openLuup

@akbooer, November 2015

Contents

Overview

openLuup is an environment which supports the running of some MiOS (Vera) plugins on generic Unix systems (or, indeed, Windows systems.) Processors such as Raspberry Pi and BeagleBone Black are ideal for running this environment, although it can also run on Apple Mac, Microsoft Windows PCs, anything, in fact, which can run Lua code (most things can - even an Arduino Yún board.) The intention is to offload processing (cpu and memory use) from a running Vera to a remote machine to increase system reliability.

Running on non-specific hardware means that there is no native support for Z-wave, although plugins to handle Z-wave USB sticks may support this. The full range of MySensors (http://www.mysensors.org/) Arduino devices are supported though the Ethernet Bridge plugin available on that site. A plugin to provide a bi-directional 'bridge' (monitoring / control, including scenes) to remote MiOS (Vera) systems is provided in the openLuup installation.

openLuup is extremely fast to start (a few seconds before it starts running any created devices startup code) has very low cpu load, and has a very compact memory footprint. Whereas each plugin on a Vera system might take ~4 Mbytes, it's far less than this under openLuup, in fact, the whole system can fit into that sort of space. Since the hardware on which it runs is anyway likely to have much more physical memory than current Vera systems, memory is not really an issue.

There is no built-in user interface, but we have, courtesy of @amg0, the most excellent altUI: Alternate UI to UI7 (see the Vera forum board http://forum.micasaverde.com/index.php/board,78.0.html) An automated way of installing and updating the ALTUI environment is now built-in to openLuup. There's actually no requirement for any user interface if all that's needed is an environment to run plugins.

Devices, scenes, rooms and attributes are persisted across restarts. The startup initialisation process supports both the option of starting with a 'factory-reset' system, or any saved image, or continuing seamlessly with the previously saved environment. A separate utility is provided to transfer a complete set of uncompressed device files and icons from any Vera on your network to the openLuup target machine.

For compatibility with Vera systems, openLuup has been written in Lua version 5.1 and requires the appropriate installation to be loaded on your target machine. Installation is not trivial, but it's not hard either. **I would urge you to read this document carefully and then follow the installation steps, and, particularly, the section on "Suggested commissioning process steps"** which move from simple tests to more complex ones and will aid any problem diagnosis if and when it comes to that.

Release 6 (2015-11-17)

Release 6 is a comprehensive (but not totally complete) implementation of Luup, and includes a set of features which are generally sufficient to run a number of standard plugins. The latest additions extend the number of compatible plugins.

Changes from the previous release include:

- **improved integration with the ALTUI interface**
- **uses the LuaFileSystem (lfs) module to enhance portability (works better on PC)**
- **device attributes better preserved over reloads (specifically device .json files)**
- **lul_device global in plugins (see Appendix: Undocumented features of Luup)**
- **nil device and variable parameters supported in luup.variable_watch (ditto)**
- **nil device parameter supported in luup.variable_set/get (ditto)**
- **VeraBridge plugin now clones remote scenes**
- **considerable internal refactoring to support future enhancements**

The details are in the Configuration section below (which is perhaps worth a careful read.)

What openLuup does:

- runs on any Unix machine - I often run it within my development environment on a Mac
- also runs under Windows. However, plugins which spawn commands using the Lua os.execute() function may not operate correctly
- runs the ALTUI plugin to give a great UI experience
- runs the MySensors Arduino plugin (ethernet connection to gateway only) which is really the main goal - to have a Vera-like machine built entirely from third-party bits (open source)
- includes a bridge app to link to remote Veras (which can be running UI5 or UI7 and require no additional software.)
- runs many plugins unmodified – particularly those which just create virtual devices (eg. Netatmo, ...)
- uses a tiny amount of memory and boots up very quickly (a few seconds)
- supports scenes with timers and ALTUI-style triggers
- has its own port 3480 HTTP server supporting multiple asynchronous client requests
- has a fairly complete implementation of the Luup API and the HTTP requests
- has a simple to understand log structure - written to LuaUPnP.log in the current directory - most events generate just one entry each in the log.
- writes variables to a separate log file for ALTUI to display variable and scene changes.

