Lua SPOT Documentation

Release 1.0

Fajran Iman Rusadi, Zheng Zhangzheng

March 26, 2009

CONTENTS

CHAPTER

ONE

INTRODUCTION

Lua SPOT is a script execution framework on Sun SPOT, a wireless sensor network. It uses a tiny Lua virtual machine that is written in Java, Kahlua.

Lua SPOT allows you to develop applications in Lua scripting language and run the applications on the Sun SPOT. Each application contains one or more function that can be invoked by sending a message in a certain format to the Sun SPOT. Therefore, we can see the Sun SPOT as a service provider that contains services which can be executed remotely.

This Lua SPOT package contains two applications. The first one is *Lua SPOT* itself which will be installed on the Sun SPOT. The second one is *LuaDeskSpot* which is a desktop application that can be used to send messages to the Sun SPOT and do a certain task that shows the service invocation.

CHAPTER

TWO

LUA SPOT

2.1 Compilation

To compile Lua SPOT, go to the Lua SPOT project directory and make the application suite by entering `ant suite` command as follows:

```
$ cd src/LuaSpot
$ ant suite
```

After executing that, under the `suite` directory, you can find a file called `LuaSpot_1.0.0.jar`.

2.2 Deployment

To install the application to the Sun SPOT, run `ant deploy` in the project directory. This will compile and install the suite to the Sun SPOT.:

```
$ ant deploy
```

2.3 Execution

After the deployment, the Lua SPOT is ready to be used. Reset the Sun SPOT and after some time, you can see the LEDs turn to red. It tells that the Lua SPOT is initializing. If the LEDs turn to green and off after that, it means that the Lua SPOT in the Sun SPOT is ready to be used.

