FSUIPC7

For
Advanced Users

FSUIPC Versions 7.4.12, June 2024

For changes since the previous version, please review the History document

Contents

Options in the FSUIPC7.INI file

In a user-registered FSUIPC installation, all of the interesting options can be controlled through the FSUIPC7 User Interface (UI). This is the recommended way, and allows changes 'on the fly'. Changes made in that dialogue are recorded in a file so that they are retained for the next re-load.

All options are recorded in FSUIPC7.INI, which is an editable text file initially created for you in the FSUIPC installation folder.

This section of the Advanced User's Guide deals with general options which don't occur under specific headings but are all collected in the [General] section of the file. These are only ever read during loading, unlike many of the other sections.

Only those parameters shown **underlined** are *not* adjustable within the Settings window (for a registered user).

Parameters to help some add-ons operate correctly

These are also available in an unregistered install:

AxisIntercepts can be set to 'Yes' to force the intercepting and forwarding of axis controls by FSUIPC, even if this action is not needed for FSUIPC calibration (where it will be done automatically in any case). This action will be needed for some "fly-by-wire" aircraft only.

UseAxisControlsForNRZ=No: This is a facility for the **[JoystickCalibration]** section(s) of the INI file, *not* [General]. It is a special option provided to try to cope with some different add-on practices (notably, in this case, the Wilco A320). Normally, the 4-Throttles, 4-mixtures and 4-Prop pitch calibrations result in an output with either a range which includes the reverse zone, or, if the "**no reverse zone**" option is checked, a range from 0 (idle) to 16383 (max). These are sent to MSFS using the older "????n_SET" controls (THROTTLE1_SET, etc), since these are the ones providing the reverse zone below zero.

If you set the [JoystickCalibration] INI parameter **UseAxisControlsForNRZ** to "Yes", then the NRZ (no reverse zone) option for all three axis types will use the AXIS_????n_SET controls (e.g. AXIS_THROTTLE1_SET) instead, with a range of -16363 (idle) to +16383 (Max). This will be Aircraft or Profile-specific if you set it in the appropriate calibration section of the INI file.

AxesWrongRange=No: This goes into the [General] section, and when set to **Yes** it makes FSUIPC revert to an erroneous way of providing certain control axis values in offsets. more details:

Offsets 332E–3336, and 3412, 3416, 3418 are intended, and documented, to provide the axis values in the correct range, calibrated if so set. For throttles, for example, the correct range is 0-16k for forward thrust, with negative values providing reverse. These values are then suitable for application directly to the MSFS control offsets, exactly as documented. Unfortunately these offsets were wrong for a long time, often providing the incorrect range (-16k to +16k). This went unreported and therefore unfixed during a period when some add-ons were developed which used the incorrect values.

Therefore, when this serious bug was eventually located and fixed, those add-ons stopped working correctly. So, to force these back to their old (wrong) behaviour: set "**AxesWrongRange=Yes**".

EventsViaCommands=No: MSFS axis controls ("events") are normally sent to MSFS via SimConnect. The events this refers to are any control assigned to an MSFS axis event in the Axis assignments tab.

In case this causes an issue with priorities (which can be the case in some add-ons), you can revert to using windows messages (WM_COMMAND) by adding **EventsViaCommands=Yes** to the [General] section. You won't find the parameter there by default. Alternatively, if you only want this change to apply to specific aircraft, place the line in the relevant [JoystickCalibration ...] section instead. An entry in the current aircraft's calibration section overrides any in the [General] section.

LogOptionsProtect: By default FSUIPC prevents programs or plug-ins writing to offset 3400 and so changing the Logging Options. If such control is needed, it can be allowed by setting this parameter to "No".

TimeForLuaClosing=2 sets the time allowed for Lua plug-ins to close correctly before the FS is closed. The time starts as soon as the event.terminate function has been called, assuming the plug-in uses it. The value in in seconds and can be set from 1 to 20. A longer period might help those more complex plug-ins to tidy up properly.

WideLuaGlobals=Yes can be set to 'N' if you don't need Lua globals to be copied over an enabled WideFS Network and wish to stop this to make the ipc.Set and ipc.Get functions work faster..

ProvideAIdata=Yes:. Set to No to stop FSUIPC reading any AI traffic data.
ProvideAIairports=Yes: Set to No to just stop FSUIPC reading departure and destination airport data.
These two parameters are provided to help overcome some problems, especially on some Prepar3D releases, where reading AI data for TCAS displays can upset other add-ons such as sophisticated GPS based gauges.

ObeyThrottleLowerLimit: some add-on aircraft allow the throttle to be decremented (using the Throttle Decr controls/events) past the lower limit actually allowed in the aircraft, which can result in full reverse thrust being applied. For such aircraft, adding **ObeyThrottleLowerLimit=Yes** to the [General] section of your *FSUIPC7.ini* file will prevent such throttle decrement controls/events being sent to the FS once the lower limit for the loaded aircraft has been reached. Note that you won't find this parameter there by default.

MaxCustomControlNumber: allows the maximun custom control number available for assignment to be specified. Several add-on aircraft use "custom controls". These are control numbers starting from 69632. By default, FSUIPC7 only maps, and therefore makes available, custom control numbers 69632 to 84232. If you want to use custom control numbers greater than this, then use this parameter to set the maximum custom control number needed.

General weather options

Currently the MSFS SDK provides no access to weather data, for read or write purposes.

Other general user options

InvokeFSUIPCOptionsKey: Records the Hotkey to display the FSUIPC7 main window.

DisconnTrimForAP: When this option is enabled, FSUIPC disconnects the analogue elevator trim axis input to FS whenever either the FS autopilot is engaged in a vertical mode (altitude hold or glideslope acquired), or a program, gauge or module has disconnected the elevator axis via FSUIPC (offset 310A).

Note that the setting can be overridden for specific aircraft which have specific FSUIPC joystick calibrations by setting this parameter differently in that [JoystickCalibration …] section.

ZeroElevForAPAlt: controls the option for FSUIPC to automatically centre the elevator input each time the Autopilot altitude hold mode is changed (switched on or off, including AP engaged changes too).

Note that the setting can be overridden for specific aircraft which have specific FSUIPC joystick calibrations by setting this parameter differently in that [JoystickCalibration …] section.

MagicBattery: This reduces the discharge rate on the battery, keeping the voltage from dropping. If this is set 'Yes' or 0 then no drop is allowed. If set 'No' or 1 then the battery discharges normally. Any value from 2 to 999 acts as a divisor on the discharge rate, so 2 makes the battery last twice as long, and so on. This is designed to assist in getting over the apparent error in the airliners which makes it discharge far too quickly before engine start.

TCASid: FSUIPC supplies data on the additional AI aircraft flying in the neighbourhood, for external TCAS or mapping programs to display. Normally such aircraft are identified by Airline and Flight number, if there is one, otherwise by the Tail number.

However, other types of identification string can be chosen instead. In particular, the optional labels placed on the aircraft by FS in the scenery view only shows tail numbers, so if you want to match them up you'd want to set this parameter to "Tail". The utility "TrafficLook" can show these differences in its display. The full list of options here is:

Flight	for airline+flight, or tail number, as available (default)
Tail	for tail numbers only
Type	for the "ATC type", generally only the Make
Title	from the aircraft title (in the .CFG file), truncated to 17 characters
Type+	for the type as above, truncated if necessary, plus the last 3 characters of the tail number
Model	for the model description

TCASrange: Sets the maximum range at which AI aircraft will be added to the tables for external TCAS applications. This defaults to 40 nm for both ground and air traffic, although it is fixed at 3nm when the user aircraft is on the ground to avoid affecting the performance of programs wish wish to use the information to handle taxiing or runway conflicts. Both ground and air ranges can be changed in the FSUIPC Traffic options tab. The parameter in the ini file maintains both values in the same parameter, in the form "x,y" (air,ground). A value of 0 turns off the limit altogether.

FixedTCASoptions=Yes can be added by the User if the above two settings are to remain locked, unchangeable except by editing here, in the INI file.

ThrottleSyncAll: controls whether the Throttle Sync Hot Key and added controls also operate on the Prop Pitch and Mixture values as well as throttles. This has no effect on jets and helicopters.

SpoilerIncrement: This controls the amount the FSUIPC "Spoiler inc" and "Spoiler dec" change the spoiler position on each use. The default is 512, giving 32 steps from spoilers lowered (0) and fully deployed (16383).

AileronSpikeRemoval
ElevatorSpikeRemoval
RudderSpikeRemoval: These control the options to ignore any aileron/elevator/rudder signals specifying maximum possible deflection.

ClockSync: This facility synchronises the seconds values with that of your PCs system clock. It is defaulted off (=No). Note that the synchronisation can only operate when the seconds = 0, and then it also has to set the minute. Consequently, it will only attempt to make an adjustment when the minutes difference is less than that set by the next parameter:

ClockSyncMins: The minutes difference within which FSUIPC's **ClockSync** facility will operate. This defaults to 5, but note that if you want to reduce the occasions that MSFS reloads textures, you will need to set this lower. Conversely, if you want the exact minutes value to be maintained as well as seconds, set this to 59 or 60.

UseProfiles: By default this will be set to 'Yes', but set it to 'No' if you want to use the Aircraft Specific facilities instead of Profiles. This change will only stick if you have no "Profile" sections in the INI file.

You can also set **UseProfiles=Files** which will organise your settings for each Profile in separate files. This may be much easier to manage for those who have made extensive use of the Profiles facilities. A separate document explaining all about this option is installed into your **Documents\FSUIPC7** folder.

UseAirLocForProfiles: setting this ini parameter to **Yes** will change the profile matching from using the aircraft name to using the folder name of the folder under which the currently loaded aircraft's aircraft.cfg file is located. This can provided a better match for different versions of the same aircraft, such as when using different liveries. If using this option, it is also recommended to set **ShortAircraftNameOk=No** (see below).

ShortAircraftNameOk: This is normally set to "**substring**" to make sure aircraft- or profile-specific Keys, Buttons and Joystick Calibration settings apply to any aircraft whose title matches a shorted form from within the names.

Alternatively, you can set S**hortAircraftNameOk=Yes** for matching abbreviated names starting at the first character, rather than any parts within, or **ShortAircraftNameOk=No** to only ever match full names.

ShowPMcontrols: This merely remembers the Project Magenta option setting for the assignment drop-downs.

ReversedElevatorTrim: This is probably not of any real use nowadays, as all the axes can be reversed in FSUIPC's joystick calibration facilities. Best left set to 'No'.

Note that the setting can be overridden for specific aircraft which have specific FSUIPC joystick calibrations by setting this parameter differently in that [JoystickCalibration …] section.

PauseAfterCrash: This is the Miscellaneous option to set MSFS into Pause mode after it has reloaded a flight after an aircraft crash. It allows the aircraft to be moved away from danger, getting out of a continuous loop.

SaveDataWithFlights=Never: This records the "Miscellaneous" tab option, telling FSUIPC whether to save "offset" data when flights are saved, or not, and when to reload it. The options are **Never**, **Menu**, **Auto**, and **Yes** ("Yes" being the option displayed as "always" in the options tab). IPCBIN files are always saved with flights in all except the **Never** mode.

For a complete explanation of the differences in the options, please see the User Guide. For technical folks and programmers, note that the IPCBIN files produced contain a snapshot of the entire 65536 range of FSUIPC "offsets", so all sorts of data can be read from it using a hexadecimal editor.

BrakeReleaseThreshold=75: This controls a "brake release threshold", for when your braking is controlled by toe pedals rather than by using the keyboard or joystick buttons assigned to non-axis brake controls. In the latter cases, operating the brakes automatically releases the parking brake (and possibly may also cancel autobraking action). This doesn't normally happen with brake axes being used for braking, as they are separate controls. That could be viewed as a drawback of having proper toe brake action, so this parameter is provided to set the amount of braking needed to release the parking brake.

The parameter can be set in the [General] section, for application to all aircraft, but also, or instead, in the individual [JoystickCalibration] sections so it can be set individually to suit different aircraft or profiles. The specific value overrides the general one.

The number is a percentage of total braking -- so the default is 75% *. If you set 0% it turns the facility off. Pressure on *both* brakes to at least the set level is required, and the release action is not "re-armed" until both brakes have returned to "off". The toe brakes must both be calibrated in FSUIPC.

* As a special case, unless specifically overridden in its JoystickCalibration section, the value for any aircraft with an ATC Type starting "Airbus" (in any case) is 0%, disabling the automatic parking brake release altogether.

JoystickTimeout=20: This timeout is no longer applicable except for EPIC USB devices, and may now be ignored.

AboutUserLine: A small amount of user-specified text can be displayed in one line in the box beneath the version number, date and registration confirmation in the FSUIPC options "About" tab. The text can be up to 127 characters (if they'll fit), and is specified using this parameter.

TextFileforDisplay=<filepath>
MaxTextRead=n: These parameters are related to special Lua display facilities and are described separately within the ZIP file entitled "**TextMenu display Lua package**".

SetForegroundOnKeySend=Yes: FSUIPC will try to set the focus to the MSFS main window before sending any key events to the sim.

MaxNumberOfCustomEvents: defines the maximum number of custom events (those loaded from *.evt files) that will be loaded. The default value for this parameter is 1024.

StopWAPIInMenu: setting this to **No** will prevent the WAPI interface from being stopped when in the MSFS main menu and started again when returning to a flight. This prevents additional simconnect connections being used which can sometimes cause issues due to a long standing simconnect bug where unused connections are not recycled. Note that you still cannot use the WASM facilities when in the main menu, although the WAPI simconnect connection will remain open. The default value for this parameter is **Yes**.

AdjustNavForMagVar=No: When set to **Yes**, the Nav1/2 values in offsets 0x0870 &7 0x0844 will be automatically adjusted by adding the magnetic variation held in offsets 0x0C40 & 0x0C42.

NumberOfPumps=0: This controls the request if the (indexed) simvars:
   FUELSYSTEM PUMP ACTIVE
   FUELSYSTEM PUMP SWITCH
Only the indices for the number of defined pumps will be requested and available in the relevant offsets.
This has been added as requesting these variables for aircraft that don't use the new FuelSystem or use the new MSFS FuelSystem but these are requested for too many pumps can result in the MSFS developer console menu being flooded with messages.

RunningOnClientPC=No: set this to **Yes** if running FSUIPC7 on an FS client PC. Details on how to configure FSUIPC7 to run on a client PC can be found in Appendix 4 at the end of this guide.

MaxButtonAssignments: by default, you can have up to 2048 button assignments, with indices 0 – 2099. If you would like to use higher indices, you can set this ini parameter to the number required, up to a max of 9999. Note however, the maximum number of actual entries is still restricted to 2048.

ShowSplash=Yes: set this to **No** disable the display of the splash window displayed when FSUIPC7 is started.

ExitIfEventsAreMissing=No: setting this to **Yes** will cause FSUIPC7 to exit if it is started when the WAPI is enabled but the MobiFlight *events.txt* file is missing or could not be loaded. This prevents the FSUIPC7.ini file from being re-written with assignments to presets removed, as can happen as the mapping between preset name and associated control number is missing in such circumstances.

OpenOnStart=No: setting this to **Yes** will cause the FSUIPC7 main window to open on start-up. This option can be controlled via the *Options->Open on Start* menu item. Note that FSUIPC7 will still open on start-up when a new version os available for download (indicated by a message in the FSUIPC7 main window) even when this is set to **No**. To prevent this, you can set this option to **Never**.

DetectToConnectDelay=1: this is the delay (in seconds) between when MSFS2020 is detected and when FSUIPC7 will attempt to connect, used when FSUIPC7 is manually started.

DetectToConnectDelayAuto=60: this is the delay (in seconds) between when MSFS2020 is detected and when FSUIPC7 will attempt to connect, used when FSUIPC7 is auto-started by MSFS. If your MSFS installation takes a long time to start-up (i.e. get to the initial main menu state) then you may find FSUIPC7 is reconnecting several times during this period. To reduce these re-connection attempts, you an tune/increase this parameter to delay the initial connection attempt. You can also reduce this parameter if your installation starts quickly.

StartUpTuningActive: If this is set to **Yes**, then auto-tuning will run the next time FSUIPC7 is auto-started (unless the **StartUpTuningDoneVersion** ini parameter is set to **-1** – see below). You can add this manually to force an auto-tuning, and this can be added automatically by FSUIPC7 if a start-up connection issue is detected.

StartUpTuningDoneVersion: this parameter is added to the *FSUIPC7.ini* file once auto-tuning has been performed, and is used in by the auto-tuning algorithm to determine if it should activate or not. You can also manually set this to a value of **-1** to completely disable auto-tuning.

IgnoreDevice: this ini parameter can be added to the [JoyNames] section of the FSUIPC7.ini file to instruct FSUIPC7 to ignore a specific device based on the vendor and product ids of the device. The format of this ini parameter is
   IgnoreDevice=<vendorId>,<productId>
where **<vendorId>** is the hexadecimal id of the vendor and **<productId>** is the hexadecimal id of the vendor, as reported in the FSUIPC7.log file (for example), preceded by '0x' or 'x'. This can be used to either ignore a specific device, when both ids are given, or to ignore all devices from a specific vendor, when only the **<vendorId>** is specified. This can be used, for example, to ignore a phantom xbox controller entry that is sometimes detected (for some users) when FSUIPC7 is started when MSFS is starting-up. To ignore such a device, add the following:
   IgnoreDevice=0x28DE,0x11FF

Less used technical options

AutoTuneADF: This controls an option to 'auto-tune' the ADF radio. If this is enabled, when FSUIPC detects no NDB signal being received it alternates the fractional part of the ADF frequency between .0 and .5 every seven seconds or so. This allows external cockpits built with only whole-number ADF radio facilities to be used in areas like the U.K. which have many NDB frequencies ending in .5.

AxisScanOnSimConnectOpen=No: This parameter can be changed to **Yes** to instruct FSUIPC7 to perform an additional scan of your joystick axes when a SimConnect connection is opened. This may help if you axes are not initially recognized when MSFS is started. However, beware setting this parameter as it is also known to cause a CTD in some situations (this is currently under investigation).