What it doesn't do:

- Some less-used HTML requests are not yet implemented, eg. lu_invoke.
- Doesn't support the <incoming> or <timeout> action tags in service files, but does support the device-level <incoming> tag.
- Doesn’t directly support local serial I/O hardware (there are work-arounds.)
- Won't run encrypted, or licensed, plugins.
- Doesn't use lots of memory.
- Doesn’t use lots of cpu.
- Doesn’t constantly reload (like Vera often does, for no apparent reason.)
- Doesn't do UPnP (and never will.)

Installation

There are seven basic installation steps to set up an openLuup system running the ALTUI interface running the Arduino plugin, and bridging to a remote Vera:

1. install Lua on your target machine
2. create a directory tree on the target machine
3. install ALTUI using files using a simple HTTP request
4. install the Arduino plugin (if you want it)
5. install the VeraBridge plugin (if you want it)
6. install device files and icons from your Vera
7. add the code for the script /usr/bin/GetNetworkState.sh

Once set up, initial overall system configuration is done with a startup file which, like almost everything else in openLuup, is written in Lua (the exceptions being the standard device .xml and .json files.)

---

1. Lua installation

For compatibility with Vera systems, openLuup has been written in Lua version 5.1 and requires the appropriate installation to be loaded on your target machine. You will also need the LuaSocket library, LuaFileSystem, and for some plugins with secure (SSL) network connections the LuaSec library.

It's target machine dependent, but couple of the simpler cases are:

DEBIAN (BEAGLEBONE BLACK)

```
# apt-get install lua5.1
# apt-get install lua-socket
# apt-get install lua-filesystem
# apt-get install lua-sec
```

OPEN-WRT (ARDUINO YUN)

(Lua is pre-installed)

```
# opkg update
# opkg install luasocket
# opkg install luafilesystem
# opkg install luasec
```

2. Host directory tree

The minimal requirement is a single directory which should contain the basic openLuup installation (itself containing one sub-directory called 'openLuup') which can be anywhere. However, it's recommended to construct a more complete emulation of the Vera environment. To do this you need to create:

A. **/etc/cmh-ludl/** this is the directory from which openLuup will be run (it doesn't actually have to be called this at all, but this is the conventional Vera name for this place) into which the D_xxx.xml, D_xxx.json, I_xxx.xml, J_xxx.js, L_xxx.ljua, S_xxx.xml (etc.) files should go **These files should NOT be compressed** (.lzo), as they are in Vera – again, memory is not an issue in most environments. See step 6 below for an easy way to get these files.
B. **/www/cmh/skins/default/icons/** into which any .png image files (device icons) you need should go.
C. **/www/cmh/skins/default/img/devices/** which should contain a symbolic link to the above icons/ directory. This is to allow plugins written for either UI5 or UI7 to run correctly. You do this with the command line instruction

```
> cd /www/cmh/skins/default/img/devices/
> ln -s /www/cmh/skins/default/icons/device_states
```

D. **/var/log/cmh/** this is where the LuUPnP.log file gets written for ALTUI.

You also need to ensure that the port:80 HTTP server on your system (I assume you have one running, perhaps Apache?) sees the /www/cmh/ directory you created above from its server host directory. On my BeagleBone Black, with its native server (cloud9, not Apache) this is done with

```
> ln -s /www/cmh /var/lib/cloud9/
```

Your system may well be different.

3. ALTUI installation

ALTUI installation is now greatly simplified - it can be done with a single HTTP request to a running factory-reset openLuup system. See the suggested commissioning steps below for details.

ALTUI can also auto-update to the latest version. When doing so it creates a directory tree:

```
/etc/cmh-ludl/
          plugins/downloads/altui/   - containing the latest download files
          plugins/backup/altui/ - containing the previous running file versions
```

4. Arduino Gateway plugin

The Arduino Gateway plugin is available from the MySensors site: http://www.mysensors.org/controller/vera

Simply place all the files into the **/etc/cmh-ludl/** directory (or wherever you put the other plugins.)