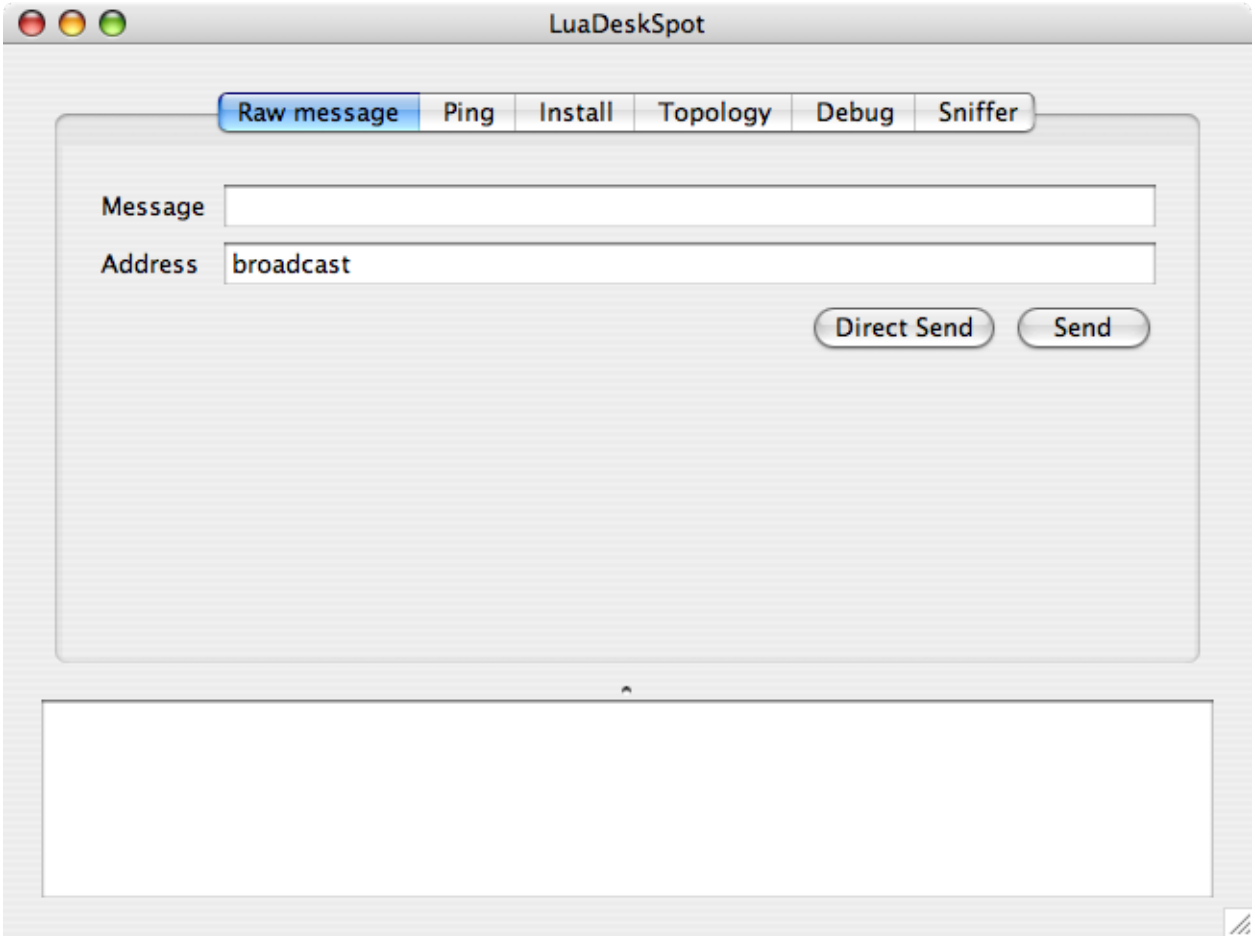
CHAPTER

THREE

LUADESKSPOT

LuaDeskSpot is an example application that contains several modules that call services in the Lua SPOT. One of the module is the script installation module that will call installation service in the Lua SPOT. You can use this installation module to install new Lua scripts to the Lua SPOT.

You will need the Sun SPOT SDK and Java 5 in order to run this application.



3.1 Compilation

To compile LuaDeskSpot, go to the project directory and run `ant host-compile`.:

```
$ cd src/LuaDeskSpot
$ ant host-compile
```

3.2 Execution

One of the way for using base station is by setting the base station to run in shared mode. Edit the `.sunspot.properties` under your home directory and insert two line like the following.:

```
basestation.shared=true
multi.process.basestation.sharing=true
```

After that, you can start using the host application by running it.:

```
$ cd src/LuaDeskSpot
$ ant host-run
```

3.3 LuaDeskSpot Modules

3.3.1 Raw Message Module

This module is used to send raw message to the Lua SPOT. The the message in the Message text box, specify the address in the Address text box, and click the Send or Direct Send button.