AxisCalibration: This facility deals with inputs to the rudder, aileron and elevator axis offsets, via the IPC offsets. It is intended for use with hardware drivers which, instead of sending normal axis inputs to FS, control the main flight surfaces by direct writes to FSUIPC's offsets, thus bypassing assignments, calibration, etc. The only such hardware driver known to me is the one for the Aerosoft GA28R console. *In FSUIPC this facility still operates for compatibility with FSUIPC4 and before. However, you are advised now to leave this as "No", and instead use the new facility 'DirectAxesToCalibs':*

DirectAxesToCalibs: Setting this to 'Yes' makes FSUIPC assume that any direct writes to FSUIPC's offsets for rudder, aileron and elevator are from a hardware driver and are really meant as axis inputs. FSUIPC directs the values to its Joystick Calibrations section, where you should then calibrate the inputs exactly as you would for a normal flight control. The only hardware known to me which benefits form this is the one for the Aerosoft GA28R console. Do not set this option if you use any sophisticated panels or external programs with their own autopilots, as it is possible that they route their control values the same way. FSUIPC cannot distinguish the source.

To also allow for brakes to be sent to the Joystick Calibration, this value can be set to 'All'.

DisableMSFSMonitor: this disables FSUIPC's MSFS (window) monitor, which disconnects (and possibly closes) FSUIPC when MSFS can not be found. Some users have reported this issue on Windows 11, with FSUIPC7 closing (or repeatedly disconnecting) even though MSFS is still running. Setting this option to **Enum** can prevent this issue by using a different method (window enumeration) to find MSFS, and setting to **Yes** will disable the MSFS monitor. As of FSUIPC version 7.3.7 (and later), windows 11 should be detected and this option automatically set to **Enum**, although you can still override this by explicitly setting this parameter in your *FSUIPC7.ini*.

KillLuasOnSimStop: setting this parameter to **Yes** will kill all luas as soon as a SimStop event has been received. Only to be used if you have issues with Lua threads hanging when FSUIPC7 is stopped.

StartImmediately is not expected ever to be used directly by the user. When it is set to 'Yes' it makes FSUIPC initialise the data interface with SimConnect immediately it is started, rather than wait for SimConnect to indicate "SimStart".

NormalStallTime=1, and **InitialStallTime=30** can be used on poorer performing systems, very heavily loaded, to attempt to stop FSUIPC re-initialising the SimConnect interface to MSFS when it finds itself starved of information which should be arriving all the time.

The Initial one is used during initialisation, the other the rest of the time. This parameter will be tuned by FSUIPC7 via the auto-tuning facility – see this section for further details.

The range of values that can be set for these parameters is -100 to 100. The negative values give the same timeout, but only result in a log entry informing you of the problem rather than an attempt to reinitialise the SimConnect connection.

TrafficStallTime=1: This is similar to NormalStallTime, but only operates on the AI Traffic data collection interface to SimConnect. The value can be 1-10. There is no separate option for a log-only action.

UseEpsilon is normally omitted altogether. If needed, it can be added, in the [General] section of the INI file, set to "Yes". This will impose change limitations on most variables sent to FSUIPC by SimConnect, so lessening the load on that interface but giving a little less precision in some values.

FiddleMachForPM=Yes is, hopefully, a temporary fix for Project Magenta (PM) users who suffer the problem of high speed descents under MCP control as a result of PM incorrectly setting a Mach speed in the wrong mode. This only affects MSFS adversely because of the odd way FS9 and before worked, for which PM was originally designed. To enable this fix add this parameter (it won't appear in the INI by default).

By way of explanation, when the PM MCP sets the IAS for the A/P it also sets the Mach value. This would be okay, except for three things:

1. The Mach value it sets after cruise is ALWAYS the last one it set in cruise, not a correct one related to the IAS and altitude/temperature/pressure.

2. The Mach is written after the IAS

3. In FS9 and before, the Mach setting did not affect the IAS setting nor vice versa until and unless you switched IAS/Mach modes. However, in FSX and later the last-written value applies and is converted to the equivalent IAS/Mach no matter which mode you are in!

This FSUIPC7.INI parameter simply makes FSUIPC discard any MACH writes whilst the PM MCP is in IAS mode.

FiddleAppAltForPM=Yes is another hopefully temporary fix for Project Magenta. It makes FSUIPC automatically replace any altitude written during PM MCP APP mode by zero. It also sets the MCP altitude to zero in PM MCP APP mode when a negative VS is set, and it does both these things even if Altitude Hold is enabled.

The intention here is to avoid any unwanted climbs when descending on the GlideSlope due to the fact that FSX seems to be different to FS9 and before in that writing to the FS MCP's altitude register can affect the requested vertical speed even though FS's altitude hold option is not enabled.

LuaPath=<*path*>: allows Lua files to be indexed for assignment into a separate folder, anywhere on the same PC. This parameter should be placed in the *[LuaFiles]* section of your ini file. It can be a sub-path from the installation folder (in which case just give the sub-path), or a full path anywhere elsewhere on the same PC (determined by seeing a ':' character in the path, denoting a drive spec).

You either have *all* the *assignable* Lua files in the Modules folder, or in another. The limit is still 127 and the numbering in the [Luafiles] section of the INI will still be based on the order of the Lua files in the folder, as discovered initially (i.e when they first appeared). The numbering will stay the same if you merely copy all the Lua files out of the installation folder and into the new one.

LuaRerunDelay=66: This parameter sets the time, in milliseconds, which is imposed between re-runs of the same Lua plug-in. It is a safety precaution against repetitive execution causing an FS crash by stack overflow. For plug-ins which take more than that amount of time to load and execute this effectively restricts the repeat rate from dials to 15 times per second. The rate from buttons or keypresses being held down was already restricted as for those the repeat is not effective until the current plug-in execution finishes.

Console=No and **ConsoleWindow**: This records the Logging option for a real time console window copy of the Log file, seen when FS is run in Windowed mode. The ConsoleWindow parameter records its position and size.

LuaTrapKeyEvent=No: By default, key events trapped by the lua *event.key* function are not passed on to FSUIPC for assignment activation after processing. Setting this parameter to **No** allows this, and and keys processed will also be processed by FSUIPC's key assignments. Note that with this set (to **No**) there is also fine-grained control for this in the flags used for **event.key**.
Note that it is not possible to stop MSFS key assignments being triggered with this parameter. The only way to prevent MSFS assignments being triggered on key presses is to give the focus to another program - an FSUIPC added control is provided to do this – **Key Focus FSUIPC** (control number 1156).

TrapMSFSkeys: FSUIPC requests all key presses & releases from standard keys (i.e. not modifier keys) to be received via the MSFS SDK SimConnect interface. Setting this parameter to **Yes** (the default is **No**) will tell MSFS that FSUIPC will *mask* this event, and np other lower priority clients will receive it. Note that this will NOT prevent MSFS assigned keyboard events from being processed.

DontResetAxes=Yes: Normally using the Axis assignments dialogue in FSUIPC options does not affect the surfaces controlled by those axes. If this parameter is set to 'No', the memory of the previous axis values is reset to zero, which may therefore cause a movement in those axes when returning to flight mode, but does, on the other hand, ensure a correct value when the axis function is changed.

NoActionOn7B91: This can be added to the [General] section of the FSUIPC7.INI file, and set to 'Yes', to prevent FSUIPC setting the SquawkBox 4 transponder mode when an external program writes to offset 7B91. This gets over a problem with the OpenCockpits driver, in particular, which seems to try to handle both SB3 and SB4 transponders by writing to 7B91 and dealing direct with SB4 at the same time. The FSUIPC parameter does not stop the FSUIPC-added SB4 transponder controls from operating, however.

MaxSteerSpeed=60: This parameter appears only in [JoystickCalibration] sections, and deals with FSUIPC's facility to 'blend' its steering tiller control into rudder control as speed increases whilst on the ground. Both tiller (the FSUIPC direct control, not MSFS's own), and rudder need to be assigned in FSUIPC by the "direct to FSUIPC calibration" method, and both be properly calibrated for any blending to be active.

The **MaxSteerSpeed** parameter includes more complex facilities to restrict the rudder effect in different groundspeed ranges. The simplest of these keeps the rudder at 10% of its input until half way to the full threshold speed, then increase linearly to 100%. This is intended to make reasonably easy to check the rudder pedals whilst taxiing without causing bad swerves, and also allows some use of rudder even at very slow speeds at the end of the landing ground roll. The value of 10% minimum comes from the 737NG where at taxi speeds the rudder deflection is a maximum of 7 degrees compared with 67 degrees fully.

To make FSUIPC do this blending instead of the normal 0-100% linear method, just change the **MaxSteerSpeed** parameter in the relevant [JoystickCalibration] section of the INI file to a negative value, eg -60 for the default 60 knot threshold.

A more complex specification can be provided which allows the user even more scope. The **MaxSteerSpeed** parameter can be given as **MaxSteerSpeed = Qn1,n2,n3,n4** where n1 to n4 are numbers used as follows:

- If n1 is not zero, then rudder effect is 0% (ie eliminated) until a groundspeed of n1 knots. Then the effect rises linearly from 0% at n1 knots to 10% at n2 knots.

- If n1 is zero, then rudder effect is 10% until the groundspeed reaches n2 knots. n2 is not allowed to be zero.

- If n3 is not zero, then rudder effect rises linearly from 10% at n2 knots to 30% at n3 knots, then linearly again from 30% at n3 knots to 100% at n4 knots.

- If n3 is zero, then rudder effect rises linearly from 10% at n2 knots to 100% at n4 knots. n4 is not allowed to be zero.

Note that apart from the option for n1 and n3 to be zero, n4 > n3 > n2 > n1. You should see that the option: **MaxSteerSpeed=-60** is in fact the same as specifying **MaxSteerSpeed=Q0,30,0,60**.

There is one shortcut. **MaxSteerSpeed=Q** is the same as specifying **MaxSteerSpeed=Q10,20,30,60**

You can also set this parameter dynamically by specifying an offset value preceeded by an 'x' character, e.g.
>   **MaxSteerSpeeed=x66C0**

The offset specified is read as a two-byte signed word and can be updated dynamically. There is also an additional FSUIPC provided control that can be used to change this parameter dynamically.

RudderBlendLowest: This is an optional parameter which can be added to any of the [JoystickCalibration] sections of the INI file (not [General]). It operates with the blending (described above) of an FSUIPC steering tiller axis and the rudder during calibration. Previously, because of the way the blending operated, the rudder pedals used to have no effect when the aircraft is stationary, or nearly so. This meant that rudder operation checks--those on screen within the cockpit (along with the other control surfaces), or via viewing the rudder from outside–meant operating the steering tiller instead of the rudder pedals.

The **RudderBlendLowest** parameter fixes that by giving a ground speed below which the rudder blending is not taking place and only the rudder input is used. *This speed defaults to 1 knot, which means the aircraft just needs to be stationary.* So there is normally no need to change this parameter.

TransmitErrorsReconnect: This optional parameter allows you to configure the number of errors received when transmitting events to the FS before a SimConnect re-connection is performed. The default for this parameter is 5.

VRIDisableCMDRST: This optional parameter in the [General] section of the INI can be set to 'Yes' to disable the sending of the CMDRST call for VRI devices.

UseKeyboardHook=No: when set to **Yes**, a global keyboard hook will be installed to receive all key strokes directly from windows, and keyboard input will not be requested via SimConnect (unless running FSUIPC7 on a client PC). This can (possibly) help with certain controllers that function by sending key presses.

InitialKeyRepeatRate: This optional parameter in the *[Keys]* or profile specific *[Keys.xxx]* sections of the INI can be set to the number of milliseconds that pass until a first key repeat is recognised. The default value is 250 (ms).

SubsequentKeyRepeatRate: This optional parameter in the *[Keys]* or profile specific *[Keys.xxx]* sections of the INI can be set to the number of milliseconds that pass until a second and subsequent key repeat is recognised. The default value is 100 (ms).

Auto-tuning of initial start-up ini partameters

FSUIPC7 contains a facility to auto-tune certain start-up parameters to facilitate quicker an error free connection to the simulator. This facility is active by default, but can also be disabled and tuned manually.

Due to the variable start-up times of MSFS for each user (mainly due to the number of add-ons it loads, but also related to your PC specification and the aircraft loaded at atart-up), long start-up times can cause issues when FSUIPC7 is auto-started, or when started during the MSFS initialisation phase. This is because a response is not received from the initial connection request, and this response times-out after a pre-defined (and configurable) time-limit, and a new request is sent. A few re-connection attempts at start-up should not cause any issues, but too many attempts can starve SimConnect of available connections and then prevent other add-ons/simconnect clients from connecting/functioning correctly. FSUIPC7 will therefore try and tune some parameters for optimal performance and to prevent issue.

There are three parameters that can be tuned to get FSUIPC7 to connect at the right time and to avoid re-connection attempts. These are:

- **DetectToConnectDelayAuto**: this is the time/delay used, in seconds, between when FSUIPC7 detects that MSFS is running and when it first tries to connect, and is used when FSUIPC7 is auto-started (by MSFS) only. This has a default value of 60 (seconds), so FSUIPC7 will attempt to connect 60 seconds after it has detected that FSUIPC7 is running. This parameter has a maximum value of 500 seconds,

- **DetectToConnectDelay**: similar to the above, but used when FSUIPC7 is manually started. This has a default value of 1 (second), which is designed for use when FSUIPC7 is manually started and MSFS is already loaded, i.e. in the main-menu system or with an aircraft loaded and ready-to-fly. You should adjust this to the same default as **DetectToConnectDelayAuto** if you start FSUIPC7 before you start MFS, or if you start FSUIPC7 during the MSFS start-up process. This has a maximum value of 300 seconds.

- **InitialStallTime**: this is the time taken between issuing a connection request and canceling/closing this request when a response is not received,. This has a default value of 20 (seconds), and a minimum value of 15 (seconds) when FSUIPC7 is auto-started. What this means is that FSUIPC7 will close/cancel the connection request and issue another connection request if a response is not received in 20 seconds (by default).

All of these parameters go in the [General] section of the FSUIPC7.ini file, although some may not be present by default.

Auto-tuning will configure these parameters when active, and will set them for use in subsequent auto-starts. Auto-tuning will activate under the following circumstances, and only when FSUIPC7 is auto-started wither via the *MSFS.bat* file (default method since version 7.4.12) or the *MSFS EXE*.xml file:

1. When the default value of the **DetectToConnectDelayAuto** parameter is being used and auto-tuning has not already been performed

2. When explicitly instructed to do so, by adding the following ini parameter to the [General] section of your *FSUIPC7.ini* file:
   StartUpTuningActive=Yes

3. If I detect a major difference in start-up times with a new MSFS release (either increase or decrease), then I can force auto-tuning to be ran in a new FSUIPC7 release.

4. If some re-connection attempts are detected during start-up, then auto-tuning will be set to run on the next auto-start of FSUIPC7.

You can also completely disable auto-tuning by setting the value of the ini parameter **StartUpTuningDoneVersion** to a value of **-1** – add this to the *[General]* section of your *FSUIPC7.ini* file if not there or update the value if this parameter is present, i.e.
   StartUpTuningDoneVersion=-1

Note that the auto-tuning process terminates (that is, when the auto-tuning parameters are updated) when MSFS arrives at the main-menu state. If FSUIPC7 is not yet connected to MSFS when this occurs, then the auto-tuning process will not terminate. This is not an issue – it will terminate (and the required parameters updated) when you next go back to the MSFS main menu. If you exit before going back to the main menu, auto-tuning will run on the next auto-start of FSUIPC7.

AUTOSAVE: INI-file only options

AlsoManage, for additional files

Some add-on programs produce files when Flights are saved separately from the usual ones in the MSFS flights folder, so that the AutoSave option fails to manage their numbers, deleting older ones when the FLT files are deleted. The types handled by default are FLT, FSSAVE, SPB, PSS, FMC, ABL, RCD. PNL and IPCBIN. For any others, and files in other folders, you have to manually add some lines to the [AutoSave] section of the FSUIPC7.INI file.

Give the complete path name, from the drive (e.g. C:\ ...) onwards (or the computer name for a Network in the usual form, i.e. \\<name> ...).

Up to 64 "AlsoManage" lines can be given, numbered 1 to 64.

AlsoSave, more variable and informative naming

The "AlsoSave" flight saving facility can operate with a filename containing special values, using the following:

%A   for the Aircraft name. This is the aircraft title but stripped of every character other than alphabetics and numerics.
%D   for the Date in the form YYYYMMDD
%T   for the Time in the form HHMM (no seconds)
%W   for the day of the Week (MON, TUE etc).

The A,D,T and W can be upper or lower case.

Note that, unlike normal autosaved files, FSUIPC never erases any of these, so don't set a short time interval and include the Time in the name—otherwise your disk directories will become very long quickly, and slow things down.

Logging facilities

These options can be controlled 'on the fly' from the FSUIPC7 main window **Log** menu entry

FSUIPC always produces a text file called FSUIPC7.LOG in the installation folder. Entries in the log are timed, from the start of the FS session. The time is in milliseconds and appears on the extreme left of each line.

Please use the logging facilities to check things before reporting problems or omissions in FSUIPC, and supply an appropriate log file (or extract) properly zipped up with such reports.

Note that log files can get very large if all the options are turned on. Keep test flights short. You can read log files whilst flying provided you use a reader which shares access (like recent Notepad programs), or use the 'NewLogKey' described below to close logs and start new ones.

All Log control parameters go into the [General] section of FSUIPC7.INI. None are included by default. However, most logging options can be controlled via the **Log** drop-down menu entries.

LogWeather=Yes: Logs weather data. This will log incoming data, set by a weather control program and the actual weather data constructed by FSUIPC in FS terms. Then you get the weather read out by FSUIPC and lastly placed back into the Offsets for applications to read. Incoming weather control data on the Advanced Weather and New Weather Interfaces is also logged in full.

However, not that as there is no interface to MSFS weather at the moment, this option will not log anything and should not be used.

LogWrites=Yes: Logs the offset 'writes' received from applications, with global offset address and data size, plus all bytes of data. The offsets shown are the ones used by the application. [Take care: the Log file may get very large!]

LogReads=Yes: Logs the offset 'reads' received from applications, with global offset address and data size, plus all bytes of data. The offsets shown are the ones used by the application. [Take care: the Log file may get very large!]

LogEvents=Yes: This option logs all FS "key events", other than those from axis controls. This can be very useful to those seeking to understand the actions of their buttons and keys, or to view the sorts of things some of the more complex panels do, repeatedly, every second.

LogAxes=Yes: This logs just the axis input events.

LogButtonsKeys=Yes: This logs most Keyboard events (KEYUPs only when programmed), and all button operations. The logging can get quite long, but it will be very useful when trying to analyse exactly what your complex FSUIPC button or key programming is doing.

LogLua=Yes: Enables extra Lua plug-in logging, and causes each running Lua plugin to use its own Log file. These files are cumulative, though—each time the same plug-in runs it adds to an existing Log file, unless the following **LogLuaNew** option is set.

LogLuaNew=Yes: setting this option will create a new lua log file, when **LogLua** is set, rather than appending to the existing log file for the lua script.