At this time, **only the Ethernet version** of the plugin is supported (ie. hardware serial I/O is not yet supported in openLuup)

---

5. VeraBridge plugin

The VeraBridge plugin is an openLuup plugin which links to remote Veras. Unlike the built-in Vera 'bridge' capability (which, I think, uses UPnP discovery) this has no limitation as to the Vera firmware version that it links to and you can quite happily run multiple copies of this linking to UI5 and UI7 remote machines (which, themselves, require no special software installation.)

The VeraBridge installation (in the openLuup - VeraBridge directory) comprises:

```
D_VeraBridge.json
D_VeraBridge.xml
I_VeraBridge.xml
L_VeraBridge.lua
```

These files should also go into **/etc/cmh-ludl/**

and the file `VeraBridge.png`

which should go into **/www/cmh/skins/default/img/devices/**

VeraBridge provides local clones of devices and scenes from a remote Vera, which:

- reflect the status of the remote devices (ie. device variables)
- present, through ALTUI, a display panel and controls
- allow control of the remote devices (eg. on/off, dimming, ...)
  through the usual luup.call_action mechanism (or the control panel)
- makes remote scenes visible locally, allowing timers and triggers to be added

The device numbering is different from that on the remote machine, being incremented by multiples of 10000 for each different bridge, and one of VeraBridge's main functions is to intercept actions on the local devices and pass them to the remote ones. What does NOT work is to set a variable on a local device and expect that variable on the corresponding remote device to change.

Since it is possible (but fairly unlikely) that the VeraBridge will miss an update to a Vera device variable, every so often (actually, about every minute) it scans the whole lot and updates all the cloned devices on openLuup. This means that WHATEVER changes you make to cloned device variables will get overwritten frequently. The motto is this...
do NOT write variables to cloned devices.

Running under openLuup, VeraBridge supports *any* serviceId/action command request (ie. those *not* returning device status parameters) This covers most device control scenarios.

6. Installing other Device files

The entire set of device files (*.xml, *.js, *.json, *.lua) can be copied from a Vera on your local network using the openLuup_getfiles utility found in the Utilities folder of the distribution.

Simply run the Lua file from the command line when in the /etc/cmh-ludl/ directory

```
> lua openLuup_getfiles.lua

openLuup_getfiles - utility to get device and icon files from remote Vera
Remote Vera IP: 172.16.42.14     <—— input your remote Vera IP here
221       ALTUI_LuaRunHandler.lua
4728      D_ALTUI.json
1086      D_ALTUI.xml
4814      D_ALTUI_UI7.json
10564     D_Arduino1.json
1034      D_Arduino1.xml
…
```

It lists the file size and names as it uncompresses and transfers files. Device files are put into a ‘files’ sub-directory, and icons into ‘icons’.

From there, they should be moved to their respective directories as described in the section 2 above.

7. Adding code for /usr/bin/GetNetworkState.sh and /etc/cmh/ui

Some plugins (eg. Sonos) use a shell command `/etc/bin/GetNetworkState.sh` to determine the machine's IP address. I used to provide this (actually for ALTUI, but it doesn't need it now.) So you need to create that file with the contents shown in the Appendix.

On a Windows machine you may need to use the simpler hard-coded address version of the script shown there. I’m not actually sure that this works.

At least one plugin (IOSPush) requires the file /etc/cmh/ui with the single line (the digit seven):

```
7
```

Configuration

Configuration changes to the system happen in different ways in the three major phases of normal running, startup and shutdown:

STARTUP

- the default behaviour is to look for a JSON file called `user_data.json` in the current directory (`/etc/cmh-ludl/`) loading the device/room/scene configuration from that.
- an optional startup parameter is one of:
  - the word `reset`, which forces a 'factory reset', or
  - a user_data JSON formatted filename, or
  - a Lua filename to be run in the context of a factory-reset system (with no rooms or scenes, and only devices 1 (Gateway) and 2 (Z-wave controller) defined)

RUNNING

- every 6 minutes the system configuration is check-pointed to the file `user_data.json`, regardless of the name or type of the startup file. This will capture device variable and attribute changes, scene creation/deletions, etc.
- logged events are written to the log files as they happen and not cached.