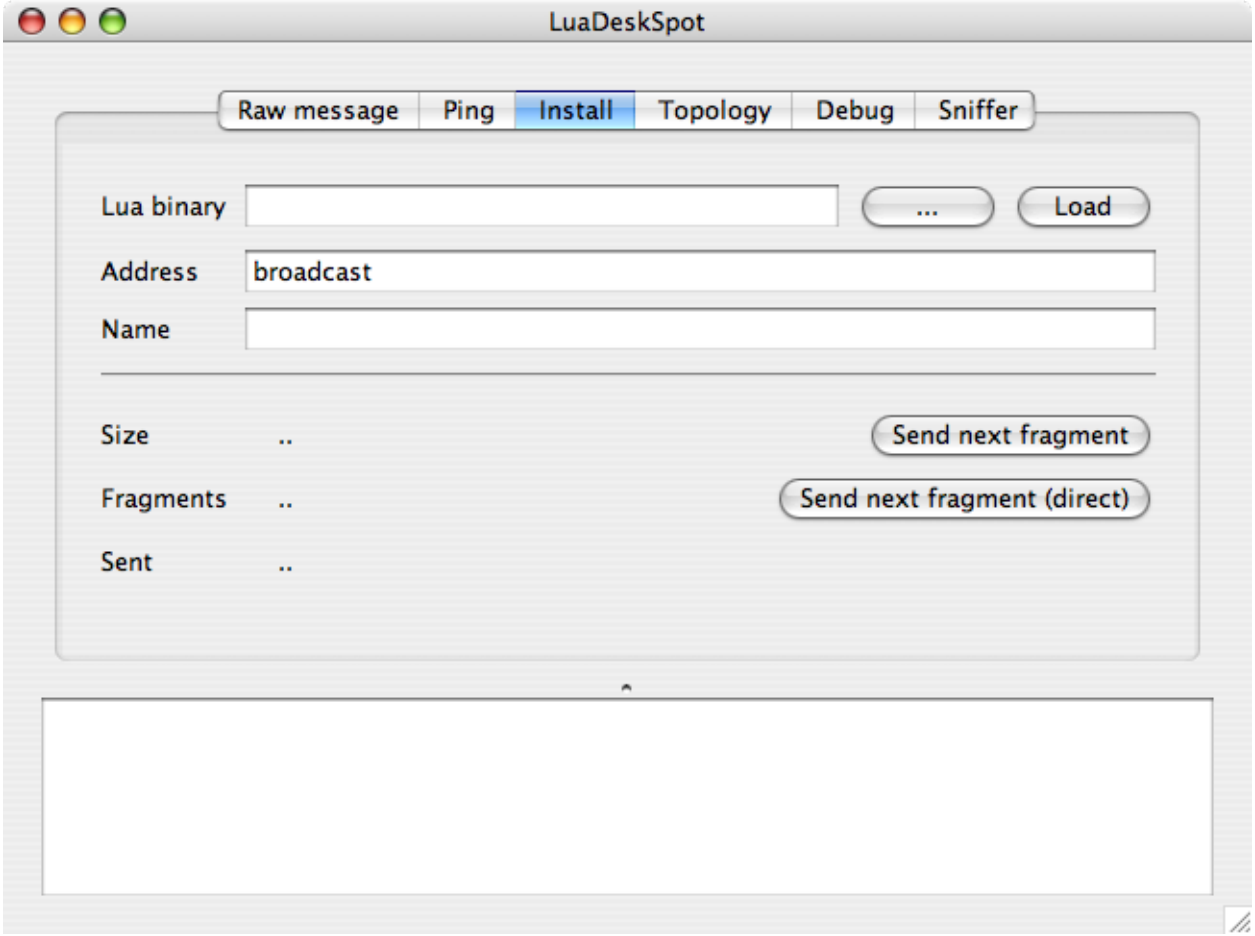
The Send button is used to send the message by calling the routing function in the Lua SPOTs. If you want to send a raw message without using the routing function, click the Direct Send button instead.

3.3.2 Ping Module

This module is used to call `ping` function of `demo` application that is pre-installed on the Lua SPOT. Any node that receive this message will send a response `pong` to the sender node. You can see the nodes that respond to the ping message using this module.

3.3.3 Install Module

This module is used to install new script to the Lua SPOTs.



First, select your compiled script file by typing the path or clicking the `...` button. Click the `Load` button after that to load the file into memory. You can also see the size and number of fragments that need to be sent.

After specifiying the destination address and the application name, you can start sending the install message (including the fragmented application binary) by clicking the Send Next Fragment button (or the direct send one). Do this multiple times until all fragments are sent.

The receiving Lua SPOT should turn on a LED when it receives and sucessfully installed the fragmented application. All LEDs will be turned on with blue color if the last segment is received.

3.3.4 Topology Module

This module can be used to get the network topology. It also call the `demo` application and tell the nodes to exchange messages so all nodes will know their neighbors. This neighbor information is sent back to the host so the host can know the overall topology. The topology graph will be drawn in the application.

3.3.5 Debug Module

This module shows debug messages that are sent by the Lua SPOT.

3.3.6 Sniffer Module

This module will show all messages that are received by the application.

CHAPTER

FOUR

SCRIPTING REFERENCE

Once the Lua SPOT is ready and deployed, you can add new applications to it by writing a Lua script. The script script is need to be compiled before you can actually send it to the Lua SPOT.

4.1 Lua Installation

Lua 5.1.4 is needed to compile the Lua scripts that are going to be installed on the Lua SPOT. The Lua source code can be downloaded from http://www.lua.org/ftp/lua-5.1.4.tar.gz. Follow the installation instruction inside the source code archive to install Lua on your machine.