DebugLua=Yes: Enables Lua tracing automatically, for all Lua programs. this is now preferred over the "LuaDebug" method of starting Lua plug-ins.

LogExtras=Yes: This logs additional technical data about the inner workings of FSUIPC, the nature of which will vary from time to time according to needs. There is nothing here that would be of interest to the user, but when investigating problems users may be asked to enable it so that the logs returned can be more meaningful in solving them (especially as this now also logs thread Ids).

LogSimC=xxxx,xxxx ... (where each 'xxxx' is either an offset, or a range xxxx-xxxx): whenever the values associated with offsets listed or included here are read from or written to a SimConnect Variable ("SimVar") those values are logged. The list can request several disparate offsets or ranges—the limit is imposed only by the INI file maximum line length (255 characters).

DontLogThese: In order to make it easier to check Events in the FSUIPC7.LOG file when using aircraft such as the PMDG 737NGX, which appear to be sending some events continuously all the time they are loaded, this INI file parameter is available to avoid logging specific event numbers. You can list individual events (by their decimal control number), or a range (n-n), inclusive of the end points. Each is separated from the next by a comma.

For example, for the PMDG 737NGX it seems the following is a good value for this parameter to avoid logging thousands of entries and flooding out any useful information:

        DontLogThese=66485,69000-70999,66503,66504

The 66485 is "Anti ice toggle Eng2" which appears to be sent at a rate of about 15-16 per second! The large range 69000-70999 maybe too big, but prevents non-FS controls in that range being sent at similar rates whenever the Mouse pointer rests over a switch or button! Finally the last two merely suppress the two axis events for "mouse look" being logged.

Note that, unlike all other parameters in the [General] section, you can modify this one without closing FSUIPC and reloading it. Simply go into FSUIPC options and come back out to get it re-read.

This ini parameter can also be used in your [Profile.*xxx*] sections to be used when that profile is loaded only.

DontLogTheseIE: this is similar to the **DontLogThese** ini parameter, described above, but is for Input Events. Specify the Input Events names, separated by a comma. This ini parameter can also be used in your [Profile.*xxx*] sections to be used when that profile is loaded only, which is the recommended place as Input Events tend to be aircraft-specific.

NewLogKey, StopLogKey: These allow you to assign keypresses to close the current Log file (if logging was enabled), and start a new one. The 'NewLogKey' will carry on with the same logging options, whilst the 'StopLogKey' will revert to default logging (the minimum). Between them these two keys give complete control over the logging. (Note that both actions are also available in the FSUIPC dialogue window).

The current log file is always called FSUIPC7.LOG. The others are named in numerical order FSUIPC7.1.LOG, … 2.LOG, … etc. The keystrokes are defined as in Flight Simulator's own controls, and listed below in the Button Programming section. For example, I use "Shft+Ctrl+L" and "Shft+Ctrl+O" (for "Log" and "Off" respectively) which would be

        NewLogKey=76,11

        StopLogKey=79,11

EnableNewLog: the **New Log** menu item (under the **Log** menu) has been disabled, by default, in versions 7.3.4 and later. If you would like to use this feature, you can add **EnableNewLog=Yes** to the [General] section of your *FSUIPC7.ini* file. If you enable this function, please do not use it when sending logs for support issues, as a full log file (and not a *continuation* log file) is always needed.

LogLuaNew=No: set this to **Yes** to start a new lua log file each time a lua script is started. Only applicable when lua scripts are logged to separate files.

Monitor facilities

FSUIPC can monitor, on every FS frame, up to four values (or the same values in different formats, if needed), and display or log them when they change. You can access this from the **Log → Offsets...** menu entry. For each value to be logged you enter or select three things:

Offset: which identifies the position of the value. This is a hexadecimal number, normally in the range 0000 to FFFF. For a list of the offsets, see the Offset Status document.

Type: this defines the type of variable, so that the formatting in the display will show something meaningful. The types currently supported are tabulated below.

Type	Description	C type	Type	Description	C type
S8	Signed 8-bit value, -128 to +127	**signed char**	UIF32	4 byte Integer & Fraction: 16-bit fraction followed by 16-bit unsigned integer	Uses an **unsigned int**
U8	Unsigned 8-bit value, 0 to 255	**unsigned char**, or **BYTE**	SIF64	8 byte Integer & Fraction: 32-bit fraction followed by 32-bit signed integer	Uses an **unsigned** then signed **int**
S16	Signed 16-bit (2 byte) value	**short**	UIF64	8 byte Integer & Fraction: 32-bit fraction followed by 32-bit unsigned integer	Uses two **unsigned int**s
U16	Unsigned 16-bit (2-byte) value	**unsigned short**, or **WORD**	FLT32	32-bit (4-byte) standard floating point value	**Float**
S32	Signed 32-bit (4-byte) value	**int**	FLT64	64-bit (8-byte) standard floating point value	**Double**
U32	Unsigned 32-bit (4-byte) value	**unsigned int**, or **DWORD**	ASCIIZ	A string of single-byte characters terminated by a zero byte. A length an limited number of these is shown	**Char[]**, or **ASCIIZ**
SIF16	2 byte Integer & Fraction: 8-bit fraction followed by 8-bit signed integer	Uses a **short**	SA16	16-bit signed Angle in FS format (-180 degrees = max+1)	Uses a **short**
UIF16	2 byte Integer & Fraction: 8-bit fraction followed by 8-bit unsigned integer	Uses an **unsigned short**	UA16	16-bit unsigned Angle in FS format (360 degrees = max+1)	Uses **unsigned short**
SIF32	4 byte Integer & Fraction: 16-bit fraction and 16-bit signed integer	Uses an **int**	SA32	32-bit signed Angle is FS format	Uses **int**
			UA32	32-bit unsigned angle in FS format	Uses **unsigned int**

Hex: For most numerical values the sensible display will be decimal. However, for the plain fixed point integer values (S8, U8, S16, U16, S32 and U32) you may want to view them in hexadecimal instead.

Then you have to select how you want the values to be displayed. There are four options, and any or all of these can be selected:

Normal Log File: Changes in the monitored values are listed in the FSUIPC7.LOG for later viewing. Additionally, for any monitored offset, the offset is also treated as a "**LogSimC**" offset (see above) automatically so that SimConnect reads/writes are logged.

Debug String: The same messages are sent to a debugger or debugging monitor such as DebugView, for viewing in parallel to the FS actions.

FS Window: ~~The monitoring is done by using up to 4 lines in the FS message display window. This appears near the top of the screen.~~ Currently not available due to issues with the SimConnect text facilities.

FS Title Bar: The messages replace the FS title altogether. Only one is shown at a time, so this is only useful for monitoring one value.

Note that at least one of these options needs selecting for the value to be monitored, and if it is monitored *and* is a numeric (i.e. not ASCIIZ) then its value is also available in an FSUIPC offset for use by Lua or GFDisplay or other programs to display on hardware when needed. The values relating to the four Monitor slots are provided in 32-bit **float** form in offset 03A0, 03A4, 03A8 and 03AC, respectively, and already converted to reflect the type specified.

If the value requested is not available at any time the result will show "<invalid>". When looking at some Engine or other aircraft things, this can happen transiently, for instance whilst an aircraft is being loaded.

All the monitoring selections are saved in the FSUIPC7.INI file, in a section called [Monitor].

This section can also include up to 4 parameters in the following format:

> **Monitor*N*=*xxxx,yyyy***

where ***N*** is an integer between 4 and 7, and ***xxxx***, ***yyyy*** are two offsets defining an area of the 65KB offset data. The area is inclusive of both of these offsets.

These offsets will be logged when changed, but only if the **Normal Log File** option is set in the Monitor section of the logging tab (see above).

JoyNames

The INI file section [JoyNames] is fully described in its own chapter in the User Guide.

Profiles

If you opt to use the Profile facilities, to have different button, key, axis and calibration settings for a number of types of aircraft, then FSUIPC will create [Profile.<name>] sections in your INI file. These take the name of the profile you request, for example "Jets", "Props", "Helos", and simply contain a list, in the usual 1=<name>, 2=<name> ... format, of those aircraft names which belong to the particular profile, according to your assignments. Those aircraft names may be the full names, as when you assign in the FSUIPC options dialogue, or can be shortened or substring names.

For extensive use of Profiles you might find it much easier to manage by using the **UseProfiles=Files** facility. This splits all of the settings for each profile into a separate file, one per profile, stored in a separate folder. For more details and full instructions please refer to the separate document about it installed in your **Documents\FSUIPC7** folder.

Button Programming

FSUIPC's options dialogue provides a page for programming button in all the main ways. Here we look at how this programming is encoded in the FSUIPC7.INI file, and how the programming can be extended to provide multiple keystrokes and controls for a button, mixed if required, and to provide compound (conditional) actions—ones depending on other buttons, switch settings and even previous keyboard presses. There are even facilities to make Button actions depend upon values in offsets from the FSUIPC IPC interface, which really provides a wealth of possibilities (for that part you will need to get the FSUIPC SDK too, as the offset listings are provided in that package, in the Programmer's Guide).

FSUIPC reloads all Button parameters each time the aircraft is changed in Flight Simulator, so you can edit theses and test them out without having to reload Flight Sim every time.

Before embarking on the programming itself, several global parameters need to be described. These won't appear in the INI file unless you add them, and you only need to add them (in the main [Buttons] section) if you need something other than the defaults:

InitialButton: This controls a facility to make FSUIPC perform one-off actions when FS is first loaded and running (i.e. actually ready to fly). This is by programming a real or imaginary Button. Simply add the line "InitialButton=j,b" to the [Buttons] section. The values of j (0–255) and b (0–31) can specify a real joystick and button, or a non-existent one, it doesn't matter. Real ones can have an action assigned on-line, in the Buttons option page, but multiple actions for any button, real or not, can be accomplished by editing the INI file as described here.

IgnoreThese: This can be used to list a number of buttons which are to be ignored by FSUIPC in the Buttons & Switches tab. This is to deal with faulty button signals which are repeating without control and thus preventing the others from being registered on the screen ready to program. The parameter takes this form:

> **IgnoreThese**= j.b, j.b, ...

listing the joystick number (j) and button number (b) of each button to be ignored. To make it easy, you can edit the INI file whilst in the Button assignments dialogue and simply press "reload all buttons" to activate the changes.

Note that the action of ignoring buttons only applies in the button assignments dialogue—if they are already assigned the assignment will still be effective.

You can also use the wild card '*' for the button number, in which case all buttons will be ignored. This is for convenience only, and when used this line in your ini will be re-written to contain the complete button list (numbers 0-39).

This parameter can also be used in the [Axes] section of the FSUIPC7.INI, using the axes letter instead of the button number (see later).

EliminateTransients: This can be added, and set to 'Yes', to eliminate short (transient) button press indications. This is intended to help deal with some devices which create occasional spurious button press signals. It operates only with locally-connected joysticks (but not EPIC or GoFlight devices).

Note that enabling this option may mean you have to consciously press buttons for slightly longer. It depends on the **PollInterval** (below). A "transient" button indication is one which only exists for one poll, so a real press would have to last up to 50 mSecs (twice the default poll interval) to be sure of being seen (more, allowing for variations in the polling due to processor/FS activity). You may find you need to adjust the **PollInterval**.

PollEpicButtons=Yes: Set this to No if you experience any difficulty getting FSUIPC to operate correctly on a system with an EPIC installed but which you do not want to program via FSUIPC's "Buttons" page.

ButtonRepeat=20,10: The first number here controls the button repeat rate, when repeating is enabled for a specific button. The range is 1 to 100 and is the number of repeats per second. Note that the higher rates may not actually be achievable. If you want no limit placed, allowing the repeats to go as fast as they can under each circumstance, set this parameter to 0. This can be *very* fast, so beware!

Note that it is unlikely that this rate will be exactly maintained as it is subject to FS performance variations, depending on the action being repeated, but it acts as a good target control value.

The second number gives an initial delay, before repetitions begin. This is in terms of how many potential repetitions to miss, so with 20 repeats per second, 10 would give a delay of half a second. This allows the same button to operate to increment/decrement a value just once, or, by holding the button down, repeat until released.

A value of 0 for the initial delay value means there will be no delay before the repeats start -- this is how FSUIPC has been until the delay facility was added.

PollInterval=25: This parameter tells FSUIPC how often to read ("poll") the joystick buttons. The time is in milliseconds, and the default, as shown, is 25 (40 times a second).

A polling rate of 0 will stop FSUIPC looking at buttons altogether. This may come in useful for checking whether a rogue joystick driver is causing problems.

A polling rate of 40 per second is more than adequate for all normal button programming. It is only when you come to the more advanced uses that you may want to change this. Rotary switches, for instance, may give pulses so fast that some are missed at such a rate.

Any value from 1 millisecond upwards can be specified, but those from 50 upwards result in a specific number of "ticks" (55 mSecs) being used. i.e. 40–82 actually result in 55 (1 tick), 83–138 in 2 ticks, and so on. Ticks are also approximate, in that they depend on the other activities and loading upon FS.

Values 1–59 milliseconds are actually handled by a separate thread in FSUIPC and give more accurate results, but note that polling the joysticks too frequently may damage FS's performance, and may even make its response to joystick controls more precarious. No truly adverse effects have been noticed during testing, but it is as well to be warned. If you think you need faster button polling, try values in the range 10–25, and make sure that FS is still performing well each time.

Note that the PFCcom64 and PFChid64 "emulated" joysticks (those with numbers 16 upwards) are polled four times more frequently in any case—this is done because there is no overhead in doing so—there are no calls to Windows but merely some data inspections.

KeyboardFocus: When this is included and set to 'Yes' it ensures that any keypresses sent by external programs (as FSUIPC controls) are directed to the main FS window for processing. This would normally be the case except that folks using Windows external to MSFS might be changing the keyboard focus away from the main FS window. Using a touchscreen, for instance, moves the keyboard focus even though it is the mouse which is activated by touch. Setting **KeyboardFocus=Yes** makes FSUIPC restore focus to the main FS window every time it is asked to send a keypress. The FS window will become the foreground window at the same time.

For key-presses assigned to buttons in FSUIPCs Buttons & Switches assignment dialog, FSUIPC will always give MSFS the focus so that these key presses can be received and processed.

More ambitious users may wish to retain focus elsewhere for some key presses. This can be performed using the added FSUIPC control "key focus restore" (number 1125), as listed later in the Added controls list.

FORMAT OF BUTTON DEFINITIONS

The button programming is saved in sections in the INI file. For globally operative buttons this is called [Buttons]. For aircraft-specific buttons it is [Buttons.<aircraft name>]. Up to 2048 separate entries defining button actions can be included in each section, usually numbered from 0-2047, provided that the total of the definitions in the Global section and the largest aircraft-specific section is not greater than 2048. The numbering is only important in that it determines the order of actions for multiple assignments involving the same buttons, and doesn't need to be sequential otherwise. You can increase the allowed numbering of entries using the **MaxButtonAssignments** ini parameter, used in the *[General]* section. Note that this only allows higher index numbering to be used, but the total number of button assignments is still limited to 2048.

The <aircraft name> part of the section heading can be abbreviated (manually, by editing the INI file) so that it applies to more than one aircraft. It will then be selected on a substring match to the actual <aircraft name>. Note that only the first substring matched profile will be selected and used.

The basic format of each entry in the Buttons section is as follows:

For keypresses: <Entry number> = <Action><Joy#>,<Btn#>,K<key>,<shifts>

For controls: <Entry number> = <Action><Joy#>,<Btn#>,C<control>,

For macros (see the separate section on macros):
          <Entry number> = <Action><Joy#>,<Btn#>,CM<file#>:<ref#>,

For presets (see the separate section on calculator code presets):
          <Entry number> = <Action><Joy#>,<Btn#>,CP<Preset Name>,

For Input Events:
          <Entry number> = <Action><Joy#>,<Btn#>,CI<Input Event Name>,

The format of the parameters becomes more complex for conditional actions, so they will be described later.

The <Entry number> is not material most of the time—except in sequences for single button presses/releases. It is just a sequence number from 0–2047 (but limited to a total of 2048 entries for the general section plus any one Aircraft-specific section).

Each entry must have a unique entry number, and the actual order is only important when multiple actions are defined for the same button. FSUIPC will retain the numbering, and hence the order which the number (not the line position) defines.

You can add comments following a semicolon (;) at the end of the line, and these will be retained. You can also insert lines containing only comments, but they need an <Entry number> too, otherwise they may not retain their relative position. Comments can contain up to 63 characters—longer ones will be truncated if and when the [Buttons] section is re-written by FSUIPC.

<Action> is a single letter denoting the action being defined:

P Pulse the key press or control: i.e. do not hold the keys down whilst the button is held down. This is always the case for controls, and should always be the case for any key presses involving ALT key usage, because once the FS Menu is entered FSUIPC cannot supply further changes like key releases.

H Hold the specified keys down until the button is released. (This doesn't apply to Controls and will be treated like P in their case). Do *not* use this with key presses involving ALT, for the reason just given.

R Repeat the key press or control whilst the button is kept held down. The repeat rate is approximately 6 per second and is not adjustable. Do *not* use this with key presses involving ALT, for the reason already given.

U Pulse the key press or control when the button is released.

Any button can have a U entry as well as a P, H, or R entry. Provided the button only has one P, H or R, and/or one U entry, and that when it does have two they are either both key presses or both controls, then the button programming can be handled entirely in FSUIPC's Buttons dialog.

The <Joy#> identifies the joystick number (0–15 for normal joysticks, 16 upwards for PFC, GoFlight or other future 'emulated' joysticks) as displayed by FSUIPC. The <Btn#> identifies the specific button (0–39), again as in FSUIPC's display. Of these buttons 0–31 are regular buttons and 32–39 are the 8 possible POV view angles, starting forward and going clockwise every 45 degrees. (There are no emulated POVs so for joysticks 16 and upwards the buttons numbers are always in the 0–31 range).

Note that the Joystick numbers 0–15 may be replaced be an assigned letter (A–Z, omitting I and O) if the JoyLetters facility is being used (now the default) to assign joysticks indirectly, in case their real ID numbers change.

When buttons on WideFS clients are programmed, the Joystick number also includes a Client PC number—1000 for client 1, 2000 for client 2 and so on. The client numbering is actually handled by WideServer, which keeps a record of Client PC names and assigns them numbers in the WideServer.ini file. You only need to worry about that when changing PCs or renaming them.

