FSUIPC: Lua Plug-Ins

(For FSUIPC4/ESPIPC version 4.325 and later, or FSUIPC3 version 3.845 and later)

This document is the interim Manual for the LUA plug-in facilities first added to FSUIPC4 at version 4.211, and FSUIPC3 at version 3.841. For now it takes the form of a series of questions and answers:

What is "Lua"?

It is a programming language. The best way to see and learn what it is all about is to visit this web page:

http://www.lua.org/about.html

What is a "plug-in"?

Plug-in is simply a technical term for a program which can be run inside or as part of another program. Effectively FSUIPC is a "plug-in" for FS. The Lua facilities in FSUIPC allow multiple plug-ins by loading and running individual Lua programs.

Why has this been added to FSUIPC?

Because I am often asked by users, especially cockpit builders, how to do quite sophisticated things with FSUIPC's quite basic button programming facilities, and adding Lua plug-in capabilities makes those much more powerful and much easier to deal with and see what is going on.

The use of the compound and conditional button programming facilities, combined with multiple parameter assignments to buttons, is not only awkward, it is really pushing the simple parameter design in the INI files to the limit.

What is provided in FSUIPC for Lua programming?

First, FSUIPC recognises all files placed into the Modules folder that have filetype ".lua". These should all be Lua programs, either in normal interpreted source format or in the "compiled" format if desired (Lua provide a compiler "luac.exe" which just saves a little loading time by pre-processing the source into a binary format easier for the interpreter).

All .lua files are assigned a numeric reference and listed with it in the [LuaFiles] section of the FSUIPC INI file. It is the reference number which is encoded into other references to Lua programs within the INI file – much like the way Macro files are handled.

When there are Lua files in the modules folder, FSUIPC adds a number of new controls for assignment in all of the usual places – Buttons & Switches, Key Presses, and Axis Assignments. The controls added for each Lua program are:

Lua <name>	to run the named program
Lua Debug <name>	to run the program in debug mode (more below)
Lua Kill <name>	to forcibly terminate the named program, if it is running
Lua Set <name>	to set a flag (0-31 according to parameter) specifically for the named program to test
Lua Clear <name>	to clear a flag (0-31 according to parameter) specifically for the named program to test
Lua Toggle <name>	to toggle a flag (0-31 according to parameter) specifically for the named program to test

There's also a general Lua control "Lua Kill All" to forcibly terminate all currently running Lua programs.

FSUIPC currently allows up to 256 simultaneously running Lua programs, each independently running in their own FS thread. When you start a Lua program running which is already running, the previous incarnation is first ruthlessly and unceremoniously terminated. Because of the termination facilities provided it is not a problem having a program which is designed to sit in a loop forever doing things, like monitoring the state of FS values.

There are currently three explicitly reserved Lua names, for programs which are run automatically if present:

Ipcinit.lua	automatically run as soon as FSUIPC has initialised (and for FSX or ESP, connected correctly to SimConnect).

Ipcready.lua	automatically run when FS is really "ready to fly".
ipcDebug.lua	automatically loaded before any Lua program which is started in Debug mode.

What about access to FSUIPC offsets, FS facilities, and files?

The main useful standard libraries provided with Lua version 5.1 are present and already loaded when any FSUIPC Lua plug-in is run. These are:

package	Facilities for loading and building Lua modules
table	Table manipulation, operating on arrays or lists
io	File input and output facilities
os	Operating system functions like date, time, plus more ambitious stuff
string	String manipulation
math	All the maths functions you could possibly desire
debug	Functions to help get more complex Lua programs working

Note that in the early releases I have not specifically removed anything from these libraries. That doesn't mean that all of their facilities will work, nor are safe to use without risk of crashing FS or distorting its operations. But that's one of the risks of power. Looking at the Package and Operating System functions I can see plenty of scope for getting into real trouble! (If folks would please notify me when they find something so dangerous it should be removed, I will gradually make it all "safer", but hopefully still not restrictive).

The only change to the standard libraries so far is to make the ***os.exit*** function merely exit and terminate the Lua thread it is executed in. (It is the same as the added IPC library ***ipc.exit*** function).

In addition to these standard libraries, FSUIPC adds three more:

ipc	Facilities for interfacing to FS and FSUIPC.
logic	Bit-manipulating logic facilities, otherwise missing in Lua
event	Facilities for taking action on events in FS – arising from buttons, keypresses, FS controls and FSUIPC offset changes

These three are documented in a separate document which you should find with this package.

On top of these facilities, when a Lua plug-in is run because of an FS control (Lua <name> or Lua Debug <name>), any parameter passed with that control is available to the Lua program as a variable called **`ipcPARAM`**. This might be particularly useful if the control is assigned to an Axis or POV, where the axis or POV value is thereby passed to the program.

The facilities provided by Lua and these libraries are certainly quite sufficient to actually program working subsystems for your aircraft cockpit. Currently there are no specific hardware interfaces – you'd talk to most current hardware via FSUIPC offsets, or by using USB-type filenames and the "io" file functions. If folks would like direct interfaces to popular hardware interface cards, those used for display driving, and button/switch/dial inputs, I'm sure these can be built in, or added on, perhaps partially as Lua programs themselves, or with some extra libraries specifically oriented. Mostly I cannot do these directly myself, or at least not without the hardware in question, but I'd be glad to discuss ways and means with those who could either do it, or assist appropriately.

What about some examples, please?

Included in the package you have downloaded are four LUA files, ready to be used:

ipcDebug.lua	The auto-loaded program section loaded before any Lua program being debugged. It enables line tracing to the Lua program's own Log file.
log lvars.lua	**[For FSUIPC4 only]** a useful little routine which logs all of the currently available local panel variables (LVARS) which can be read and written using the Lua ipc library, or written using FSUIPC Macros via the "L:<name>,action" facilities. The values are listed in the Log initially and when any change, and also displayed as they change on the screen, in the Lua display window. Use this to work out how to define your macros in order to operate many switches and facilities otherwise inaccessible without using a mouse.

Init pos.lua	A small program which simply places the user’s aircraft at a fixed place with a given airspeed. (The airspeed setting only works correctly with FSX or ESP).
Freeze.lua	Attempts to freeze the aircraft position and attitude by reading these values and then repeatedly copying those details back in a continuous loop. On FSX/ESP, because of the asynchronous nature of the SimConnect interface, this tends to produce a judder with the position and attitude gradually changing, so it isn’t a very good idea (FSX/ESP provide Freeze controls in any case. On FS9 the freeze does tend to work a little better, without judder.
Display vals.lua	Continuous on-screen displays of some aircraft variables. Undock the window fro greater clarity.
Record to csv.lua	A data recorder, writing lines of important data about the aircraft at up to 20 times a second, The file is in CSV format, displayed nicely in Excel and similar programs. [Note that the original version included a syntax error in line 49].
Landing.lua	A (failed) attempt to show off some of the things you can do using the event library. This one *tries* to provide landing assistance automatically, but needs a lot of development and tailoring. Nevertheless, the example does show some interesting ideas for the Event library facilities.

I’d like to add more *practical* examples of cockpit solutions too, when they arise, so please do tell me things you manage to achieve with Lua!

Published by Peter L. Dowson, 4th February 2009