Some simple explanations for beginners

Carsten Lynker

December 2013

First steps programming FlyWithLua

This is more a *cooking book* than a real in-deep tutorial. It will give some fresh ideas to beginners, helping to find the golden thread to run successful through a Lua scripting adventure. So don't forget to study the manual, especially as a reference to all the functions used in this book.

When are my scripts running?

All scripts written in Lua (the file name ends at ".lua" and the file is inside the "FlyWithLua/Scripts" directory) are executed once if you start X-Plane, if you change your plane or location - or if you force FlyWithLua to reload all scripts. This can be done by clicking the menu "Plugins" -> "FlyWithLua" -> "Reload all Lua script files".

A first Lua file

Start X-Plane and start a text editor (like Notepad++ or VIM). Then write the file "hello world.lua" into the "Scripts" directory. Fill the file with these Lua code:

```lua
logMsg("Hello World!")
```

Save the file with your editor and switch to X-Plane. Click on "Special" -> "Show Dev Console". You see some logging info from X-Plane. Now force FlyWithLua to reload your script and watch the developer console. You will see this (and other code from FlyWithLua):

```
FlyWithLua Info: Start loading script file Resources/plugins/FlyWithLua/Scripts/hello world.
Hello World!
FlyWithLua Info: Finished loading script file Resources/plugins/FlyWithLua/Scripts/hello wor
```

You can see, your little script does its work once during its run.

Setup start parameters

You can use this behavior to setup some start conditions. This can be done, as FlyWithLua automatically runs the scripts when you change your plane or location. So if you want to always start with cold&dark setting in X-Plane, but have the orange beacon on by default, write a script like this:

```
command_once("sim/lights/beacon_lights_on")
```

You can still use your beacon light switch, as the script is executed once and the command is executed only once as well. The Lua function "command_once()" will call an X-Plane command. You will see all the commands, if you click on "Settings" -> "Joystick & Equipment" -> "Buttons: Adv".

DataRefs

The most exiting thing programming with FlyWithLua is, that you can read and write DataRefs. These are X-Plane's internal variables, representing much more than the commands can reach. Take a look at this web page:

http://www.xsquawkbox.net/xpsdk/docs/DataRefs.html

So let's try the same procedure as before. Change your little script to this:

```
dataref("Beacons", "sim/cockpit/electrical/beacon_lights_on", "writable")
Beacons = 1
```

And again, all your action in X-Plane starts with beacon lights on (to be correct, it will start the beacon when you switch the battery on). You must be careful when using a DataRef. Not all of them can be set by a plugin. If they are writable or not can be found in the fourth column of the official DataRef listing.

Event driven programming

You can't only do things once when the script loads. There are Lua functions, that allows to react on simulator events. For example when the mouse scroll wheel is moved. Write a new script "no more zooming.lua" and fill it with this code:

```
do_on_mouse_wheel("RESUME_MOUSE_WHEEL = true")
```

Hey, this disables the zoom on the mouse wheel. In fact this little piece of software, a tiny one-liner, does nothing on a mouse wheel event, but resumes the event. So the SDK's event handler stops action and scroll wheel info never gets to X-Plane.

Doing some action on wheel movements

As killing the zoom only is a little bit boring, we will add some action. The mouse wheel movement is represented by the variable MOUSE_WHEEL_CLICKS. We can use this info to move the trim wheel with the mouse wheel. Use this code:

```lua
dataref("xp_elv_trim", "sim/flightmodel/controls/elv_trim", "writable")

function set_trim_by_mouse_wheel()
    xp_elv_trim = xp_elv_trim - MOUSE_WHEEL_CLICKS * 0.0025
    if xp_elv_trim > 1.0 then
        xp_elv_trim = 1.0
    end
    if xp_elv_trim < -1.0 then
        xp_elv_trim = -1.0
    end
    RESUME_MOUSE_WHEEL = true
end

do_on_mouse_wheel("set_trim_by_mouse_wheel()")
```

This is a little bit more complex. We first define a DataRef connection to a variable, that allows writable access. Then we define a function, to not have too many code to write inside the do_on_mouse_wheel() function. This is not really a problem, it's more a sort of ugly code.

The function sets the DataRef with the MOUSE_WHEEL_CLICKS variable as a multiplier. As this can result in forbidden values (range from -1.0 to +1.0), we make sure that correct values are given back to X-Plane with two if statements.

Graphical output

We want to see the movement of the trim when we use the mouse wheel. To show something with Lua, it must be done with the do_every_draw() function. So we add the code above:

```lua
function draw_trim_info()
    XPLMSetGraphicsState(0,0,0,1,1,0,0)
    glColor4f(1, 1, 1, 0.5)
    glRectf(100, 100, 110, 300)
    glBegin_LINES()
    glVertex2f(90, 200 + xp_elv_trim*100)
    glVertex2f(120, 200 + xp_elv_trim*100)
    glEnd()
end

do_every_draw("draw_trim_info()")
```

This will show an info onto the screen. We use OpenGL functions, that are implemented into FlyWithLua's Lua dialect. But one thing is still stupid, the info is shown all the time. We want it to disappear after 5 seconds. Let's modify the code again:

```
dataref("xp_elv_trim", "sim/flightmodel/controls/elv_trim", "writable")

local show_trim_info_until = 0

function set_trim_by_mouse_wheel()
    xp_elv_trim = xp_elv_trim - MOUSE_WHEEL_CLICKS * 0.0025
    if xp_elv_trim > 1.0 then
        xp_elv_trim = 1.0
    end
    if xp_elv_trim < -1.0 then
        xp_elv_trim = -1.0
    end
    RESUME_MOUSE_WHEEL = true
    show_trim_info_until = os.clock() + 5
end

do_on_mouse_wheel("set_trim_by_mouse_wheel()")

function draw_trim_info()
    if os.clock() < show_trim_info_until then
        XPLMSetGraphicsState(0,0,0,1,1,0,0)
        glColor4f(1, 1, 1, 0.5)
        glRectf(100, 100, 110, 300)
        glBegin_LINES()
        glVertex2f(90, 200 + xp_elv_trim*100)
        glVertex2f(120, 200 + xp_elv_trim*100)
        glEnd()
    end
end

do_every_draw("draw_trim_info()")
```

Now we use the Lua function os.clock() to get the time in seconds the simulator is running. We add five to this value, to get the time in five seconds, and store this value in a local variable. The variable must be set (and defined as local) *before* the function to display it is defined and connected to the simulator's drawing loop. Else the first run in the drawing loop crashes the script as the variable is unknown.

As we didn't know if an other script set's the OpenGL graphic state wrong for our output, we set it to the default value with the XPLMSetGraphicsState() function. If not, it can end in some strange behavior, if am other script disables alpha for example.