4.2 Writing Lua Script for Lua SPOT

Before writing the script, you have to determine a name for your script. This name will also be your script identifier when installing and calling the script from the Lua SPOT.

The script should be written according to the following standard.

```
appname = {
   func1 = function()
      -- your code here
   end,

   func2 = function()
      -- your other code here
   end,

   -- your other functions
}
```

Your application name will be a new table containing all functions that you want to make.

Functions can have one or more parameters. Lua SPOT will only pass one parameter, therefore if you expect more parameters, you have to split the parameters manually. There is a function provided by the Lua SPOT (`luaspot.next_token`) that will help you split the parameters.

Lua SPOT will only send the parameter using string format. You have to manually convert the string to number or other format you expect if you don't want to use string. For example, you can use `tonumber()` function to convert a string to a number.

4.3 Compiling The Script

After writing the script, you have to compile the script into the binary format that later can be installed to the Lua SPOT.

Use the following command to compile the script.:

```
$ luac -o app.lbc app.lua
```

Substitue `app.lua` with your script name and `app.lbc` will be the binary format of your script.

4.4 Installing The Script

The binary version of the script can be installed to the Sun SPOT (with Lua SPOT installed) using the help of LuaDeskSpot application. Follow the *instruction* that has been explained earlier.

4.5 Lua SPOT APIs

In order to use facilities that are provided by Sun SPOT and Lua SPOT, there are several functions that can be used by the Lua script. These functions will be divided into two components, Sun SPOT and Lua SPOT API.

4.5.1 Sun SPOT API

This API provides functions to access functions provided by Sun SPOT.

1. `sunspot.led_on(pos)`
   Turns on the LED at position `pos`.
2. `sunspot.led_off(pos)`
   Turns off the LED at position `pos`.
3. `sunspot.led_rgb(pos, r, g, b)`
   Sets the color of LED at position `pos` with RGB value of `(r, g, b)`.
4. `sunspot.temp_celcius()`
   Returns temperature value in Celcius.
5. `sunspot.temp_fahrenheit()`
   Returns temperature value in Fahrenheit.
6. `sunspot.accel_x()`
   Returns the accelerometer value at x axis.
7. `sunspot.accel_y()`
   Returns the accelerometer value at y axis.
8. `sunspot.accel_z()`
   Returns the accelerometer value at z axis.
9. `sunspot.sleep(delay)`
   Pauses the execution for `delay` miliseconds.

4.5.2 Lua SPOT API

This API provides functions to access functions provided by Lua SPOT.

1. `luaspot.send(addr, msg)`

   Sends a message `msg` to destination address `addr` using the routing function.

2. `luaspot.send_raw(addr, msg)`

   Sends a message `msg` to destination address `addr` directly without using the routing function.

3. `luaspot.add_id(id)`

   This function is used to insert a new message id provided in `id` to the message id table. This table is intended to store all received message id so messages with duplicate id can be dropped.

4. `luaspot.check_id(id)`

   Checks whether a message id is already marked as received or not.

5. `luaspot.get_new_id()`

   Returns a new globally unique message id.

6. `luaspot.get_node_addr()`

   Returns the node address.

7. `luaspot.mem_set(app, var, value)`

   Stores a value to the persistent storage. The value `value` is identified by application name `app` and a variable name `var`.

8. `luaspot.mem_get(app, var)`

   Returns a value from the persistent storage. The value is located using the application name `app` and a variable name `var`.

9. `luaspot.dispatch(addr, msg)`

   Dispatches a new message `msg` from source address `addr` to the main Lua SPOT message dispatcher.

10. `luaspot.lock(app)`

    Creates a global lock identified by `app`. Other threads that are trying to acquire lock with the same identifier will get blocked until the corresponding `luaspot.unlock` function called.

11. `luaspot.unlock(app)`

    Releases the lock identified by `app`.

12. `luaspot.next_token(str)`

    Gets the next token inside the string `str`. This function returns two values, the next token and the remaining string.