For key presses, the <key> value following the letter 'K' is the virtual key code for the key to be pressed. Here's a list for convenience (but note that not all of these will be usable):

0	Null (+ Alt, Shift etc alone)	39	Right arrow
8	Backspace	40	Down arrow
12	NumPad 5 (*NumLock Off*)	44	PrintScreen
13	Enter	45	Insert
19	Pause	46	Delete
20	CapsLock	48	0 on main keyboard
27	Escape	49	1 on main keyboard
32	Space bar	50	2 on main keyboard
33	Page Up	51	3 on main keyboard
34	Page Down	52	4 on main keyboard
35	End	53	5 on main keyboard
36	Home	54	6 on main keyboard
37	Left arrow	55	7 on main keyboard
38	Up arrow	56	8 on main keyboard

57	9 on main keyboard		112	F1
65	A		113	F2
66	B		114	F3
67	C		115	F4
68	D		116	F5
69	E		117	F6
70	F		118	F7
71	G		119	F8
72	H		120	F9
73	I		121	F10
74	J		122	F11
75	K		123	F12
76	L		124	F13
77	M		125	F14
78	N		126	F15
79	O		127	F16
80	P		128	F17
81	Q		129	F18
82	R		130	F19
83	S		131	F20
84	T		132	F21
85	U		133	F22
86	V		134	F23
87	W		135	NumPad Enter (or F24?)
88	X		144	NumLock
89	Y		145	ScrollLock
90	Z		186	; : Key*
96	NumPad 0 (*NumLock ON*)		187	= + Key*
97	NumPad 1 (*NumLock ON*)		188	, < Key*
98	NumPad 2 (*NumLock ON*)		189	- _ Key*
99	NumPad 3 (*NumLock ON*)		190	. > Key*
100	NumPad 4 (*NumLock ON*)		191	/ ? Key*
101	NumPad 5 (*NumLock ON*)		192	' @ Key*
102	NumPad 6 (*NumLock ON*)		219	[{ Key*
103	NumPad 7 (*NumLock ON*)		220	\	Key*
104	NumPad 8 (*NumLock ON*)		221	] } Key*
105	NumPad 9 (*NumLock ON*)		222	# ~ Key*
106	NumPad *		223	` ¬ ¦ Key*
107	NumPad +		225	'AX' key
109	NumPad -		226	"<>" or "|"
110	NumPad .		227	Help key
111	NumPad /		228	00 key

* These keys will vary from keyboard to keyboard. The graphics indicated are those shown on my UK keyboard. It is possible that keys *in the same relative position* on the keyboard will respond similarly, so here is a positional description for those of you without UK keyboards. This list is in left-to-right, top down order, scanning the keyboard:

223	` ¬ ¦	is top left, just left of the main keyboard 1 key
189	- _	is also in the top row, just to the right of the 0 key
187	= +	is to the right of 189
219	[{	is in the 2nd row down, to the right of the alpha keys.
221	]}	is to the right of 219
186	; :	is in the 3rd row down, to the right of the alpha keys.
192	' @	is to the right of 186
222	# ~	is to the right of 192 (tucked in with the Enter key)
220	\		is in the 4th row down, to the left of all the alpha keys
188	, <	is also in the 4th row down, to the right of the alpha keys
190	. >	is to the right of 188
191	/ ?	is to the right of 190

The <shifts> value is a combination (add them) of the following values, as needed:

1	Left Shift
2	Left Control
4	Tab
8	Normal (add this in anyway)
16	Left Alt
32	Right Shift
64	Menu key (the application key, to the right of the right Windows key)
128	Right Control
256	Right Alt
512	Win (can only be used with the FSUIPC-added key press and release controls)

[*Note that the Tab and Left Alt keys are denoted by opposite bits here than when used for key programming. Apologies for this, which was a design oversight now too late to change*]

If only "normal" is needed, the whole parameter and the preceding comma can be omitted. Usual values are:

> 9 for lshift+ …
> 10 for lcontrol+ …
> 11 for lshift+lcontrol+ …

For FS controls the <control> is a number from 65536 upwards, denoting the specific FS control number. Lists of these can be found in my various FS controls documents. In the FSUIPC Buttons page the controls are shown by name normally, but if you want to try a control which has no name but *might* do something useful for you, enter it here, in the INI file. In the Buttons page FSUIPC will show this by number instead of name.

The <parameter> for a control is optional – just omit this along with the preceding comma for most toggle/button type controls. A parameter value of 0 will be assumed anyway.

Either or both of the <control> and <parameter> values can be provided in hexadecimal, preceded by an 'x' character.

As well as the FS controls, a number of additional FSUIPC controls are available. These range from 1000 to 3000, and also values 'xcc00zzzz' (in hexadecimal) which encode the FSUIPC "Offset" controls. See the list below the discussion on 'Keys' for full details.

SEQUENCES, COMBINATIONS, and MIXTURES

The Buttons page in the FSUIPC options is deliberately kept rather simple, hiding some of the programming possibilities. By editing the INI file you can do more:

- Hold one key down whilst pressing another

- Press and release a sequence of keys

- Mix key presses and FS controls in one button operation

- Make button actions conditional on the state of other buttons (see 'Compound' buttons, below)

- Make button actions conditional on values in FSUIPC offsets (see 'Adding offset conditions', below)

The first three are simply done by defining the actions in separate entries, each referring to the same joystick/button number. I'd recommend you first use the Buttons page to get the initial action programmed (this making sure you have the right button number), then close FS and edit the entries already made in the INI file. The only important thing is to number the entries in sequence – preferably, but not necessarily, consecutively.

Examples:

> 16=H1,2,K69,8
> 17=H1,2,K49,8

Presses and holds the 'E' key then presses and holds the '1' key, so both are pressed together. They are both released (in the same order) when the button is released.

> 18=P1,3,K69,8
> 19=P1,3,K49,8
> 20=P1,3,K50,8
> 21=P1,3,K51,8
> 22=P1,3,K52,8

Presses and releases 'E', then '1', '2', '3', and '4' in rapid succession, selecting all Engines.

COMPOUND BUTTON CONDITIONS

Facilities are included to allow you to specify actions for one button which are dependent on the state of another button (or more likely, switch). This by using what I call "Compound" button programming—though it could equally be "Conditional" or "Co-operative". Anyhow, I use the letter C in the definitions, as follows:

> n=CP(+j2,b2)j,b, ...
> n=CU(+j2,b2)j,b, ...
> n=CP(–j2,b2)j,b, ...
> n=CU(–j2,b2)j,b, ...

Here the 'C' denotes compound button checking, whilst P = pulse on pressing, U = pulse on releasing, as before. You can also use CR in place of CP for a repeating action—the repeats continue whilst all the conditions are true. There is no facility for the Hold action with the compound facilities.

Inside the parentheses are details of the *secondary* button, which must be in a certain condition for the current button to operate:

> (+j2,b2) means that button b2 on joystick j2 must be pressed ("on") for the current button action (for j,b) to be obeyed.

> (–j2,b2) means that button b2 on joystick j2 must be released ("off") for the current button action (for j,b) to be obeyed.

The j,b, ... part is the usual button parameter, for the action of the "current" button which is button b on joystick j.

You can have one condition, as shown above, or two, or more (up to 16 in fact), like this:

> n=CP(+j2,b2)(+j3,b3)j,b, ...

where, now, *both* the parenthesised conditions must be met for the 'j,b' button action to result in the defined event.

The conditions can be made to apply *not* to the *current* state of a button, but to the state of a 'flag' that is set and cleared by a button (or even a keypress). For every possible "normal" button (16 joysticks x 32 buttons = 512 buttons) FSUIPC maintains a "Flag" (F). Each time any button is pressed (goes from off to on) FSUIPC toggles its flag. This makes the buttons flag a sort of "latching" switch. You can test it in any parenthesised condition by preceding the condition by F, thus:

> N=CP(F+j2,b2) …

This says the rest of this parameter is obeyed if the Flag associated with j2,b2 is set. A condition (F–j2,b2) tests for the Flag being clear. Note that the actual current state of the button j2,b2 is not relevant. All that matters is whether it last left its Flag set or clear.

Any of the conditions in a multiple-conditioned setting may be on Flags.

These Button Flags can also be set, cleared and toggled by three special FS controls, **Button Flag Set** (C1003), **Button Flag Clear** (C1004), and **Button Flag Toggle** (C1005). In all three cases the Joystick (0–15 *only*) and Button (0–31) referenced is given in the Parameter, by a value calculated as:

> 256 * J + B   (for example, Joystick 15, Button 31 would be 3871).

These three controls are listed in the FSUIPC options drop downs for assignment in both the Buttons and Keys pages, so you can program them there, or here in the INI file. With these themselves as controls resulting for conditional button actions, you can influence conditions for button actions in a whole multitude of ways.

One point to note: since you can use the keyboard or other compound button actions to set, clear or toggle the flags, the actual button for which the Flag is assigned *does not actually need to exist*!

Okay. Now what does this really mean? Some simpler examples will suffice here. I leave it to the more imaginative amongst you to come up with some really complex applications!

First, it means that you can assign multiple uses to any number of buttons by making them conditional on a number of others. For example, a 12-position latching rotary switch could be wired to operate buttons 1 to 12 on joystick 1. Then for any other button I can program 12 different actions. For example, button 0,3 could have twelve different actions assigned, like this:

> 1=CP(+1,1)0,3, ...
> 2=CP(+1,2)0,3, ...
> 3=CP(+1,3)0,3, ...

...
                12=CP(+1,12)0,3, ...

and so on. For example, you may have a set of assignments for ground operations, a set for take-off, a set for climb, a set for cruise, and so on.

Second, to economise sensibly on the use of buttons, where you really need a toggle you can make any button toggle between two actions by using a flag as a condition. For example, suppose your button is Joy 11, button 3, and a spare flag (a button on joysticks 0-15 not otherwise used) is 15, 2. Program your button with three lines in FSUIPC (the numbers on the left need to be sequential with whatever's there already, but I'll assume you have no others so will start with 1):

                1=P11,3,C1005,3842

This says execute Control 1005 whenever your button is pressed. Control 1005 is "Button Flag Toggle". The parameter '3842' identifies the Flag: 256 x joystick 15 + button 2. So, this flag will now alternate between being set and clear each time you press the button.

                2=CP(F+15,2)11,3, ...

This tells FSUIPC what to do if the button is pressed AND the flag is set. Replace the ... part by the Control number and parameter for one of the actions you need.

                3=CP(F−15,2)11,3, ...

Similarly, this tells FSUIPC what to do when the button is pressed and the flag is not set.

Third, you can now program those two-phase type rotary switches, the ones where turning the spindle one way gives pulses on two lines phase shifted one way, and turning the spindle the other way gives the opposite phase relationship.

Say the inputs from the rotary are on Joystick 1, Buttons 1 and 2. When B1 is ON and B2 goes from off to on, then the spindle has turned one way. When B1 in ON and B2 goes from on to off, the spindle has turned the other. That is the simplest example:

                1=CP(+1,1)1,2, ...   turn direction 1 action
                2=CU(+1,1)1,2, ...   turn direction 2 action

You can also have double speed action, operating on every off to on and on to off change of B2. Just add two more conditions:

                3=CP(−1,1)1,2, ...   turn direction 2 action (B2 goes off to on when B1 is off)
                4=CU(−1,1)1,2, ...   turn direction 1 action (B2 goes on to off when B1 is off).

Since the whole thing is completely symmetric (there is no reason why B1 should control B2, it could also be the other way around), you can actually program it to act on ALL edges of both buttons, by adding another 4 conditions:

                5=CP(+1,2)1,1, ...   turn direction 2 action (B1 goes off to on when B2 is on)
                6=CU(+1,2)1,1, ...   turn direction 1 action (B1 goes on to off when B2 is on)
                7=CP(−1,2)1,1, ...   turn direction 1 action (B1 goes off to on when B2 is off)
                8=CU(−1,2)1,1, ...   turn direction 2 action (B1 goes on to off when B2 is off)

So, you can effectively choose how many pulses you will get for a given turning rate. As you can see, you can get rates of 1x, 2x or 4x—even 3x if you do one part for only half the changes! Note that for reliability at higher speeds you may need to reduce the **PollInterval**.

By the way, it is with some of these rotary switches where the double condition facility can come in very useful. If you have a single rotary of this type with also a push button action available, you can program it to adjust both the units and fractions of, say, a radio receiver. Just use the Flag associated with the button action to choose between one pair of actions or another, thus, supposing 1,3 to be the button:

                1=CP(F+1,3)(+1,1)1,2, ...   increment fraction
                2=CU(F+1,3)(+1,1)1,2, ...   decrement fraction
                3=CP(F−1,3)(+1,1)1,2, ...   increment integer
                4=CU(F−1,3)(+1,1)1,2, ...   decrement integer

One last thing. Using several rotaries of this type (that is, with the two signals in different phase relationships to indicate direction of turning), if they are of the type that have both signals 'off' in the detent you can save button connections by making one of them (on each one) common. If you do this you can only turn one of them at a time, but this is probably a worthwhile restriction if you are getting short of button connections.

ADDING OFFSET CONDITIONS

As well as all the above (and below, for Keys) any or all entries in all Buttons and Keys sections of FSUIPC7.INI can each contain a single condition based on the value of bits, bytes, words or double words in the FSUIPC IPC interface. These values are addressed by an "offset" value in hexadecimal and include just about anything you can think of about what is happening in FS.

Just taking some examples, you can make conditions based on:

- Whether the aircraft is airborne or on the ground
- Whether the engines are running
- Whether one or more of specific lights are switched on or off
- Whether the gear is up or down
- Even whether there are valid radio signals for NAV1, NAV2, GS, ILS LOC

… and so on. The possibilities are endless!

To make good use of this you will need the Programmer's Guide, which lists all of the offsets. This document is in the FSUIPC SDK. You'll find a lot of data in there that you cannot make use of—the conditions here deal with bits or values in 8-bit bytes, 126-bit words and 32-bit "double words". You cannot make use of string values, tables ot floating point values.

You add an offset condition to any Key or Button parameter line in FSUIPC7.INI as follows:

        \<sequence number>=\<offset condition> \<usual parameter>

The space between the new condition and the normal parameter is essential.

A simple example will help. Take this button push parameter, designed to toggle the landing gear when the button is pushed:

        1=P1,0,C65570,0

By adding an offset condition we can stop this doing anything when the aircraft is on the ground:

        1=W0366=0 P1,0,C65570,0

The inserted part, "W0366=0" specify that the Word (16-bit or 2-byte value) at offset 0366 must be zero for this line to be obeyed. Offset 0366 contains 0 when the aircraft is airborne, 1 when it is on the ground.

The format of the condition is:

        \<size>\<offset>\<mask>\<condition>

where

\<size>	is B for Byte, W for Word or D for Double Word,
\<offset>	is the FSUIPC offset, an hexadecimal value between 0000 and FFFF,
\<mask>	is optional, and if given selects one of more bits: specify as &x where 'x' is the 8, 16 or 32-bit mask in hexadecimal. The value in the offset is "ANDed" with this mask before being used,
\<condition>	is one of:

        =value for equality

        !value for inequality

        \<value for less than

        >value for greater than

and the "value" here is *decimal* unless preceded by an x (or X) in which case it is *hexadecimal* like the offset and mask. FSUIPC will output hexadecimal where a mask is used, otherwise decimal. All values are treated as unsigned.

The optional mask facility is useful for testing specific bits, as in the case of the light switches in offset 0D0C or the radio reception details in offset 3300. For example, the offset condition:

        W3300&0040!0

is TRUE when the currently tuned NAV1 is for an ILS.

The <condition> part is optional too, defaulting to !0 when omitted, so this last example could be abbreviated to:

> W3300&0040

For Project Magenta users who sometimes use the default FS autopilot instead one very useful condition is simply:

> W0500

Offset 0500 is non-zero when PM's MCP is running, zero otherwise, so you can program buttons and keys to operate PM when it is running, but FS otherwise.

Finally, for clever switching you may want to consider using one button to adjust an FSUIPC offset value which then, via offset conditions, selects between a number of alternative button and/or key assignments. To assist in this, offsets 66C0 to 66FF are reserved purely for you to do with as you like. The offset cyclic increment/decrement controls allow, say, a byte value in offset 66C0 to cycle through a number of values, then each value selects particular actions for defined keys or buttons. The entries in Buttons or Keys might look like this:

> 31=P174,10,Cx510066C0,x00030001
> 32=B66C0=0 P117,6,C1030,0
> 33=B66C0=1 P117,6,C1034,0
> 34=B66C0=2 P117,6,C1038,0
> 35=B66C0=3 P117,6,C1042,0

Here the value in the Byte at offset 66C0 is cycled from 0–3, and back to 0, by button 174,10, and this value, in turn, selects what happens with button 117,6.

These are real examples related to programming of a Go-Flight GF45 unit for different frequency adjustments. Many fuller examples of all this will appear in the documentation for my GFdisplay program, due shortly. GFdisplay brings my support for GF devices to a completion with display handling to complement the button programming in FSUIPC.

FOR FURTHER STUDY AND BETTER EXAMPLES

Additional interesting and useful examples of button programming are provided in an Appendix to this current document. That Appendix was graciously contributed by an enthusiastic FSUIPC user, to whom I am most grateful.

ERRORS IN BUTTON PARAMETERS

When the [Buttons] sections are read (or re-read via the "Reload" button in the FSUIPC Buttons page), the lines are thoroughly checked. Any that are syntactically wrong are ignored. However, where a line is ignored, an error message is appended in the form:

… << ERROR n …

The error numbers possible here are listed below. You can then correct the line and press "Reload" again to re-check it. You don't have to erase the << ERROR … additions. If the line is now okay, that message will be erased for you. If it is still in error a new error number may appear.

The errors are:

1	Offset condition: no hexadecimal offset following the size (B, W or D)
2	Offset condition: the offset is too big (more than 4 hex digits)
3	Offset condition: the '&mask' part has no hexadecimal mask
4	Offset condition: the mask is too big (more than 8 hex digits)
5	Offset condition: condition not recognised (not =, !, <, > or space representing !0)
6	Offset condition: comparison value X for hex, not followed by hex value
7	Offset condition: comparison value X for hex, too big (more than 8 hex digits)
8	Offset condition: no decimal or hex value after =, !, < or >.
9	Button operation not specified as H, P, R, U or C
10	Conditional button operation, no P, R or U after the C
11	Too many (…) button conditions
12	Condition joystick number too big
13	Button number omitted in condition (the ,b in (j,b))
14	No matching) found for (condition
15	Button number cannot be > 31 in condition
16	Main button joystick number is too big
17	Main button number is greater than 39
18	Comma (,) missing after main button number
19	The C.r Kor M needed for Control, Key or Macro is missing
20	Unknown formatting, syntax unintelligible

Keyboard Programming

FSUIPC's options dialogue provides a page for programming key presses to assign specific single FS controls. Here we look at how this programming is encoded in the FSUIPC7.INI file, and how the programming can be extended to provide multiple controls for a single keystroke combination.