SHUTDOWN

- on luup.reload, or full exit, the configuration will be written to the file `user_data.json` in the current directory
- on luup.reload, or any other configuration change requiring a reload (eg. new child devices created) the process will exit with status 42
- on exit (from the HTTP request id=exit) will exit with status of 0 (successful exit)

This means that it is easy to save and restore arbitrary configurations, or start from scratch. Since any reload will cause the process to exit, it's necessary to launch openLuup from within a script, in order to restart automatically. A Unix shell script to do this, `openLuup_reload`, is included in the distribution (in the openLuup - Utilities directory) and shown here. Under Windows, you'd no doubt need a slightly different syntax.

```
#!/bin/sh
#
# reload loop for openLuup
# @akbooer, Aug 2015
# you may need to change 'lua' to 'lua5.1' depending on your install

lua openLuup/init.lua $1

while [ $? -eq 42 ]
do
   lua openLuup/init.lua
done
```

An example Lua startup code, shipped with the openLuup installation and listed in an Appendix, shows how to make a minimal system running ALTUI, Arduino Gateway, and a VeraBridge to a remote Vera. The **bold lines** show where you might change the customisation.

Suggested commissioning process steps

On the basis of walking before you can run, the following sequence should help getting openLuup up and running and simplify diagnostics:

INSTALLATION CHECK

From the **/etc/cmh-ludl** directory run
```
> lua5.1 openLuup_check.lua
```

This may produce diagnostic warnings, but should not generate an error. If it does, some installation files are missing.

OPENLUUP - FACTORY RESET

From the **/etc/cmh-ludl** directory run
```
> lua5.1 openLuup/init.lua reset
```

After a second or so, the process should terminate with a reload message.
Has this created the files: `LuaUPnP.log` and `user_data.json` ?
Compare these with the examples shown in the Appendices.

STARTUP THE RESET SYSTEM

From the **/etc/cmh-ludl** directory run
```
> lua5.1 openLuup/init.lua
```

This should not terminate, and should allow you to proceed with the following steps.

PORT:80 SERVER

From your browser, check you can access an image file in the icons directory:
`http://<YourIP>/cmh/skins/default/icons/VeraBridge.png`

and in the other alias

`http://<YourIP>/cmh/skins/default/img/devices/device_states/VeraBridge.png`

PORT:3480 SERVER

From your browser:
`http://<YourIP>:3480/data_request?id=alive`

…should return the message “OK”.

Also from the browser:

`http://<YourIP>:3480/data_request?id=user_data`

…should return a JSON object (which your browser may render in a special view or a plain text) showing an expanded 'user_data' version of the user_data.json file you already saw.

The following URL should halt the openLuup engine and exit cleanly:

`http://<YourIP>:3480/data_request?id=exit`

ALTUI USER INTERFACE

Restart the openLuup engine, doing a factory reset:
`> ./openLuup_reload reset`

All the ALTUI files can be installed into the `/etc/cmh-ludl/` directory automatically using the URL:

`http://<YourIP>:3480/data_request?id=altui`

(this takes about one minute and requires, of course, an internet connection.) If you want to watch it working you can tail the log with `tail -f LuaUPnP.log`

This should load a functional ALTUI interface which should run when you access the URL

`http://<YourIP>:3480/data_request?id=lr_ALTUI_Handler&command=home#`

You will immediately be prompted to upgrade to the latest version (you don't have to – again, it takes about a minute and will restart openLuup.) You should be able to exercise all the ALTUI functionality through menus, etc., and simply see a single device (ALTUI) in the system.

Exit cleanly:

`http://<YourIP>:3480/data_request?id=exit`

STARTUP WITH LUA FILE

Make a copy of the `startup_example.lua` file and name it `startup.lua` customising it to your needs. Since you'll be starting from a clean system you do need to explicitly create the ALTUI device. All the files, though, should be in place following the previous update step.

