FSUIPC7: Lua Plug-Ins

This document is part of the Manual for the LUA plug-in facilities. This part takes the form of a series of questions and answers. The library reference document, "**FSUIPC Lua Library**" provides the technical data for the FSUIPC Lua additions.

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 its installation 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). Note that the filename part (preceding the .lua filetype) can contain spaces and other characters acceptable to Windows, ***but there is a limit of 16 characters on that name.***

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 installation 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
LuaDebug <name>	to run the program in debug mode (more below) Ths is actually superseded by the Trace Lua option in FSUIPC4's Logging Tab.
LuaKill <name>	to forcibly terminate the named program, if it is running
LuaSet <name>	to set a flag (0-255 according to parameter) specifically for the named program to test
LuaClear <name>	to clear a flag (0-255 according to parameter) specifically for the named program to test
LuaToggle <name>	to toggle a flag (0-255 according to parameter) specifically for the named program to test
LuaValue <name>	to set the ipcPARAM variable to the given parameter, or the axis value when so assigned

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

NOTE that the <name> in these assignments can only be up to 16 characters in length, so this restricts the names of the Lua files provided for FSUIPC use in the FSUIPC installation folder. The 16 does not include the ".lua" filetype, but does include any spaces in the filename.

FSUIPC 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. However, generally it is better to write those as event-driven programs—the **event** library provides the methods for this. Note that, in order to prevent the over-rapid killing and resurrecting of the same plug-in over and over, there is a limit imposed on the time between each resurrection. This defaults to 66 milliseconds (i.e. up to 15 cycles/sec), but can be adjusted in the FSUIPC INI file by the parameter LuaRerunDelay. This really only affects assignments to rotary encoders and axes, because button and key repetition is naturally limited in any case.

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 connected to the FS.
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.

Can I run Lua plug-ins on a WideFS client PC?

Yes. Such facilities have now been added. Since version 6.81, WideClient automatically runs any Lua programs it finds in its own folder (i.e. the folder containing the WideClient.exe being used). These are started only after WideClient is fully connected to FS on the server PC.

Additionally, WideClient will look for a Lua program called "Initial.lua" when it first starts up. This will run immediately, before any access to FS and FSUIPC is obtained, so it should not be used for anything dependent upon valid offset values. It can run other Lua programs, using the "ipc.runlua" function, but they will need to be located elsewhere—not in the WideClient folder—or else they will be restarted when FS is connected. The INITIAL.LUA program is not re-run later and is the only exception.

Note that the library support for Lua on WideClient is a subset of that provided in FSUIPC. The library document highlights those functions omitted in WideClient.

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, so far at least, 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).

Additionally, the third-party file system library, **lfs** (LuaFileSystem) by the Kepler Project is also built-in. A separate document is supplied describing this library.

For 64-bit lua socket support, please see this User Contribution (it references FSUIPC5 but is still valid for FSUIPC7)..

The only changes to the standard libraries so far are as follows:

(a) Made 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).

(b) Made the ***print*** function act identically to the added IPC library ***ipc.log*** function.

(c) Made the ***io*** library function send the data to the log when the ***stdout*** or ***stderr*** devices are specified.

(d) Changed the searching for modules, carried out by ***require***, to look in the FSUIPC installation folder for Lua modules, and a Lua subfolder (...\lua) for Lua modules and code DLLs. I recommend that all Lua add-ons which are *not* run directly by FSUIPC, be placed in the Lua subfolder, or subfolders off that. This applies whether they are Lua modules or DLLs. DLLs should *not* be placed in the FSUIPC installation folder directly.

In addition to the standard libraries, FSUIPC adds eight 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, mouse events, FS controls, FS events, FSUIPC offset changes, flag and parameter changes, and on a timer.
gfd	Go-Flight Device facilities: for reading GF switches, dials, levers, and setting GF displays and indicators, using the GoFlight module "GFDev.dll".
sound	Facilities for playing and looping on sound files on any available sound devices.
com	Facilities for handling serial port (COM) and USB HID devices, with special facilities for extracting axis and button data from joystick types.
mouse	Facilities for moving the mouse pointer and manipulating mouse buttons and wheels.

ext	Facilities for running, stopping, and manipulating external programs and their windows. Ths can also do some manipulation (sizing and positioning) of FS's own windows, when undocked.
wnd	Facilities for creating text display windows.

and one extra in WideClient only:

display	Facilities for creating a simple dialogue window with up to 16 read-only edit fields which can easily be populated by FS or other data.

These are documented in a separate document which you should find with this package. There's a specific example Lua plug-in for the WideClient display library, too, called just "mydisplay.lua".

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 added FSUIPC control "**LuaValue** ..." also passes values via **`ipcPARAM`**, and changes to it can trigger Lua plug-in code via a special **event**.

Note that the **`ipcPARAM`** value is set enternally to the Lua plug-in before each function entry from an event, and also after every use of the **ipc.sleep** function. It is not a variable which can be used internally to the plug-in to retain its own value unless both events and sleeps are completely avoided.

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 many LUA files, ready to be used. Here are details of some. Please also visit the **User Contributions** sub-forum on my Support Forum for lots more and additional help.

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.
HidDemo.lua	This shows what can be done with the **com** library facilities to read and process HID (Human Interface Devices) with particular attention to joystick or analogue axis values and buttons. If you have a device which exceeds FSUIPC's capacities for axis types and numbers or buttons, this is the way to solve that problem. You can handle up to 16 each of 12 difference named axes, and up to 256 buttons, on each connected device of a joystick type. And because WideClient also supports the **com** library, you can do this across a Network too.
rotaries.lua	This uses the HID features of the **com** library to implement fast and slow turning button results for rotary encoders which otherwise only indicate one button press/release for each direction.
TripleUse.lua	This is an example of using the event.button() function for getting three separate uses from a single button, by single click, double click and longer press methods. It could be extended to cover many buttons. It would need running initially by an ipc.macro() call in **ipcReady.lua**.
TileSix.lua	This is a very simple example of the use of the **ext.position** function (in the **ext** library) to tile six undocked FS windows on a second screen.
log lvars.lua	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.