FORMAT OF KEY DEFINITIONS

The key programming is saved in sections in the INI file. For globally operative keys this is called [Keys]. For aircraft-specific buttons it is [Keys.<aircraft name>]. Up to 1024 separate entries defining key actions can be included in each section, normally numbered sequentially from 0, provided that the total of the definitions in the Global section and the largest aircraft-specific section is not greater than 1024.

As with the Button parameters, Key press entries are reloaded each time you change aircraft in Flight Sim, so you can make changes in the INI file and test them without reloading FS.

The <aircraft name> part of the section heading can be abbreviated (manually, by editing the INI file) so that it applies to more than one aircraft. FSUIPC will automatically select the first section with a sub-string match – there's no concept of "longest match" any more.

The format of each entry in the Keys section is as follows:

> n=key,shifts,control,parameter

for a key press action only, or

> n=key,shifts,control1,parameter1,control2,parameter2

for a key with press (1) and release (2) actions.

Here   n can run from 0 to 1023 (i.e. maximum 1024 different keystroke actions can be added),

    key   virtual keycode, as in the FS CFG file (see list above, in the section about Buttons).
        Note: If the key press automatic repeats are to be ignored, this code is preceded by the letter '**N**'.

shifts		
	8	normal
	+1	left shift
	+2	left control
	+4	left alt
	+16	tab (an added "shift" to give more combinations)
	+32	right shift
	+64	Menu key (the application key, to the right of the right Windows key)
	+128	right control
	+256	right alt

        [*Note that the Left Tab and Left Alt keys are denoted by opposite bits here than when used for button programming. Apologies for this, which was a design oversight now too late to change*]

    control   This can be a FS control number, or a special FSUIPC number for additional controls. It can be in decimal, or, preceded by 'x' in hexadecimal. The additional FSUIPC controls range from 1000 to 3000, and also values xcc00zzzz in hexadecimal which encode the FSUIPC "Offset" controls. See list below for full details.

        It can also be a preset name, preceded by the letter **P**, or the name of an Input Event, preceded by the letter **I**, i.e.
            P<PresetName> for presets, and
            I< Input Event name> for Input Events

        Alternatively, it can be a Macro reference, in which case it takes the form:

            M<file#>:<ref#>

        For example M3:4 would refer to macro file 3, macro number 4 in that file. Please see the section on macros for more details.

        It can also be a Lua plug-in reference:

            L<file#>:<action>

Where the File number refers to the [LuaFiles] list in the INI, and the action is one of these letters:

R=Run, K=Kill, S=Set, C=Clear, T=Toggle, D=Debug

parameter   value to go with control, for "SET" types and some special FSUIPC controls. This also is normally an integer in decimal, but can be in hexadecimal preceded by 'x'. For preset and input event assignments, it will be a floating point number (even if the actual parameter is used as an integer, e.g. a parameter of 1 will be shown as 1.000000 when used with presets or input events)

You can add comments following a semicolon (;) at the end of the line, and these will be retained. You can also insert lines containing only comments, but they need an <Entry number> too, otherwise they may not retain their relative position. Comments can contain up to 63 characters—longer ones will be truncated if and when the [Buttons] section is re-written by FSUIPC.

You can do all of this programming directly in the FSUIPC "Keys" page whilst in FS. In fact it is better to do it there, so you can test it out directly. Note that some of the listed FS controls either do not work, or do not do as you might suppose! And some seem to be mixed up—for instance the "Zoom Out" and "Zoom In" controls appear to be switched, even though the Fine variants of these are okay.

There are two reasons you may want to edit the details in the INI file. The first is to make a single button press operate more than one control. You can specify such actions here, merely by adding the appropriate parameter lines. The controls will be sent in the order of the parameter entries (i.e. the 'n' in "n= …"). You can view all these, and delete them, in the Keys page on-line, but you cannot edit any other than the first such assignment for that key press.

The second reason is to add FSUIPC offset conditions. The facilities for making Button presses conditional upon assorted FS internals all apply to Key programming too, and the format and other details are the same as for Buttons. Please refer to the section above entitled "adding Offset Conditions".

ERRORS IN KEY PARAMETERS

When the [Keys] sections are read (or re-read via the "Reload" button in the FSUIPC Keys page), the lines are thoroughly checked. Any that are syntactically wrong are ignored. However, where a line is ignored, an error message is appended in the form:

… << ERROR n …

The error numbers possible here are listed below. You can then correct the line and press "Reload" again to re-check it. You don't have to erase the << ERROR … additions. If the line is now okay, that message will be erased for you. If it is still in error a new error number may appear.

The errors are:

1	Offset condition: no hexadecimal offset following the size (B, W or D)
2	Offset condition: the offset is too big (more than 4 hex digits)
3	Offset condition: the '&mask' part has no hexadecimal mask
4	Offset condition: the mask is too big (more than 8 hex digits)
5	Offset condition: condition not recognised (not =, !, <, > or space representing !0)
6	Offset condition: comparison value X for hex, not followed by hex value
7	Offset condition: comparison value X for hex, too big (more than 8 hex digits)
8	Offset condition: no decimal or Xhex value after =, !, < or >.
20	Unknown formatting, syntax unintelligible
21	Virtual key number not in range 1–255
22	No comma (,) after key number
23	No comma (,) after shift code value
24	Bad control value

Additional "FS" Controls added by FSUIPC

All the true FS controls are represented by numbers above 65536. They are listed in the "MSFS Controls" document in your FSUIPC Documents folder. FSUIPC has augmented these with its own set, programmable for both Button and Keys, and these utilise lower numbers, currently in the 1000–3000 range. These are:

1001	PTT on (for Squawkbox 3, Roger Wilco or AVC Advanced Voice Client)
1002	PTT off (for Squawkbox 3, Roger Wilco or AVC Advanced Voice Client)
1003	Set button flag (param = 256*joy + btn or JjBb)
1004	Clear button flag (param = 256*joy + btn or JjBb)
1005	Toggle button flag (param = 256*joy + btn or JjBb)
1006	KeySend to WideClients (param = KeySend number, 1–255)
1007	Autobrake Set (param=0 for RTO, 1=off, 2-5 for 1,2,3,Max)
1008	– [was 'Traffic Density Set (param = 0–100 %)', currently not available]
1009	– [was 'Traffic Density Toggle (param = 0–100 %)', currently not available]
1010	Spoiler inc (by 512 or amount set in SpoilerIncrement= INI parameter
1011	Spoiler dec (by 512 or amount set in SpoilerIncrement= INI parameter
1012	--
1013	--
1014	--
1015	--
1016	Ap Alt Var Dec Fast (–1000)
1017	Ap Alt Var Inc Fast (+1000)
1018	Ap Mach Var Dec Fast (–.10)
1019	Ap Mach Var Inc Fast (+.10)
1020	Ap Spd Var Dec Fast (–10)
1021	Ap Spd Var Inc Fast (+10)
1022	Ap Vs Var Dec Fast (–1000)
1023	Ap Vs Var Inc Fast (+1000)
1024	Heading Bug Dec Fast (–10)
1025	Heading Bug Inc Fast (+10)
1026	Vor1 Obi Dec Fast (–10)
1027	Vor1 Obi Inc Fast (+10)
1028	Vor2 Obi Dec Fast (–10)
1029	Vor2 Obi Inc Fast (+10)
1030	Com1 use whole inc
1031	Com1 use whole dec
1032	Com1 use frac inc
1033	Com1 use frac dec
1034	Com2 use whole inc
1035	Com2 use whole dec
1036	Com2 use frac inc
1037	Com2 use frac dec
1038	Nav1 use whole inc
1039	Nav1 use whole dec
1040	Nav1 use frac inc
1041	Nav1 use frac dec
1042	Nav2 use whole inc
1043	Nav2 use whole dec
1044	Nav2 use frac inc
1045	Nav2 use frac dec
1046	Adf1 use whole inc
1047	Adf1 use whole dec
1048	Adf1 use frac inc
1049	Adf1 use frac dec
1050	Adf2 use whole inc
1051	Adf2 use whole dec
1052	Adf2 use frac inc
1053	Adf2 use frac dec
1054	Xpndr low NN inc
1055	Xpndr low NN dec
1056	Xpndr high NN inc
1057	Xpndr high NN dec
1058	Freeze pos on
1059	Freeze pos off
1060	Freeze pos toggle

1061	Engine 1 Autostart
1062	Engine 2 Autostart
1063	Engine 3 Autostart
1064	Engine 4 Autostart
1065	Throttles off
1066	Throttles on
1067	Throttles toggle
1068	PVT voice transmit on (for Squawkbox 3.0.4 or later)
1069	PVT voice transmit off (for Squawkbox 3.0.4 or later)
1070	Key Press and Release (param is Keycode + Shift code (i.e. enter with + symbol, not calculate), or JsBk)
1071	Key Press/Hold (param is Keycode + Shift code(i.e. enter with + symbol, not calculate), or JsBk)
1072	Key Release (param is Keycode + Shift code (i.e. enter with + symbol, not calculate), or JsBk)
1073	FSUIPC display window toggle
1074	Airline traffic density set
1075	GA traffic density set
1076	Shipping traffic density set
1077	Cloud cover density set
1078	Simple/complex clouds set
1079	--
1080	Wheel trim toggle (for mousewheel trim adjusting) (?)
1081	Wheel trim faster (?)
1082	Wheel trim slower (?)
1083	Wheel trim speed toggle (?)
1084	Lua Kill All
1085	--
1086	FollowMe please (i.e. request) (needs FollowMe 2)
1087	FollowMe cancel (needs FollowMe 2)
1088	FollowMe continue (needs FollowMe 2)
1089	--
1090	--
1091	--
1092	Re-SimConnect (re-initialises SimConnect interface)
1093	Efis ND scale inc (default B738 and A321)
1094	Efis ND scale dec (default B738 and A321)
1095	Efis ND mode inc (default B738 and A321)
1096	Efis ND mode dec (default B738 and A321)
1097	Efis ND map item inc (default B738 and A321)
1098	Efis ND map item dec (default B738 and A321)
1099	Efis VORADF1 inc (default B738 and A321)
1100	Efis VORADF1 dec (default B738 and A321)
1101	Efis VORADF2 inc (default B738 and A321)
1102	Efis VORADF2 dec (default B738 and A321)
1103	Efis 738 ND centre (default B738)
1104	Efis 738 ND arc (default B738)
1105	Efis A321 InHg/hPA toggle (default A321)
1106	Efis A321 ILS mode toggle (default A321)
1107	AP alt change rate toggle (default A321)
1108	Efis ND scale set (parameter 0–7 for 738, 0–5 for A321) (default B738 and A321)
1109	Efis ND mode set (parameter 0–2 for 738, 0–3 for A321) (default B738 and A321)
1110	Efis ND map item set (parameter 0–3) (default B738 and A321)
1111	Efis VORADF1 set (parameter 0–2) (default B738 and A321)
1112	Efis VORADF2 set (parameter 0–2) (default B738 and A321)
1113	Efis A321 InHg/hPA set (parameter 0–1) (A321)
1114	List local panel variables (“L:vars”) in log when change aircraft
1115	IYP Listen On
1116	IYP Listen Off
1117	IYP ComeFly Active
1118	IYP ComeFly Inactive
1119	Xpndr stby (sb3)
1120	Xpndr on/mode c (sb3)
1121	Xpndr toddle (sb3)
1122	Xpndr ident (sb3)
1123	--
1124	Com1/2 tx switch
1125	Key focus restore
1126	Nothing: no action
1127	--
1128	--

1129	--
1130	--
1131-1138	Log option controls - see below
1139	--
1140	Throttle Sync off
1141	Throttle Sync on
1142	Throttle Sync toggle
1143	--
1144	RemoteTextMenuToggle
1145	--
1146	--
1147	ResetTCAS: restarts the AI Traffic Scanning
1148	Traffic freeze toggle (*not working?*)
1149	Traffic freeze on (*not working?*)
1150	Traffic freeze off (*not working?*)
	These three operate on AI ground traffic in "taxi out" mode.
1151	Auto-Save enable/disable toggle. An additional control to toggle AutoSave on/off. This is only active once the AutoSave functionality has been activated.
1152	Pause (ms) – this adds a delay in ms. It can be used to insert a delay between key press/releases and mouse operations (where needed)
1153	Set MaxSteerSpeed
1154	Trigger Auto-save
1155	Reload WASM
1156	Key Focus FSUIPC

The following controls manipulate Logging settings without visiting the Logging tab of the options dialogue:

1131	Log set (parameter sets additional logging options*)
1132	Log clear (parameter clears specified logging options*)
1133	Log toggle (parameter specifies options to be changed*)

* The bits in the parameter specify the options being set/cleared/toggled.
(These are in the same order as the list in the Logging tab):

2^0	1	Weather
2^1	2	IPC Writes
2^2	4	IPC Reads
2^3	8	Buttons and Keys
2^4	16	Events (non-axis)
2^5	32	Axis events
2^6	64	Lua logging to separate files
2^7	128	Debug/Trace Lua
2^8	256	Log Extras

To clear all logging options, use "Log clear" with parameter 511.

1134	Log console on *(Note: from 4.859r the focus is restored to FS)*
1135	Log console off
1136	Log debug (parameter gives debug instructions, *for use only under Support direction*)
1137	New log file (close last and start a new one)
1138	Log test options (parameter gives test instructions, *for use only under Support direction*)

Note that changes made by these controls are NOT saved to the INI file *unless* the Logging tab is visited whilst the changes are in use.

FSUIPC autopilot controls:

1930	FSUIPC bank hold off
1931	FSUIPC bank hold on
1932	FSUIPC bank hold set
1933	FSUIPC bank hold toggle
1934	FSUIPC mach hold off
1935	FSUIPC mach hold on
1936	FSUIPC mach hold set
1937	FSUIPC mach hold toggle
1938	FSUIPC pitch hold off
1939	FSUIPC pitch hold on
1940	FSUIPC pitch hold set
1941	FSUIPC pitch hold toggle
1942	FSUIPC speed hold off
1943	FSUIPC speed hold on
1944	FSUIPC speed hold set
1945	FSUIPC speed hold toggle

Project Magenta controls:

2010	PM MCP SPD push on B747
2011	PM MCP HDG sel on B747
2012	PM MCP ALT push on B747
2013	–
2014	–
2015	–
2016	--
2017	PM MCP FD2 off
2018	PM MCP FD2 on
2019	PM MCP A/T on
2020	PM MCP A/T off
2021	PM MCP THR mode button
2022	PM MCP SPD mode button
2023	PM MCP Mach/IAS sel
2024	PM MCP FLCH mode button
2025	PM MCP HDG mode button
2026	PM MCP VNAV mode button
2027	PM MCP LNAV mode button
2028	PM MCP LOC mode button
2029	PM MCP APP mode button
2030	PM MCP ALT mode button
2031	PM MCP VS mode button
2032	PM MCP AP1 (L) button
2033	PM MCP AP2 (C) button
2034	–
2035	--
2036	PM MCP AP3 (R) button
2037	PM MCP FD1 off
2038	PM MCP FD1 on
2039	--
2040	PM MCP AP Disc (not 747)
2041	PM MCP AP Eng (not 747)
2042	PM MCP AP Disc (747 only)
2043	–
2044	–
2045	–
2046	–
2047	–
2048	--
2049	PM AB LS button
2050	PM AB STD QNH rel (push)
2051	PM AB STD QNH set (pull)
2052	PM AB SPD button push
2053	PM AB SPD button pull
2054	PM AB HDG button push
2055	PM AB HDG button pull
2056	PM AB ALT button push

2057	PM AB ALT button pull
2058	PM AB VS button push
2059	PM AB VS button pull
2060	PM AB EXPED button
2061	PM AB TRKFPA button
2062	–
2063	--
2064	PM PFD Decision Ht Dec
2065	PM PFD Decision Ht Inc
2066	PM MCP Hdg Dec 1
2067	PM MCP Hdg Inc 1
2068	PM MCP Hdg Dec 10
2069	PM MCP Hdg Inc 10
2070	PM MCP Alt Dec 100
2071	PM MCP Alt Inc 100
2072	PM MCP Alt Dec 1000
2073	PM MCP Alt Inc 1000
2074	PM MCP Spd Dec 1/.01
2075	PM MCP Spd Inc 1/.01
2076	PM MCP Spd Dec 10/.10
2077	PM MCP Spd Inc 10/.10
2078	PM MCP V/S Dec 100
2079	PM MCP V/S Inc 100
2080	PM MCP Crs Dec 1
2081	PM MCP Crs Inc 1
2082	PM QNH Dec 0.01/1
2083	PM QNH Inc 0.01/1
2084	PM ND Range Dec
2085	PM ND Range Inc
2086	PM ND Mode Dec
2087	PM ND Mode Inc
2088	PM ND2 Range Dec
2089	PM ND2 Range Inc
2090	PM ND2 Mode Dec
2091	PM ND2 Mode Inc
2092	–
2093	–
2094	–
2095	--
2096	PM AB ND ILS Mode
2097	PM ND Map Arc Mode
2098	PM ND Map Ctr Mode
2099	PM ND Rose Mode
2100	PM ND Map Plan Mode
2101	PM ND Range 10
2102	PM ND Range 20
2103	PM ND Range 40
2104	PM ND Range 80
2105	PM ND Range 160
2106	PM ND Range 320
2107	PM ND Range 640
2108	PM ND VOR display
2109	PM ND NDB display
2110	PM ND WPT display
2111	PM ND ARPT display
2112	PM ND DATA display
2113	PM ND POS display
2114	PM AB ND VOR1 on
2115	PM AB ND ADF1 on
2116	PM AB ND VORADF1 off
2117	PM AB ND VOR2 on
2118	PM AB ND ADF2 on
2119	PM AB ND VORADF2 off
2120	PM AB ND Metric
2121	PM AB ND HDGVS/TRKFPA
2122	PM AB THR TOGA
2123	PM AB THR FLX/MCT
2124	PM AB THR CLB

2125	PM AB THR IDLE
2126	PM AB THR REV IDLE
2127	PM AB THR MAX REV
2128	PM AB ND2 ILS Mode
2129	PM ND2 Map Arc Mode
2130	PM ND2 Map Ctr Mode
2131	PM ND2 Rose Mode
2132	PM ND2 Map Plan Mode
2133	PM ND2 Range 10
2134	PM ND2 Range 20
2135	PM ND2 Range 40
2136	PM ND2 Range 80
2137	PM ND2 Range 160
2138	PM ND2 Range 320
2139	PM ND2 Range 640
2140	PM ND2 VOR display
2141	PM ND2 NDB display
2142	PM ND2 WPT display
2143	PM ND2 ARPT display
2144	PM ND2 DATA display
2145	PM ND2 POS display
2146	PM AB ND2 VOR1 on
2147	PM AB ND2 ADF1 on
2148	PM AB ND2 VORADF1 off
2149	PM AB ND2 VOR2 on
2150	PM AB ND2 ADF2 on
2151	PM AB ND2 VORADF2 off
2152	PM AB ND2 Metric
2153	PM AB ND2 HDGVS/TRKFPA
2154	–
2155	–
2156	–
2157	–
2158	–
2159	--
2160	PM EICAS Show Controls
2161	PM EICAS Standby Gauge
2162	PM EICAS Page Dec
2163	PM EICAS Page Inc
2164	PM EICAS Synoptic Dec
2165	PM EICAS Synoptic Inc
2166	PM AB ND ILS Mode
2167	PM ND Plan Wpt Dec
2168	PM ND Plan Wpt Inc
2950	PM Elec All Toggle
2951	PM Elec PFD Toggle
2952	PM Elec ND Toggle
2953	PM Elec EICAS Toggle
2955	PM Elec PFD2 Toggle
2956	PM Elec ND2 Toggle
2958	PM Elec Stdby Toggle
2966	PM Elec All ON
2967	PM Elec PFD ON
2968	PM Elec ND ON
2969	PM Elec EICAS ON
2971	PM Elec PFD2 ON
2972	PM Elec ND2 ON
2974	PM Elec Stdby ON
2982	PM Elec All OFF
2983	PM Elec PFD OFF
2984	PM Elec ND OFF
2985	PM Elec EICAS OFF
2987	PM Elec PFD2 OFF
2988	PM Elec ND2 OFF
2990	PM Elec Stdby OFF"
2994	PM Whazzup keys (by Param), see PM offsets list, 542E
2995	PM Quickmap keys (by Param), see PM offsets list, 542C
2996	PM GC keys (by Param), see PM offsets list, 542A

| 2997 | PM CDU keys (by Param), see PM offsets list, 5428 |

Note: all the "Keys" inputs to PM modules provide efficient ways of directing specific keypresses to them, wherever they may be on the Network. The parameter in these is the keystroke code (see the list earlier in this document) , plus specific PM-defined values for shifts, thus:

256 for Shift, 512 for Ctrl, 1024 for Alt.

