script (only available for signal execution mode)	No	0	0	1

Debugging the script runner service

If the configuration for the script runner is set up, the process will start automatically. If the configuration/script is missing or contains errors, the process will not start. It is done intentionally to decrease unnecessary memory usage.

Script runner runs a service called **lua-runner**. If the script doesn't start or does not work correctly, a user can launch a debug session from the command line interface and find out what problem is causing it to not work. To launch a debugging session, a user should stop the script from the web interface and run the **lua-runner** command with respective flags and configuration as in the table given below.

Procedure for lua-runner service debugging:

- **Step 1:** The script must be stopped through a web interface.
- **Step 2:** After the script is stopped it must be started with the preferred configuration file (JSON files found in /etc/lua-runner, and the name corresponds to device_alias) and a debug level 7:
lua-runner -c /etc/lua-runner/<device alias.json> -f /etc/lua-runner/scripts/ <file_name.lua> -d7 -e
 - for example: lua-runner -c /etc/lua-runner/lua_output_control.json -f /etc/lua-runner/scripts/lua_output_control_iec103 -d7 -e
- **Step 3:** Once the problem is diagnosed normal operations can be resumed by starting the script through a web interface.