

FSUIPC: Lua Plug-Ins

(For FSUIPC4/ESUIPC version 4.60 and later, or FSUIPC3 version 3.98 and later)



This document is part of the Manual for the LUA plug-in facilities first added to FSUIPC4 at version 4.211, and FSUIPC3 at version 3.841. 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 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).

In addition to the built-in libraries, full **LuaSockets** support has been included, with all the major modules also built in. This is a package by Diego Nehab, and thanks are due to him. For reference data and the full package, go to

<http://www.tecgraf.puc-rio.br/~diego/professional/luasocket/>

These are the built-in (pre-loaded) modules:

socket.core	the code module DLL, supporting the Sockets facilities
mime.core	the code module DLL, supporting the Mime facilities
ltn12.lua	ancillary routines used especially by Mime
socket.lua	helper routines for Sockets, and the man module called by applications
mime.lua	helper routines for Mime support

Note that these modules, though built-in, are *not* automatically enabled (loaded) ready for direct use in your Lua plug-ins. You still need to do

require("socket")	for Sockets support, and/or
require("mime")	for Mime support.

but you do not have to have any of the modules listed above present in the FSUIPC folders. They are all pre-loaded. If you download the full LuaSockets package you will find all sorts of other little Lua examples. You can install all the parts *not* listed above in a 'Lua' folder, within the FS modules folder, if you don't want to run the directly from FS. Among the goodies you'll find there are these Lua modules ("modules" are loaded by **Require**), which you should place in a subfolder, **modules\Lua\socket**:

socket\ftp.lua	For FTP file transfers: use require("socket.ftp")
socket\http.lua	For HTTP web access: use require("socket.http")
socket\smtp.lua	For SMTP emails: use require("socket.smtp")
socket\tp.lua	For basic TP: use require("socket.tp")
socket\url.lua	For basic URL access: use require("socket.url")

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 FS Modules folder for Lua modules, and a Lua subfolder (modules\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 Modules folder directly, especially in FS2004 or before, as this would likely crash FS on loading.

In addition to the standard libraries, FSUIPC adds five 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
gfd	Go-Flight Device facilities: for reading GF switches, dials, levers, and setting GF displays and indicators, using the GoFlight module "GFDev.dll".
com	Facilities for handling serial port (COM) connected devices.

These 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 many 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.
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 .
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. (The airspeed setting only works correctly with FSX or ESP).
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.
Landing.lua	A (failed) attempt to show off some of the things you can do using the event library. This one <i>tries</i> 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.
Liar.lua	[For FSX only]. Demonstrates the facilities to "spooof" values being read from FSUIPC4 offsets by other applications, including other Lua plug-ins.
Testsvr.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 (actually flyable watching the Salve). 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.
GFpower.lua	A plug-in which checks various power needs (battery, avionics switch etc), and sets the brightness of attached GoFlight displays on or off accordingly, and automatically loads and runs other Lua plugins according to attached GoFlight devices. e.g "GFMCP.Lua" for a GoFlight MCP.
VRInsight_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 "
VRInsight_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 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). <i>These have been kindly donated by support forum user "Muas", with thanks.</i>