You don't need to worry about changing other bits when two codes are the same—FSUIPC takes are of that automatically.

| 2998 | PM MCP Kcodes (by Param), see Pm offsets list, 04F2 |

This way of controlling the PM MCP may offer some features not found elsewhere. The parameter is the number used in the Elan Informatique "Knnn" codes normally sent to the MCP via a serial connection. Please refer to the PM offsets document for further information.

| 2999 | Project Magenta GC Controls. Param specifies action, (see the list in the Project Magenta "Offsets" publication) |

FSUIPC Offset Controls:

x0100zzzz	Offset Byte Set (offset = zzzz), hexadecimal
x0200zzzz	Offset Word Set (offset = zzzz), hexadecimal
x0300zzzz	Offset Dword Set (offset = zzzz), hexadecimal
x0500zzzz	Offset Byte Setbits (offset = zzzz), hexadecimal
x0600zzzz	Offset Word Setbits (offset = zzzz), hexadecimal
x0700zzzz	Offset Dword Setbits (offset = zzzz), hexadecimal
x0900zzzz	Offset Byte Clrbits (offset = zzzz), hexadecimal
x0A00zzzz	Offset Word Clrbits (offset = zzzz), hexadecimal
x0B00zzzz	Offset Dword Clrbits (offset = zzzz), hexadecimal
x0D00zzzz	Offset Byte Togglebits (offset = zzzz), hexadecimal
x0E00zzzz	Offset Word Togglebits (offset = zzzz), hexadecimal
x0F00zzzz	Offset Dword Togglebits (offset = zzzz), hexadecimal
x1100zzzz	Offset UByte Increment (offset = zzzz), hexadecimal *
x1200zzzz	Offset UWord Increment (offset = zzzz), hexadecimal *
x1300zzzz	Offset UDword Increment (offset = zzzz), hexadecimal *
x2100zzzz	Offset UByte Decrement (offset = zzzz), hexadecimal *
x2200zzzz	Offset UWord Decrement (offset = zzzz), hexadecimal *
x2300zzzz	Offset UDword Decrement (offset = zzzz), hexadecimal *
x3100zzzz	Offset SByte Increment (offset = zzzz), hexadecimal *
x3200zzzz	Offset SWord Increment (offset = zzzz), hexadecimal *
x3300zzzz	Offset SDword Increment (offset = zzzz), hexadecimal *
x4100zzzz	Offset SByte Decrement (offset = zzzz), hexadecimal *
x4200zzzz	Offset SWord Decrement (offset = zzzz), hexadecimal *
x4300zzzz	Offset SDword Decrement (offset = zzzz), hexadecimal *
x5100zzzz	Offset Byte Cyclic Increment (offset = zzzz), hexadecimal *
x5200zzzz	Offset Word Cyclic Increment (offset = zzzz), hexadecimal *
x5300zzzz	Offset Dword Cyclic Increment (offset = zzzz), hexadecimal *
x6100zzzz	Offset Byte Cyclic Decrement (offset = zzzz), hexadecimal *
x6200zzzz	Offset Word Cyclic Decrement (offset = zzzz), hexadecimal *
x6300zzzz	Offset Dword Cyclic Decrement (offset = zzzz), hexadecimal *
x7000zzzz	Offset Float32 Set/1000 (offset = zzzz): the parameter is divided by 1000
x7400zzzz	Offset Float64 Set/1000 (offset = zzzz): the parameter is divided by 1000
x7800zzzz	Offset Float32 Inc/1000 (offset = zzzz): the parameter is divided by 1000
x7C00zzzz	Offset Float64 Inc/1000 (offset = zzzz): the parameter is divided by 1000
	(For "decrements" use a negative parameter in the increment controls)

* The fixed point increment/decrement values operate on Unsigned (U) or Signed (S) values, and have a parameter with the unsigned or signed limit in the upper 16 bits and the increment/decrement amount (always unsigned) in the lower 16 bits. This applies even to the Dword (i.e. 32 bit) inc/dec controls.

FSUIPC directly assigned axis controls

All of the "direct to FSUIPC calibration" axes are effectively added FSUIPC controls and can be sent using these values:

64101	Aileron	64112	Throttle4	64115	Mixture3
64102	Elevator	64113	Mixture1	64116	Mixture4
64103	Rudder	64114	Mixture2	64117	PropPitch1
64104	Throttle			64118	PropPitch2
64105	PropPitch			64119	PropPitch3
64106	Mixture			64120	PropPitch4
64107	LeftBrake			64121	ElevatorTrim
64108	RightBrake			64122	Spoilers
64109	Throttle1			64123	Flaps
64110	Throttle2			64125	Reverser
64111	Throttle3				

64127	Aileron Trim	64131	Cowlflaps3	64138	SlewSide
64128	Rudder Trim	64132	Cowlflaps4	64139	SlewAhead
64129	CowlFlaps1	64133	PanHeading	64140	SlewHeading
64130	Cowlflaps2	64134	PanPitch	64141	Reverser1
		64135	PanTilt	64142	Reverser2
		64136	SteeringTiller	64143	Reverser3
		64137	SlewAlt	64144	Reverser4

Adding Simulator variables (simvars) to FSUIPC offsets

You can add any simulator A-type variable (called simvars) to an FSUIPC offset free for general use (i.e. such as those starting at offset 0x2544, 0x2644, 0x2744, 0x66C0 or 0xA000). For a complete list of available simvars, please consult the MSFS documentation at:

   https://docs.flightsimulator.com/html/Programming_Tools/SimVars/Simulation_Variables.htm

Simvars are added to offsets from a file called **myOffsets.txt** which must be located in your FSUIPC7 installation folder. Each line of the file must be either a comment, starting with '//', or an offset entry of the form:

   offset, *size*, *simvar*, *type*, *units* [, *w*]

where

> *offset* is the FSUIPC offset that will hold the simvar value
> *size* is the size of the the data held at the offset
> *simvar* is the name of the simulator variable
> *type* is the simvars data type as defined by MSFS, and must be one of the following string identifiers:
>> I32 : for 32-bit integers
>> I64 : for 64-bit integers
>> F32 : for 32-bit floating-point numbers
>> F64 : for 64-bit floating-point numbers
>> S8, S32, S64, S128, S256, S260 : for string data types, with the number indicating the max length
>> LLA : 3*32-bit floating-point numbers (Latitude/Longitude/Altitude)
>> XYZ : 3*64-bit floating-point numbers (values representing X,Y,Z positions)
> *units* are the units of the simvar. This must be as defined by MSFS or a *compatible* unit type
>> (see the MSFS documentation for the list of compatible unit types)*
> *w* : indicates that the simvar is also writable, i.e. writing to the offset will attempt to update the simvar. Please check the MSFS documentation to confirm the simvar is writable before using this option.

* Please note that the *units* for all string data types is **Number** and not **String**.

Note that the size can be smaller than the type an an appropriate conversion will be performed, e.g. an I32/Bool or I32/Enum can fit in 1 byte - you don't need 4 bytes (32 bits).
Note also that the offset needs to be bound to the size. This means that if the size is 8, the last offset digit needs to be 0 or 8, if the size is 4, the last offset digit needs to be 0, 4, 8 or C, etc (but not for string data types).

Here is an example of the contents of a **myOffsets.txt** file:

```
// offset, size, simvar, type, units [, w]
0x66C0, 1, ELECTRICAL MASTER BATTERY:2,I32, Bool, w
0x66C1, 1, ATC HEAVY, I32, Bool, w
0x66C2, 2, COM VOLUME, I32, Percent
0x66C4, 1, COM SPACING MODE, I32, Enum
0x66C6, 2, TRANSPONDER CODE:1, I32, BCO16
0x66C8, 2, TRANSPONDER STATE, I32, Enum, w
0xA008, 4, COM ACTIVE FREQ TYPE:1, S8, Number
```

The **myOffsets.txt** file is only read at FSUIPC7 start-up, so to load any changes to this file you must restart FSUIPC7 (no need to restart MSFS).

It is suggested that you open the FSUIPC7 logging console (**Log → Open Console** menu item) when starting to use this facility or when making any changes. The log should tell you if the entries in this file are valid or not, and you will also see errors from SimConnect if the variable does not exist (be aware that not all of the simulator variables defined in the MSFS documentation actually exist!) or if there is a problem with the units or type.

Macro Controls

FSUIPC will read any file in its installation folder which has file type "mcro". Such files contain definitions of additional controls to be listed and assignable in FSUIPC's Keys, Buttons and Axis Assignments dialogues. All macro files are also re-read and re-installed whenever the Reload button in any of those three dialogues are used.

It is important that the file name (xxxx.mcro) is limited to 16 characters maximum, (plus the ".mcro"). This will be used as part of the name of the added controls in the drop-downs. Best to keep the names short and to the point—probably the name of the program or program function for which the controls are being added.

Inside a macro file there should be just one section called [Macros]. This must contain definitions of numbered controls, with names also up to 16 characters. These names only have to be unique in that file.

Here is an example, here for a possible Project Magenta glass cockpit ND Mode switch:

```
[Macros]
1=MAP Capt=C2999,1
2=NAV Capt=C2999,2
3=VOR Capt=C2999,3
4=PLN Capt=C2999,4
5=APP Capt=C2999,5
6=CTR Capt=C2999,6
101=MAP F/O=C2999,101
102=NAV F/O=C2999,102
103=VOR F/O=C2999,103
104=PLN F/O=C2999,104
105=APP F/O=C2999,105
106=CTR F/O=C2999,106
```

Note that the numbers on the left do not have to be contiguous, but must be in the range 1–999 inclusive. These will be used internally, and in the FSUIPC7.INI file, to identify the control within the file.

Supposing the example above occurred in a file called 'PM GC.mcro'. The names which would then appear, in proper alpha sequence in the FSUIPC drop-downs, would be:

```
PM GC: APP Capt
PM GC: APP F/O
PM GC: CTR Capt
PM GC: CTR F/O
PM GC: MAP Capt
PM GC: MAP F/O
PM GC: PLN Capt
PM GC: PLN F/O
PM GC: VOR Capt
PM GC: VOR F/O
```

The value assigned to each control is either another control (*any* FS or FSUIPC-added control, including offset controls and even macro controls—see later), or a Key press. i.e:

Either: Cn,p   (control number, parameter, optionally in hex with a preceding x)

Or: Kk,s   (keycode and shifts).

Both of these are exactly as already defined for Button controls—see the earlier section on Button programming.

Macro Control References

Macro controls are represented internally in the same sort of way as FSUIPC offsets controls, by using high-value bits in the control number. However, the representation in Macro files and in the INI file is as follows:

Mm:n

where m if the Macro File number (see below) and n is the control number from the file, as described above.

Macro file numbers are assigned by FSUIPC when it loads the file. These are remembered in the INI file in a new section [MacroFiles]. For example, in the above case you might get:

[MacroFiles]

                    1=PM GC

making "PM GC.mcro" file number 1 for all reference purposes.

It is important to note that different users will have a different selection of macro files in different orders. If they wish to exchange Button assignments they will need to re-assign all macro controls after making their [MacroFiles] sections the same, or at least the same for those files they have in common.

Multiple actions in one macro control

A macro control is not limited to having only one resulting action. If more than one action is required several lines are used in the definition, as follows:

                    n=<name>
                    n.1=action1
                    n.2=action2
                    etc.

For an example consider a 'Menu.mcro' file containing these definitions:

                    [Macros]
                    1=Display
                    1.1=K79,12 ;Tab O
                    1.2=K69,8 ;E
                    1.3=K68,8 ;D
                    2=FSUIPC
                    2.1=K68,12 ;Tab D
                    2.2=K70,8 ;F

This adds two controls, 'Menu: Display' and 'Menu: FSUIPC'. The first uses Tab+O E D keystrokes to call up the FS display settings dialogue, the second uses Tab+D F to call up the FSUIPC options.

Note that there's a limit of 2000 numbered parameters in total in the macro file—so, for instance, 999 macro numbers (1–999, the maximum) with an average of two actions each would be just two shy of the limit. Large files aren't good in any case as the drop-down list will be full of the added controls all beginning with the same filename. Best to split into functional groups with meaningful filenames, to make the controls easier to locate.

Parameter passing

Normally, and certainly in all the above examples, any parameter set for a Macro Control, when assigned in the Buttons or Keys dialogues, would be discarded as not relevant. However, there is a facility to allow it to be used.

If the parameter part of *any* of the controls defined in the macro is omitted, the parameter value from the *calling* macro is substituted.

As a rather silly example, if you wanted a general PM GC control but not the one named already in FSUIPC, you could define it as

                    7=by param=C2999

This would appear in the drop-downs as 'PM GC:by param', and the parameter assigned by the user would be used in the C2999 operation. Note that in multiple-line definitions, the same parameter value substitutes for every omitted parameter value.

One interesting consequence of this is the possibility of defining axis controls. To make another silly example, if I define a macro like this:

                    8=Flaps=C66534   ;FS control 66534 is Axis Flaps Set

and then assign it to an axis in the Axis assignments dropdown, the axis I've assigned will operate exactly as the Axis Flaps Set axis.

This may not seem so futile when you realise that you can have multiple line mixtures of controls and key presses also produced by the same Macro. I'm sure there would be wealth of ideas for using this 'feature' (which actually fell out of the implementation by accident rather than by design!).

Mouse macros

Mouse macros are currently not available in FSUIPC7, pending facilities to be implemented in the SDK.

Gauge local variable access (L:vars), by macro

Local named panel variables ("L:<name>"), which I'll refer to as "**Lvars**", are now accessible using FSUIPC, but only if you have opted to install the FSUIPC WASM module during installation. This WASM module provides FSUIPC with access to both lvars and hvars. The WASM module also needs to be enabled – you can do this using the **Add-ons → WASM → Enable** menu option. You only need to do this once.

With the WASM module installed and enabled, lvars can now be listed in the Log, added to free user offsets for reading and writing, written to via Macros, and manipulated with both reads and writes through extensions to the ipc Lua Library. Hvars can be listed and activated.

The log listing is obtained for the currently loaded aircraft panels by a new assignable control in the drop-down lists called "**List local panel variables**". You can also use the list functions in the **WASM** menu (under the **Add-ons** menu) to list available lvars, list available hvars, as well as set lvars and activate hvars.

Note that all FSUIPC can do is list what it finds. Whether the values are of any use or not is questionable—they are internal to the gauge and how they are used, manipulated, and so on, will vary enormously. By all means try things if you wish, but don't assume the solution is there waiting for you. Also, some lvars are classified as read-only and cannot be updated, although FSUIPC cannot distinguish between a read-only and a writeable lvar. You will just have to test to see if the lvar can be updated.

Please see the FSUIPC WASM section later in this manual for further details on using the FSUIPC WASM module for lvar/hvar access.

Macros to change Lvars

The macro facility to operate **Lvars** can only be used by editing macro files and building them manually. The format is:
> N=L:name=ACTION

Where ACTION must be one of: **Set, Inc, Dec, Sbits, Cbits, Cyclic** or **Toggle** (but only the first 3 letters are needed):

Set	copies the parameter in the Macro invocation to the identified Lvar. Alternatively, a value can be given explicitly here, by "**Set,n**". Values are limited to the normal parameter range, –32768 to 32767.
Inc	increments the value, and here the parameter (explicit or supplied) gives the upper limit, which can be equalled but not exceeded.
Dec	decrements the value, with the parameter setting the lower limit.
Sbits	sets bits into the value, assuming it is a 32-bit integer, according to the parameter (i.e OR's the parameter into the value).
Cbits	clears bits from the value, assuming it is a 32-bit integer, according to the parameter (i.e AND's the value with the inverse of the parameter).
Cyclic	is the same as **Inc**, but after the limit is reached the next value is 0.
Toggle	changes the value to zero if it is non-zero, or 1 if it is zero.

The Lvar name has a limit of 56 characters.

The multi-line macro format can still be used with the Lvar macros, as follows:

> N=L:name
> N.1=action1
> N.2=action2
> ... etc.

However, unlike the usual multi-line macros, those using an L:Var cannot be mixed with any other parameter types or other L:Vars. The single L:Var identifier is the actual macro name as well, so this prevents such complexity.