Restart the openLuup engine, this time using the shell script to implement restarts:
`> ./openLuup_reload startup.lua`

From your browser:
`http://<YourIP>:3480/data_request?id=user_data`

should now return a more extensive user data dump showing the additional devices configured in the startup file, along with any top-level attributes and rooms you may have defined.

Note that subsequent reloads of the system using the script file openLuup_reload (with no parameter) will NOT use this startup code, but whatever Lua code you have set in the "StartupCode" top-level attribute.

ARDUINO GATEWAY

This should be visible as an installed device on the ALTUI Devices pages. Start/stop inclusion and other actions are available from the device control panel.

VERA BRIDGE

If you have installed the VeraBridge plugin files and configured the correct remote Vera IP address in the `luup.ip_set` command in the startup file, then you should see in the interface all your remote devices bridge to the openLuup system.

Sensors should reflect their correct readings, actions such as set dimmer level and light on/off should work as expected.

You should be up and running with your very own openLuup system. Other devices can be installed and configured following the example in the `startup_examples file`.

BACKUP

Whatever else you do, backup your functional user_data.json file.

You can always revert to this if you get stuck, using it as the input parameter to openLuup_reload. So long as it is call something different, it will NOT be over-written, so this is a very easy way to switch between different configurations.

One other 'safety net' that openLuup gives you in terms of restoring a configuration is to use a startup.lua file (again, you can have many of these called different things.) The Lua syntax of the file gives a very simple and readable description of the configuration and, as the example shows, it's very easy to create devices, rooms, etc. Scenes are more difficult, however. In fact, at the moment, you can't – you'd have to recreate them manually through the ALTUI interface.

If you DO recreate a system from a startup.lua file, then be aware that every plugin that creates new child devices will, in turn, request a reload. Depending on what the plugin does in its own startup code (access REST APIs over the network, local files, delayed startups…) this can take quite a long time and you should see the system restart messages. During this time the ALTUI interface (if you have configured it) will not be accessible.

Even after a system reboot, the only thing required to get openLuup restored to its previous state is to start the openLuup_reload script (with no parameters) since this just picks up the latest user_data.json file and runs with it.

Appendix: user_data.json file after factory reset


```
{
  "BuildVersion":"*1.7.0*",
  "City_description":"Greenwich",
  "Country_description":"UNITED KINGDOM",
  "Device_Num_Next":"3",
  "InstalledPlugins2":[],
  "KwhPrice":"0.15",
  "LoadTime":"1444832719",
  "Mode":"1",
  "ModeSetting":"1:DC*;2:DC*;3:DC*;4:DC*",
  "PK_AccessPoint":"88800000",
  "Region_description":"England",
  "StartupCode":"",
  "TemperatureFormat":"C",
  "currency":"£",
  "date_format":"dd\/mm\/yy",
  "devices":[{
    "ControlURLs":{"service_1":{
        "ControlURL":"upnp\/control\/dev_1",
        "EventURL":"\/upnp\/event\/dev_1",
        "service":"urn:micasaverde-com:serviceId:ZWaveNetwork1",
        "serviceType":"urn:schemas-micasaverde-org:service:ZWaveNetwork:1"}},
    "altid":"",
    "category_num":19,
    "device_file":"D_ZWaveNetwork.xml",
    "device_type":"urn:schemas-micasaverde-com:device:ZWaveNetwork:1",
    "id":1,
    "id_parent":0,
    "impl_file":"I_ZWave.xml",
    "invisible":"1",
    "ip":"",
    "mac":"",
    "manufacturer":"",
    "model":"",
    "name":"ZWave",
    "room":"0",
    "states":[],
    "subcategory_num":0,
    "time_created":"1444832719"},{
    "ControlURLs":{
      "service_2":{
        "ControlURL":"upnp\/control\/dev_2",
        "EventURL":"\/upnp\/event\/dev_2",
        "service":"urn:micasaverde-com:serviceId:SceneController1",
        "serviceType":"urn:schemas-micasaverde-com:service:SceneController:1"},
      "service_3":{
        "ControlURL":"upnp\/control\/dev_3",
        "EventURL":"\/upnp\/event\/dev_3",
        "service":"urn:micasaverde-com:serviceId:HaDevice1",
        "serviceType":"urn:schemas-micasaverde-com:service:HaDevice:1"}},
    "altid":"",
    "category_num":14,
    "device_file":"D_SceneController1.xml",
    "device_json":"D_SceneController1.json",
    "device_type":"urn:schemas-micasaverde-com:device:SceneController:1",
    "id":2,
    "id_parent":1,
```

```
    "invisible":"1",
    "ip":"",
    "mac":"",
    "manufacturer":"",
    "model":"",
    "name":"_SceneController",
    "room":"0",
    "states":[],
    "subcategory_num":0,
    "time_created":"1444832719"}],
  "gmt_offset":"0",
  "latitude":"51.48",
  "longitude":"0.0",
  "mode_change_delay":"30",
  "model":"Not a Vera",
  "rooms":[],
  "scenes":[],
  "timeFormat":"24hr",
  "timezone":"0"}
```