Display vals.lua Continuous on-screen displays of some aircraft variables. Undock the window for 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].

Fuel737.lua
Payload737.lua These two examples demonstrate the **ipc.keypressplus** function, which can send keypresses to FS which even work in Menus, and when FS doesn't have the focus —the function provides options for changing focus there and back. The examples are merely editing fixed values into the default 737 fuel and payload menus, respectively, but could be generalised with more sophisticated Lua programming.

Testsrvr.lua
Testclnt.lua A pair of programs, Server and Client, which can be run together (start the server first) to test / demonstrate the LuaSockets facilities built into FSUIPC. As defined they run in the same FS session (host is defined as "localhost"), but you can change this to run between two PCs running FS if you like, or simply run one of them directly under the Lua stand-alone interpreter—but in the last case you'd need to take care of the correct LuaSockets installation.

SlaveServer.lua
MasterClient.lua Another pair of LuaSockets demos. These are more eye-catching when used, but you do need two PCs running FS. You'll need to edit the Host name in both to be the name of the Server PC, which will have its user aircraft slaved to the Client, which acts as Master. It works quite well for a rather crude un-optimised implementation—not smooth, but not as jerky as I thought it would be. If one PC is more powerful than the other it works best with the more powerful acting as the Master Client. The Salve is best put into Slew mode, though it will work in normal flight mode (jerkier) and may well work okay in Paused mode.

gfdDisplay.lua Test program for all known and connected GoFlight devices, using the **gfd** library.

VRI_SetMach.lua An example plugin for the VRInsight MCP Combi to allow it to be used with Mach mode speed control as well as IAS. This is loaded and run automatically via the FSUIPC VRInsight facilities, set up as explained in another document included with this package entitled "**Lua plugins for VRInsight devices**"

VRI_SetBaro.lua An example plugin for the VRInsight M-Panel to allow it to show the altimeter BARO setting in millibars (hectoPascals) as a switchable alternative to inches. This is loaded and run automatically via the FSUIPC VRInsight facilities, set up as explained in another document included with this package entitled "**Lua plugins for VRInsight devices**"

F1MustangSwCtl.zip Contains a Lua plugin (for FSX only) to emulate certain mouse click controls for the Flight1 Cessna Mustang subpanel and console switches, along with an "ipcReady.lua" to show one way to get it loaded ready for use. Contributed by G C McMillen, with thanks.

ThrustSym.lua, ThrustSym4.lua, SyncAxis.lua
Lua plug-ins which actively synchronise thrust settings (and other levers in the case of "**SyncAxis**") by setting the best intermediate value when the levers are close enough (within a specified distance). *These have been kindly donated by support forum user "Muas", with thanks.*

WP6.lua A demonstration of the indicator brightness and colour setting facilities for the GoFlight GF-WP-6 module.

MyDisplay.lua For WideClient only (6.895 or later). This is an example for the display library functions, displaying and updating an assortment of FS values, including, if Radar Contact 4 is running, a decode of the waypoint and runway lines from RC's menu. Just place the file into the same folder as your WideClient. It will start when FS is ready to fly and WideClient is fully connected.

TextMenu.lua	A plug-in for WideClient (6.997 or later) demonstrating the **event.textmenu** function. It handles client displays of FSUIPC texts (like Radar Contact menus) and SimConnect texts and menus arriving from SimConnect applications. Note that this will not work as menu functionality via SimConnect is currently not implemented in the MSFS SDK.
mrudder	Rudder control by mouse: An example of the use of the mouse library in conjunction with the extensive mouse events in the event library (FSUIPC4 only). The rudder control action can be enabled and disabled via a button or key assignment, and operates when the right mouse button is held down. The mouse position is returned after action, and whilst the rudder control is operating the horizontal position of the pointer in the FS screen shows the rudder position.

Additional sources of useful Lua plug-ins

One place where many users have posted useful working plug-ins, plus assistance for others, is the **User Contributions** sub-forum in my Support Forum on SimFlight. Here's a direct link:

http://forum.simflight.com/forum/143-user-contributions/

Perhaps, when you produce something useful, you too could add it to this Forum so that others can also enjoy your work?

Other Lua libraries which may be useful with FSUIPC and Flight Simulator

Often Lua libraries are provided as Lua files or DLLs, and have to be brought into your project using "require". When using these you should create a sub-folder in the FSUIPC installation folder called "Lua" or "DLL" (either will work). Place the additional libraries into one of those -- ideally Lua libraries into the Lua subfolder, and DLL libraries into the DLL sub-folder (though DLLs can also go into the Lua subfolder if you wish). Libraries written in Lua should not normally be placed into the FSUIPC installation folder itself because then they would be assignable in FSUIPC assignments drop-downs, and running them directly in FSUIPC just won't work.

Here are links to some libraries which folks have told me about:

http://www.super-hornet.com/download/	**saitek.dll**, a library for Saitek display units, by Chris Apers.
http://ittner.github.com/lua-gd/	**gd.dll**, a library for graphics, by Thomas Boutell and Alexandre Ittner.
https://github.com/davidm/luacom/	**luacom.dll**, allows Lua programs to use and implement objects that follow Microsoft's COM (Component Object Model) specification and ActiveX technology. For a good example refer to my Support forum's **User Contributions** subforum, the item on Pokeys Device interfacing by "**tlhflfsx**".

Note that these are highly technical and aimed at reasonably experienced programmers.