Macros to activate Hvars

The macro facility to activate H**vars** can only be used by editing macro files and building them manually. The format is:

> N=H:name=Set

Note that, unlike Lvars, Hvars have no associated value, and the only action that can be performed on the m is to activate them, which is achieved using the Set command. No parameter is needed or used.

Lua access to Lvars and Hvars

The Lua facilities are **ipc.readLvar**, **ipc.readLvarSTR**, **ipc.writeLvar**, **ipc.writeLvarSTR**, **ipc.getLvarName**, **ipc.getLvarId**, **ipc.activateHvar** and **ipc.reloadWASM**. These are all described in the updated Lua library documentation, and a sample Lua plug-in is provided demonstrating their use.

Add-on Custom Events

FSUIPC includes facilities for assigning buttons and keys to MSFS add-on custom events. These are named events (FS controls) implemented in add-ons via SimConnect facilities. Custom event names always contain a period (.) and this distinguishes them from MSFS internal events/controls.

The events to be made assignable are listed in '.evt' files placed in the FSUIPC installation folder. Up to 128 such files will be recognised, each one containing no more than 256 entries defining a custom event name. The format of each file is

> [Events]
> 0=name.of.event1
> 1= ...
> etc.

Numbering can be 0-255 or 1-255, but the first omission terminates the list as far as FSUIPC is concerned.

The assignments can then be made in the normal drop-down lists in FSUIPC.

Event files for MobiFlight events are included (if you opted to install them during the installation process) in a sub-folder of your FSUIPC7 installation folder called *EventFiles*. To use any of these event files, simply copy (or move) them from the *EventFiles* sub-folder to the main FSIUPC7 installation folder (i.e. up one level). Note that you must have the MobiFlight WASM module installed to use these events.

To determine the actual control number of events defined in event files (e.g. for use with the lua **ipc.control** function) you can use the following formula:

> control number = 32768 + (**event file index**)*256 + **event index number**

where **event file index** is the index number of the event file (in your FSUIPC7.ini, under [EventFiles]) and **event index number** is the index number if the event in the event file. So, the first event in the first event file will have a control number of 32768, the second event 32769, first event in the second file as 33024. etc

Automatic running of Macros and Lua plugins

By some editing in the INI file, you can arrange for one or more Macros or Lua plugins to be executed, in order, automatically whenever the current aircraft is changed (or, indeed, first loaded), or a specific named aircraft (or Profile) is loaded.

This allows switches, offsets, and other things to be set specifically for an aircraft (or aircraft type, for Profiles) when it is first loaded.

This is done by adding new sections to the INI file with the title{

> **[Auto]**

> or

> **[Auto.xxxx...]**

where the **xxxx** part is the profile name when profiles are being used.

These Auto sections thus parallel the Keys and Buttons sections -- the naming and selection follows the same system. The generic **[Auto]** section is carried out for *all* aircraft changes whilst the specific ones are only applied to a matching profile.

Each Auto section contains a series of numbered lines (1=..., 2=... etc) each of which is either a Lua command, or a Macro call. For example:

> **[Auto.737]**
> **1=Lua SetMyOffsets**
> **2=737 OHD:Air Allbleeds**

When Lua calls run a plug-in which doesn't self-terminate, the plug-in thread still running is killed automatically on an aircraft/profile change.

Axis Assignments

Axis assignments are saved in the [Axes] section, or [Axes.<aircraft name>] for aircraft specific assignments. Generic aircraft assignments can be made using the same parameter and name shortening as for the Buttons and Keyboard sections.

The polling interval can be changed by a parameter

> PollInterval=10

inserted into the main [Axes] section. The units are milliseconds, 10 being the default.

The format of the axis parameters in these sections is as follows:

For the main axis entry (explanation of values below):

> n=ja,(R)delta(/delay)

where the parentheses merely show optional parts, and

> j = joystick # (0 to 18, 16 to 18 being PFC)
> a = axis (XYZRUVSTPQMN)
> R is only present when "Raw" mode is selected
> delta is the delta value (eg 512, or 1 for Raw mode and POVs)
> /delay is an optional delay*, in milliseconds

When axis controls are assigned (the left part of the options), this is extended by the definition of the controls:

> n=ja,(R)delta(/delay),ForD,ctl1,ctl2,ctl3,ctl4

where

> ForD is an F for "FS control" or D for "Direct to FSUIPC calibration"
> ctl1 to ctl4 are the control numbers, or zero where unassigned. For Direct mode, these are the calibration indices, 1–4 on Page 1 of calibrations, 5–8 on page 2, etc. Numbers 45–48 are the "dual" controls, equating to others depending on whether FS is in flight mode or Slew mode.

> * FSUIPC can apply delays to any axis assigned through its Axis Assignment facilities. The delay is limited to a minimum of 2 x the axis polling interval (which defaults to 10 mSecs) and a maximum of 200 x this interval (i.e. 2 seconds with the default polling interval).
>
> Delays for axes have to be edited in the INI file. There is no facility to change them or even see them in the option screens. Delays of 200 mSecs or more should be reasonably accurately maintained most of the time, but short ones could vary quite a bit, the smaller you set them, because of the granularity of the polling interval and the sharing of the processor with other things going on in FS.

Here's an example of an axis assigned to the FSUIPC Spoiler, with a 1 second delay:

> 0=0Y,256/1000,D,22,0,0,0

If the axis is programmed to send controls based on the axis passing through zones (the right side of the options), there will also be entries for each such assignment, thus:

> n=ja,UDorB(R),low,high,ctl,param

where

> UDorB is U for Up, D for Down or B for Both
> R optionally specifies Repeat
> low and high give the axis values for the zone
> ctl and param are the Control numbers, and Parameter where used.

Here's an example for a Gear lever:

> 1=0Z,256/500
> 2=0Z,U,6400,16383,66079,0
> 3=0Z,D,-16384,-13783,66080,0

Note that the delay option (here half a second) still goes on the main axis entry, the one defining the delta (and "Raw" mode if applicable).

You can edit the INI file whilst FS is running, then simply going to the Axis Assignment options page and clicking the reload button at the bottom of the window.

The repeat rate of controls assigned to axes in this "range assignment" area can be changed. The parameter for this goes into the relevant [Axes] section of the FSUIPC7.INI file, and is:

RangeRepeatRate=10

with the default being 10 per second as shown. The range is 1 to 100, but be aware that this is approximate, depending on other loads on FS, and will vary as much as 30% either way.

You can also exclude axes movements from being recognised in the dialog permanently by using the **IgnoreThese** parameter in the relevant [Axes] section This can be used to list a number of axes which are to be ignored by FSUIPC in the Axis Assignment tab. This is to deal with faulty axis signals (usually from a dodgy potentiometer) and thus preventing the others from being registered on the screen ready to program. The parameter takes this form:

IgnoreThese= j.a, j.a, ...

listing the joystick number (j) and axis letter (a) of each axis to be ignored. To make it easy, you can edit the INI file whilst in the Axis Assignment dialogue and simply press "reload all assignments" to activate the changes.

This functionality is similar to what occurs if you press the **Ignore Axis** button in the Axis Assignment tab, except for the fact that those would be temporary and only valid for the current session, whereas using this ini parameter makes this permanent.

Note that the action of ignoring axes are only ignored in the dialogue—if they are already assigned the assignment will still be effective.

Additional parameters to scale input axis values

Axis values assigned in FSUIPC can be arithmetically adjusted before being passed onto FSUIPC calibration (or to FS via FS controls). To do this you assign the axis as normal, then edit the FSUIPC7.INI file. Find the axis assignment there, in the relevant [Axes] section, and add one or both of these parameters to the end:

,*<number>   to multiply the axis value by <number>. This can be a fraction, such as 0.5 (to divide by 2), and it can be negative, to reverse the axis direction. Fractions can be expressed to 7 decimal places.

,+<number> or -<number>
            to add or subtract a number (an integer, no fractions) to or from the value.

If both parameters are given, the multiplication must come first, and is performed first. The resulting value is constrained to be in the range -16384 to +16383 *except* when the assignment is to an offset, where no restriction is imposed..

As an example, if the normal input range of an axis is -16384 to + 16383 and you only want the positive half, but need to still use the whole of the lever movement:

,*0.5,+8192

would be added to the assignment. The *0.5 changes the range to -8192 to +8191, and then adding 8192 gives 0 to +16383.

After editing, just tell FSUIPC to reload the axis assignments (a button on the Axes page). You won't see the results there, but you will in the calibrations.

Special scaling for axis operation via offsets 3BA8–3BC4 (the "PFC" axes)

Axis values written to the erstwhile "PFC axis" offsets, 3BA8-3BC4, are now automatically ranged if RAW mode is not selected and the axis has not yet been assigned. This makes those offsets much more suitable to use by additional hardware which is not recognised by Windows as a joystick type, or (especially) to using any sort of joystick axes from a WideClient application or Lua plug-in.

The default range is still assumed as 0–127 (i.e. 7 bits), which suits the PFC axes, but this is expanded to anything from 255 (8 bits) to 65535 (16 bits). You just need to make sure your program or device supplies the highest value so the range is set correctly *before* making any assignment to the axis in FSUIPC. Note that the values are still assumed to be

always positive, so you may need to adjust them, by program or by using the multiplier/add parameters in the INI after assignment.

The range is saved at the end of the relevant axis assignment line in the relevant FSUIPC [Axes] section as ",Rn" where n = 1 for 8 bits, up to 9 for 16 bits. This will of course only apply to joysticks 16–18 as these are the joystick numbers applied to these offsets. This parameter can also be added manually for already-assigned axes on joysticks 16–18.

Programs: facilities to load and run additional programs

FSUIPC can, as an extra, cause other programs to be run each time you load and run Flight simulator. Details of what programs to be run are provided in an additional section in the FSUIPC7.INI file. This section cannot be edited in the on-line FSUIPC options dialogues. You need to edit the details directly in the INI file.

The additional section is

> [Programs]

and can contain up to 32 requests to run other programs—up to 16 "Run" parameters Run1 to Run16, and up to 16 "RunIf" parameters, RunIf1 to RunIf16. Both sets are otherwise identical in format. The only difference is that the RunIf programs are not run if they appear to be already running. The ordinary "Run" programs will be loaded without such checking.

The format is simply:

> RunN=(Options,)<full pathname of program to be run>
> or   RunIfN=(Options,)<full pathname of program to be run>

where N runs from 1 to 16. Details of options are given below, but if none are required the parameter simplifies into just the full pathname.

For example: Run1=D:\RadarContact\RCV4.exe

might be used to run Radar Contact version 4.

If the program or path name contains spaces or needs command-line parameters, then these can be included by enclosing the program path in quotes, so that the space(s) needed don't cause problems. You may also need to include quotes around the parameters if they includes spaces.

For example: Run2="c:\Program Files\epic\loadepic" "fs98jet"

The programs are loaded in order of the run number, 1–16. If a mixture of Run and RunIf parameters are given, the order is Run1, RunIf1, Run2, RunIf2, and so on.

The Options you can use are as follows:

HIDE	tries to get the program to hide itself when it runs. This is only possible if the program defines its window to use default settings, so it isn't very useful for many programs, unfortunately.
MIN	
MAX	similar considerations as for HIDE apply to these options, to MINimize or MAXimize the resulting programs window.
HIGH	runs the program at higher priority than FS. ***Use with care!*** Messing about with priorities doesn't work well in all circumstances, and FSX may not like it much.
CLOSE	closes the program tidily (if possible) when FS is terminated.
KILL	forcibly terminates the program, if possible, when FS is terminated.
LOW	runs the program at IDLE priority. Depending on what the program does, this may actually effectively stop it until you direct user focus to it, as FS tends to soak up all Idle time.
READY	delays loading and running the program until FS is up and ready to fly, and FSUIPC can supply valid data through its IPC interface (i.e. MSFS not in the main menu system). (This parameter may, of course, result in the programs being run in a different order to that specified by the Run number).
CONNECTED	delays loading and running the program until FSUIPC is connected to MSFS and has received initial data to populate the offsets. Note that MSFS may still be in the main menu system when this occurs. (This parameter may, of course, result in the programs being run in a different order to that specified by the Run number).
DELAY[=n]	delays the running of the program by the number of seconds given, (integer between 1 and 60), otherwise 10 seconds by default.
AM=n	A processor code affinity mask can be specified for each program individually. This is by inserting an extra parameter in the form: AM=n with n in decimal, or

AM=Xn with n in hexadecimal.

For example:

Run1=AM=x52,CLOSE,C:\PM\PFD.EXE

Of these really only CLOSE, KILL, READY and AM= are of general use. If you want to apply more than one option, list them separated by commas, but *no spaces*. For example:

RunIf1=READY,KILL,AM=82,D:\FS2002\WeatherSet.exe

Assignment of additional axis controls
(Reverser, Aileron and Rudder Trims, and Cowl Flaps)

There are no axis controls provided in FS for jet thrust reversing nor for aileron or rudder trim or even for setting the cowl flaps. To get around this, and for other axis assignments not possible in FS's menus, please check the Axis Assignment facilities in the FSUIPC options. You'll find a lot more axis type controls you can assign there, and by directing the Aileron Trim, Rudder Trim and Cowl Flaps to FSUIPC's own calibrations, they can be operational within minutes. FSUIPC's Joystick section (on page 7 or 8) deals with these.

The Reverser control is special to FSUIPC and can be assigned and calibrated in the same way. Additionally there's another controlling parameter:

MaxThrottleForReverser=0

This controls the interlock—the reverser will not engage until all throttles are reduced to this setting (normally 0, or idle). You can try a non-zero value here if you cannot calibrate your throttles to produce a stable idle zero.

Multiple Joysticks for Multiple Pilots

FSUIPC's axis assignments allows any of your joystick axes to be assigned to any of FS's or FSUIPC's axis controls, and there's no restrictions on how many you can assign to any of them. So that's the first problem solved – you can assign two sets of yokes, rudders, whatever, to the same controls.

Both FSUIPC and FS take notice of the last movement in an axis. They don't "poll" them to get regular inputs, but only see changes coming from them. So both will see the last change from multiple axes. However, that might be from an unwanted jitter or small accidental movement. So, *provided you assign your axes for Direct FSUIPC Calibration* (as opposed to an FS control), FSUIPC now arbitrates, selecting the axis with the highest deflection (defined here as a difference from zero).

Note, however, that it still only sees axes when they change, so even if one axis is held at an large deflection, once another axis for the same control moves to a similar or higher position, that takes control then even if it moves lower than the held on—the latter is effectively "out of it" until it is moved.

Note that you will need to calibrate all controls so that the ones controlling the same values are as close as possible in range and response. Do this first in Windows Control Panel, then, after making the above adjustments and assignments, in FSUIPC. Calibrate dead zones at the ends (and in the centre for aileron, elevator and rudder) to "cover up" any discrepancies—in other words, calibrate for the worst of each.

HELICOPTER PITCH and BANK TRIM facilities

A facility to operate pitch and bank trims on helicopters is provided. This uses the normal FS elevator and aileron trim controls (and axes) to modify the end value on the "Y" (elevator) and "X" (aileron) axis of the cyclic. To use this you need to ensure that the axes are calibrated through FSUIPC (as the elevator and aileron axes respectively), and add

ApplyHeloTrim=Both

to the relevant [JoystickCalibration …] section(s) in FSUIPC7.INI. Note that, as a precaution, the trim value will never be added to the relevant axis if the normal trim value is non-zero.

This new "helo trim" values are maintained in IPC offsets as follows:

0BBE   2 bytes   16-bit Helo Pitch Trim value, range –16383 to +16383

0C06   2 bytes   16-bit Helo Bank Trim value, range –16383 to +16383

Both of these can be written to for external program control.

45

Note that if you only require a pitch trim you can set

ApplyHeloTrim=Yes

Instead of 'both'. The aileron/bank axis and trim values will then be left alone.

FSUIPC WASM Module

The FSUIPC WASM module is an FSUIPC add-on that is, by default, installed into the *Community* folder of MSFS when you install FSUIPC7. This module provides FSUIPC7 with access to the aircraft's local panel variables (known as lvars or L:vars) and HTML variables (known as hvars or H:vars). As from FSUIPC version 7.3.7, the WASM features in FSUIPC will be automatically enabled if the WASM module is installed in your *Community* folder. If you do not want to use these features, you can disable the WASM by selecting the **Disable** option, which can be found under the **Add-ons → WASM** menu entry. If the WASM menu is not visible, then the FSUIPC WASM folder *fsuipc-lvar-module* was not found under your MSFS' *Community* folder.

Once enabled, the FSUIPC WASM module will allow you to use both lvars (as in previous versions of FSUIPC) and hvars (HTML variables) as well as to execute calculator code and use calculator code *presets*, such as thise provided by MobiFlight (and included with FSUIPC7).

Using Lvars

Lvars are automatically discovered by the FSUIPC WASM module and made available to FSUIPC for use as in previous versions of FSUIPC (i.e. FSUIPC4, FSUIPC5 and FSUIPC6), via Macros and lua facilities/functions. Note that lvars are not cleared (by MSFS) on change of aircraft, and so if you switch aircraft then lvars created for the previous aircraft will also be available (but they may be of no use). Up to 10,220 lvars are recognised. However, it is recommended to restart MSFS if you receive > 8000 or so lvars, as many of the lvars you see will be for previously loaded aircraft and of no use.

Adding Lvars to Offsets

A facility is provided to add Lvars directly to FSUIPC offsets, for both reading and writing (i.e. updating an lvar value by updating the offset). This facility is currently only available by editing your *FSUIPC7.ini* file.

To add an lvar to an offset, you need to open your FSUIPC7.ini file in an editor (e.g. Notepad++) and add a new section to specify the lvars you would like to add and the offset it should be added to. This can be a *general* section, which is applicable to ALL aircraft, which can be achieved by adding the following section name:

　`[LvarOffsets]`

However, as lvars are aircraft specific, it is usually better to add this as a profile specific section, in which case you need to append your profile name preceded by a full stop. So, for a profile called 'TwinProps', for example, the new section name would be:

　`[LvarOffsets.TwinProps]`

Note that a profile LvarOffsets section will replace/supersede a general LvarOffsets section, not augment it.

Note that if you are using profiles in separate files, then you should place the `[LvarOffsets]` section in your profile .ini file.