Appendix: LuaUPnP.log after factory reset

```
2015-10-15 11:18:37.531   :: openLuup STARTUP ::
2015-10-15 11:18:37.534   openLuup.init::      version 2015.10.15  @akbooer
2015-10-15 11:18:37.725   openLuup.scheduler:: version 2015.10.15  @akbooer
2015-10-15 11:18:37.728   openLuup.server::    version 2015.10.15  @akbooer
2015-10-15 11:18:37.882   openLuup.plugins::   version 2015.10.15  @akbooer
2015-10-15 11:18:37.980   openLuup.scenes::    version 2015.10.15  @akbooer
2015-10-15 11:18:38.130   openLuup.plugins::   version 2015.10.15  @akbooer
2015-10-15 11:18:38.159   openLuup.chdev::     version 2015.10.15  @akbooer
2015-10-15 11:18:38.206   openLuup.io::        version 2015.10.15  @akbooer
2015-10-15 11:18:38.209   openLuup.luup::      version 2015.10.15  @akbooer
2015-10-15 11:18:38.390   openLuup.rooms::     version 2015.10.15  @akbooer
2015-10-15 11:18:38.393   openLuup.requests::  version 2015.10.15  @akbooer
2015-10-15 11:18:38.473   luup.create_device:: [1] urn:schemas-micasaverde-
com:device:ZWaveNetwork:1 / no-implementation-file
2015-10-15 11:18:38.582   luup.create_device:: [2] urn:schemas-micasaverde-
com:device:SceneController:1 / no-implementation-file
2015-10-15 11:18:38.588   openLuup.init:: loading configuration reset
2015-10-15 11:18:38.594   openLuup.luup:: device 0 '_system_' requesting reload
2015-10-15 11:18:38.596   luup.reload:: saving user_data
```

Appendix: Example openLuup startup.lua file

```
-- Example startup.lua

do -- define top-level attributes required to personalise the installation
  local attr = luup.attr_set

  attr ("City_description", "Oxford")
  attr ("Country_description", "UNITED KINGDOM")
  attr ("KwhPrice", "0.15")
  attr ("PK_AccessPoint", "88800127")   -- TODO: use machine serial number?
  attr ("Region_description", "England")
  attr ("TemperatureFormat", "C")

  attr ("currency", "£")
  attr ("date_format", "dd/mm/yy")
  attr ("latitude", "50")
  attr ("longitude", "-1")
  attr ("model", "BeagleBone Black")
  attr ("timeFormat", "24hr")
  attr ("timezone", "0")
end

do -- create rooms (strangely, there's no Luup command to do this directly)
 local function room(n)
   luup.inet.wget ("127.0.0.1:3480/data_request?id=room&action=create&name="..n)
 end
  room "Upstairs"             -- these are persistent across restarts
  room "Downstairs"
end

do -- ALTUI
  local dev = luup.create_device (‘’, “ALTUI”, ”ALTUI”, "D_ALTUI.xml")
end

do -- ARDUINO
--  local dev = luup.create_device ('', "Arduino", "Arduino", "D_Arduino1.xml")
--  luup.ip_set ("172.16.42.21", dev)       -- your Arduino gateway IP address
end

do -- the VERA BRIDGE !!
--  local dev = luup.create_device (‘’, "Vera", "Vera", "D_VeraBridge.xml")
--  luup.ip_set ("172.16.42.10", dev)         -- set remote Vera IP address
end

-- set up whatever Startup Lua code you normally need here
luup.attr_set ("StartupCode", [[

luup.log "Hello from my very own startup code"

]])

-----------
```

Appendix: code for /usr/bin/GetNetworkState.sh

Some plugins (eg. Sonos, DLNA, Squeezebox, …) require an external shell script (part of a standard Vera installation) to define the host machine IP address.

There's two basic ways to implement this.

This file can either be VERY simple:

```
echo -n 172.16.42.88
```

...with the IP address appropriately set (manually)

…or a more sophisticated approach using a Lua script to return the result (automatic) which is described here:

http://forums.coronalabs.com/topic/21105-found-undocumented-way-to-get-your-devices-ip-address-from-lua-socket/

```
#! /usr/bin/env lua
------------------------------------------------------------------------
-- discover main IP address of machine and write to standard output
-- http://forums.coronalabs.com/topic/21105-found-undocumented-way-to-get-your-
devices-ip-address-from-lua-socket/
------------------------------------------------------------------------
local socket = require "socket"
function myIP ()
  local mySocket = socket.udp ()
  mySocket:setpeername ("42.42.42.42", "424242")  -- arbitrary IP/PORT
  local ip = mySocket:getsockname ()
  mySocket: close()
  return ip or "127.0.0.1"
end
io.write (myIP())
--------
```

Appendix: Undocumented features of Luup

The documentation for Luup is poor: out of date, misleading, incomplete, unclear, and sometimes just plain wrong. During the implementation of openLuup a number of undocumented 'features' have come to light. In some cases, these features are used by various plugins, either deliberately or unknowingly, through sins of commission or omission. As a result, I've had to include them in the openLuup implementation.

If you're a developer, please try NOT to use these things in your plugin code.

Here's what I've found - if you know more, let me know.

- **lul_device** – this variable is often included in callback function parameter lists to indicate the target device, and that's fine. However, it ALSO turns out to be in scope in the whole body of a plugin's Lua code. Some plugins rely (possibly inadvertently) on this feature.
- **nil device parameter in luup.variable_watch** – whilst the use of a nil variable parameter to watch ALL service variables is documented, the use of a nil DEVICE is not, but works as expected: the callback occurs a change in ANY device with an update to the given serviceId and variable (thanks for @vosmont for that information.)
- **nil device parameter in luup.variable_set/get** – it appears that when called from device code, a missing device parameter gets the current luup.device variable value substituted.

Appendix: Unimplemented features of openLuup

openLuup is, as of this time, an unfinished work. The following features are known to be unimplemented, poorly implemented, or non-functional at this time, for a variety of reasons.

---

unimplemented luup API calls

- **luup.devices_by_service** – the definition of the functionality here Luup_Lua_Extensions is not adequate to make an implementation.
- **luup.job.*** – all functions in this module (status, set, setting) are empty stubs.
- **luup.require** – undocumented.
- **luup.xj** – undocumented.

---

unimplemented HTTP requests

- **id=invoke** – not implemented.
- **id=scene** – only create, delete, list, and rename are implemented (ie. no interactive creation of scenes.) Also note that scene triggers (or notifications) are not stored.
- **id=action** – the virtual category 999 is not implemented. Actions on groups of devices defined by category is not implemented.
- **id=finddevice** – not implemented.
- **id=resync** – not implemented.
- **id=archive_video** – not implemented.
- **id=jobstatus** – not implemented.
- **id=invoke** – not implemented.
- **id=relay** – not implemented.