Once you have created the section, you can add the lvars to the offsets you require by adding lines of the following format to this section:

　<index>=<lvar name>=<size><offset>

where

　- *index* is the index number of the entry, starting from 0 with a max value of 1023 (i.e. maximum of 1024 entries)
　- *lvar name* is the name of the lvar, optionally preceded by '**L:**'
　- *size* designates the size/type of the offset. This can be omitted and a size/type of 8 bytes/double will be used, 　　otherwise you can use the following designators and the lvar value (double) will be converted to the appropriate
size/type:

　　　　SB – signed byte (1 byte)
　　　　UB – unsigned byte (1 byte)
　　　　SW – signed word (2 bytes), use for signed short
　　　　UW – unsigned word (2 bytes), use for unsigned short
　　　　SD – signed double-word (DWORD) (4 bytes), use for signed int
　　　　UD – unsigned double-word (4 bytes), use for unsigned int
　　　　F – floating point number (4 bytes), use for float

You can edit this section while FSUIPC is running. If running, to load any changes, use the **Add-ons → WASM → Reload** function.

Be aware that the specified offset must be boundary-aligned to the size of the offset. This means that
  - if the size of the offset is is 8 bytes, the offset address must end in 0 or 8
  - if the size of the offset is is 4 bytes, the offset address must end in 0, 4, 8 or C
  - if the size of the offset is is 2 bytes, the offset address must end in 0,2,4,6,8,A,C or E

As a simple example, to add the lvar **XMLVAR_YokeHidden1** as a on/off boolean flag for the stock B747 to my **B747** profile, I would add the following to my FSUIPC7.ini file:

```
[LvarOffsets.B747]
0=L:XMLVAR_YokeHidden1=UB0xA000
```

Once an lvar has been added to an offset, you can use the offset for the lvar value as you would any other offset. You can also update the lvar by updating the offset value, using, for example, one of the FSUIPC Offset Controls (see page 35), such as **Offset Word Set**, or **Offset Byte Togglebits**. Make sure that the control that you use matches the size of the offset defined to hold the value. So, taking my previous example using the lvar **XMLVAR_YokeHidden1**, to assign a button/switch to control this lvar via the offset I have assigned (A000), I would assign to the control **Offset Byte Togglebits**, giving A000 as the offset and 1 (or x1) as the parameter.

Using Hvars

Hvars need to be known by FSUIPC7 before they can be used. This is done by the use of a hvar file (i.e. a file with extension .hvar), which can be located either the WASM persistent storage area (see following section for location details) or under the FSUIPC WASM's modules folder. Hvar files are loaded if the filename of the hvar file (without the extension) is a substring match to the currently loaded aircraft. If hvar files are found for the current loaded aircraft under the persistent storage area, then these are used and any hvar files located under the WASM modules folder will be ignored.

Some example aircraft specific hvar files are already provided and installed under the WASM modules folder. A selection of hvar files is also installed (if selected to do so during the installation process) under your FSUIPC7 installation folder, under a folder called **HvarFiles**. To use these files, copy them to your WASM persistent storage are and rename them so that the name is a substring match on the aircraft for which you wish to use them.

Multiple hvar files can be loaded per aircraft, but it is recommended to combine any hvar files that you wish to use for an aircraft into a single hvar file for that aircraft.

There is currently a maximum limit of 584 hvars that can be used per aircraft. Of course, you can also remove individual hvars from these files if they are not needed/used.

If you change the *.hvar file for the current aircraft while FSUIPC7 is running, then you can instruct FSUIPC7 to reload this file by using the **Add-ons → WASM → Reload** option. This will also re-scan for lvars.

Note also that hvars to not have an associated value and can only be activated (or *set,* without a value). You can also test for the existence and effect of an hvar before you add it to your aircraft' hvar file by using the **Add-ons → WASM→ Execute Calculator Code..** menu option. To do this, wrap the hvar in the following way, for example:
    (>H:A320_Neo_CDU_MODE_MANAGED_SPEED)

Once a hvar has been made know to FSUIPC7, it can be used in assignments either by using a macro file (see pages 35 & 36) or in a lua script by using the **ipc.activateHvar(<*hvarName*>)** function.

However, please note that it is far easier to assign to hvars using presets, as you do not need to make the hvars known to FSUIPC7 using this method. See below for detailed instructions on preset use, but basically you just need to add a line to the *myevents.txt* file (you need to create this file if not present) ri define the preset name and the calculator code to execute. As an example, to create a preset ti activate the hvar H:A320_Neo_CDU_MODE_MANAGED_SPEED you would add the line:
    A320Neo_CDU_ManagedSpeedMode#(>H:A320_Neo_CDU_MODE_MANAGED_SPEED)
and then assign to the preset **A320Neo CDU ManagedSpeedMode**.

Issue with Lvar / Hvar names > 55 characters

Note that all lvar/hvar names used in FSUIPC7 are (currently) restricted to 55 characters (not including the terminating \ 0). Lvars with longer names will be truncated to 55 characters, hvars with longer names will not be loaded. For lvars, this should not be an issue unless there are multiple lvars with names > 55 characters in length AND the first 55 characters are the same. When this occurs, there is no way for FSUIPC to distinguish which lvar is being referred to when the name is used (e.g. in the lvars to offsets functionality), and so the lvar with the lowest id will be taken.

I am not sure what to do about this at the moment, but hopefully its not an issue (as most, if not all, lvars should differ in the first 55 characters). If you do fund this to be a problem, please post in the FSUIPC7 support forum and I can take another look. One possibility, to allow such lvars to be used, would be to implement lua functions for reading/writing lvars based upon the id (rather than the name), but this solution may also be difficult to use (e.g. how do you know the id of the lvar?).

Using Calculator Code Presets

A calculator code *preset* is a name assigned to a string of calculator code, which can also be parameterized. Calculator code presets are made know to FSUIPC7 by two files that must be located in the root FSUIPC7 installation folder. The format of these files are identical, and must have the names *events.txt* and *myevents.txt*. You do not have to use both files, but both will be loaded if present.

The format of the files is simple – each line must either be a comment, starting with '//' (see below), or define a preset in the format:

>*<PresetName>#<Calculator Code>*

where

>*<PresetName>* is the name of the preset (max 63 characters)
>*<Calculator Code>* is the calculator code assigned to the preset name (max 1023 characters)

Note that this is the same format as used by the MobiFlight *events.txt* file, so the MobiFlight presets can be used directly. This file will be installed automatically by the FSUIPC7 installer, providing the ooption for this is checked during installation (by default, this is installed). Please see the section below on the MobiFlight presets.

Two files are provided so that you can use the MobiFlight *events.txt* file directly, and use the *myevents.txt* file for your own presets. You can then update the MobiFlight *events.txt* file at regular intervals (it is updated frequently!) without affecting your own presets. However, it is highly recommended that you use the MobiFlight HubHop resource (see **MobiFlight Presets** section below) for reporting any calculator code so that this can be shared by the community.

All preset names found in these two files will be available for direct assignment in both the Buttons and Keys UI panels, in a drop-down controls menu available by checking the '**Select for Preset**' box. The preset name in the controls menu will be changed to replace the '_' character with a space, and each word will have a capitalized first letter. It is recommended (but not prohibited) to NOT use repeat on preset assignments, either on button or key assignments.

You can also assign an axis to a preset using '**Send Preset to FS**'. Note however that only presets that accept a parameter are available to assign to an axis (in the left hand-side of the axis assignment panel). You can also send a preset when entering/leaving an axis range on the right-hand side of the axis assignment panel by checking the '**Select for Preset**' box, and the full list of presets will be available.

The calculator code can also be parameterised by using the variable **$Param**. Any occurrence of such a string in the calculator will be replaced by the parameter assigned to the preset (in the UI panels) before being executed. The **@** symbol can also now be used as a parameter placeholder instead of **$Param**, as this is what MobiFlight use for the potentiometer input presets.

Both of the preset files (*events.txt* and *myevents.txt*) can also contain comments. Comments that start with the characters '//' are processed and used to create the tree-view of presets, available from the '**Find Preset...**' button in both the button and key assignments panels. When such a comment is found, all subsequent preset definitions (until the next '//' comment is encountered) will be associated to the tree-node obtained by parsing this comment. The comment is parsed by using the '/' character to deliminate the tree node names, so that, for example, when a comment such as

>`//Asobo/C172/Lights`

is encountered, the subsequent presets will be placed under the tree node

>`Asobo → C172 → Lights`

Presets found in the **events.txt** file will automatically be put under the root node **MobiFlight**, and the presets in the *myevents.txt* file will be placed under the root node **Personal**.

Comments starting '///' will be ignored completely.

Preset controls can be activated by name using offset 0x7C50 – see the Offset status document for details on how to use this offset. You can also activate presets in lua scripts using this offset and the *ipc.writeSTR* function, or you can now also use the ipc.execPreset function which is simpler and also accepts an optional parameter. Please take into account that the preset name must be given as written in the **events.txt** or **myevents.txt** files.

Preset controls can also be used via the general facility for sending any control to the FS at offset 0x3110 (also available in unregistered versions). The control number for the preset will be the index number of the preset + 4194304 (0x400000). The index number of the preset will be the line number of the preset (once comment lines have been excluded), starting from 0, with the line numbers for the presets in the *myevents.txt* file starting from where those in the events.txt file finish (i.e. each preset is given an index number, starting at 0, in the order that they are loaded). As this may be difficult to determine, it is recommended to only use the myevents.txt file if using this offset, and leave this file uncommented - but it is far easier to use the preset name via offset 0x7C50 rather than using the preset control number.

MobiFlight Presets and the *events.txt* file

The Mobiflight presets, defined in the *events.txt* file is included by kind permissions granted by the MobiFlight team. This file is generated from the MobiFlight HubHop resource (see https://hubhop.mobiflight.com/) which is a community driven project. If using this file, and if using presets in general, it is recommended that you subscribe to this resource and share/contribute your presets as well as reporting any errors that you find in any existing presets.

Note also that the MF *events.txt* included with FSUIPC7 is the file exported from the MobiFlight HubHop web site, and is updated with each FSUIPC7 release. However, as these presets are being continually updated, it will almost certainly be out-of-date compared to the presets available on the HubHop site. If there are new or updated presets that you would like to use, you can always download the latest *events.txt* file from that web site and use the file to replace the one in your FSUIPC7 installation folder (using the **Export presets** button – you need to create an account and login to do this).

Please also note that you cannot assign to MF *Output* presets, and these are not included in the provided MF *events.txt* file. There are alternative methods to read the output of such presets in FSUIPC7. Many such presets just report the value of an lvar – to read the value of an lvar in FSUIPC7 you can add the lvar value to a free FSUIPC offset and read the value from the offset. For more complicated output presets, you can reformulate the preset code to write the calculator code output value to your own lvar (which will be created on first use if not already present), and then read that lvar value by adding it to a free FSUIPC7 offset.

WASM module ini file and parameters

The WASM module takes its default settings from ini files. These are named *FSUIPC_WASM.ini* and can be located:
1. In the WASM folder, under your *Community/fsuipc-lvar-module* folder
2. In the WASM persistent storage area, which is:
    For Steam installs, in the following folder under your *user* account:
       `AppData\Roaming\Microsoft Flight Simulator\Packages\fsuipc-lvar-module\work`
    For MS Store installs, in the following folder under your *user* account:
       `AppData\Local\Packages\Microsoft.FlightSimulator_8wekyb3d8bbwe\LocalState\Packages\fsuipc-lvar-module\work`
    The FSUIPC WASM log file (*FSUIPC_WASM.log*) can also be found in this location.

Parameters found in location 2 (WASM persistent storage) will take president and overwrite any parameters found in the first location. A default ini file is installed with the WASM and can be found in location 1. It is recommended to leave this file as is, and copy to your persistent storage area and modify as and when needed from there.

The ini file contains one section, *[General],* and takes the following parameters:

   LogLevel=Info: defines the logging level of the WASM. Possible values are: *Disable*, *Info*, *Debug*, *Trace*, *Enable*

   LogType=File: defines where the log goes to, with possible values: *File*, *Console*, *Both*

   LvarUpdateFrequency=6Hz: lvar values can be updated asynchronously by the WASM and pushed to the the clients, or can be updated on request from a WASM client. This parameter controls the frequency of these updates, and can take one of the following values: *Second*, *6Hz*, *VisualFrame*.

Lvar value update requests from clients are ignored unless this parameter is set to *Off*. If set to *Off*, make sure that one (and only one) WASM client (e.g. FSUIPC7) is set to request lvar values to be updated. For doing this in FSUIPC7, again see the section below (**FSUIPC7 WAPI ini parameters**). NOTE: the use of *Off* for this parameter is still available but has been depracted – this option will be removed in future updates. If usung this option, please update.

LvarScanDelay=5: the delay in seconds between when the Ready To Fly stage is reached and the initial scan for lvars is performed. It is recommended to increase this if using complex aircraft or airliners as it can take a while for some lvars to be created for such aircraft.
Note that lvars created after the initial scan will be found and pushed to clients if the **LvarScanFrequency** is non-zero.

LvarScanFrequency=-5: the frequency of scans to detect new lvars. Positive numbers indicate maximum number of scans per second and negative numbers indicate the minimum number of seconds between scans.
Values for this parameter are constrained to between -30 and +4
A value of 0 will disable scanning for new lvars.

LvarScanPeriod=2: when positive, this is the time period, starting from aircraft load, when scans to detect new lvars will be performed. When negative, this is the number of scans that return no new lvars detected before lvar automatic lvar scanning is disabled for the currently loaded aircraft. When this limit passes, no new scans will be performed except when asked (via a reload request) or when an lvar is explicitly created by FSUIPC7.

UseAirLocForHvars=No: similar to the FSUIPC7 ini parameter **UseAirLocForProfiles**, setting this parameter to **Yes** will change the hvar matching from using the aircraft name to using the folder name of the folder under which the currently loaded aircraft's aircraft.cfg file is located. This can provided a better match for different versions of the same aircraft, such as when using different liveries.

WAPI ini parameters

The *FSUIPC7.ini* file contains a specific section for parameter relating to the WASM interface used by FSUIPC, called the WAPI (the WASM Application Programming Interface) and named accordingly as *[WAPI]*. This section accepts the following ini parameters:

EnableWAPI=No: defines whetther the WASM interface is enabled or not. This item can be controlled using the **Add-on → WASM** menu entry **Enable / Disable**.

LogSeparately=No: defines whether a separate log file should be used for logging messages from the WAPI. Set to Yes to use a separate log file, which will be called *FSUIPC_WAPI.log,* and will be located in the FSUIPC7 installation folder.
Note that there currently seem to be some issues when logging to a separate file, so it is suggested to not use this parameter for the time being.

LvarUpdateFrequency=0: defines the update frequency of the lvar values maintained in FSUIPC7. By default, this is set to 0 and the lvar values are updated asynchronously by the WASM module (by default, on each *Frame*). This is the recommended setting, and other values for this ini parameter are deprecated. This oprion will be removed at some point.

UseSimConnection: you can use this parameter to instruct FSUIPC7 to ise a specific simconnect connection (from your SimConnect.cfg file). By default, the default local connection (-1) is used.

LogLevel=Info: defines the logging level of the WAPI. Possible values are: *Disable, Info, Debug, Trace*

Adding Input Events to Offsets

A facility is provided to add Input Events directly to FSUIPC offsets, for both reading and writing (i.e. setting an input event by updating the offset). This facility is currently only available by editing your *FSUIPC7.ini* file.

To add an input event to an offset, you need to open your FSUIPC7.ini file in an editor (e.g. Notepad++) and add a new section to specify the input events you would like to add and the offset it should be added to. This can be a *general* section, which is applicable to ALL aircraft, which can be achieved by adding the following section name:

  [InputEventOffsets]

However, as Input Events are aircraft specific, it is usually better to add this as a profile specific section, in which case you need to append your profile name preceded by a full stop. So, for a profile called 'TwinProps', for example, the new section name would be:

  [InputEventOffsets.TwinProps]

Note that any global Input Events will also be added to FSUIPC offsets when a profile-specific **[InputEventOffsets.*xxx*]** is loaded.

Note that if you are using profiles in separate files, then you should place the **[InputEventOffsets]** section in your profile .ini file.

Once you have created the section, you can add the input events to the offsets you require by adding lines of the following format to this section:

  <index>=<input event name>=<size><offset>

where

  - *index* is the index number of the entry, starting from 0 with a max value of 1023 (i.e. maximum of 1024 entries)
  - *input event name* is the name of the input event, optionally preceded by '**I:**' or '**B:**'
  - *size* designates the size/type of the offset. This can be omitted and a size/type of 8 bytes/double will be used,
    otherwise you can use the following designators and the input event value (double) will be converted to the

appropriate size/type:

      SB – signed byte (1 byte)
      UB – unsigned byte (1 byte)
      SW – signed word (2 bytes), use for signed short
      UW – unsigned word (2 bytes), use for unsigned short
      SD – signed double-word (DWORD) (4 bytes), use for signed int
      UD – unsigned double-word (4 bytes), use for unsigned int
      F – floating point number (4 bytes), use for float

You can edit this section while FSUIPC is running, but changes only take effect whn an aircraft is loaded and the input event names are received from the simultor.

Be aware that the specified offset must be boundary-aligned to the size of the offset. This means that
  - if the size of the offset is is 8 bytes, the offset address must end in 0 or 8
  - if the size of the offset is is 4 bytes, the offset address must end in 0, 4, 8 or C
  - if the size of the offset is 2 bytes, the offset address must end in 0,2,4,6,8,A,C or E

As a simple example, to add the input event **HANDLING_Yoke_Hider_1** as a on/off boolean flag for the stock Cessna 172 to my **Cessna** profile, I would add the following to my FSUIPC7.ini file:

```
[InputEventOffsets .Cessna]
0=HANDLING_Yoke_Hider_1=UB0xA010
```

Once an input event has been added to an offset, you can use the offset for the input event value as you would any other offset. You can also fire/set the input event by updating the offset value, using, for example, one of the FSUIPC Offset Controls (see page 35), such as **Offset Word Set**, or **Offset Byte Togglebits**. Make sure that the control that you use matches the size of the offset defined to hold the value. So, taking my previous example using the input event **HANDLING_Yoke_Hider_1**, to assign a button/switch to set this input event via the offset I have assigned (A010), I would assign to the control **Offset Byte Togglebits**, giving A010 as the offset and 1 (or x1) as the parameter.

Note that some Input Events do not have an associated value, and also string values are also possible, although there are currently no Input Events (as of MSFS version 1.37.2.0) that currently have a string value. String values are currently not supprted by FSUIPC7 – I will look into adding such support when any string-valued Input Events are available.

APPENDIX 1: Do more with your joystick!

This section is from a contribution graciously made by an intrepid FSUIPC user. I hope you will find it useful. Apart from formatting to fit this current document I've left it exactly as originally submitted. It was written for P3D but also applies to FSUIPC7 for MSFS.

During the past flightsimming years, the PC flight simulators have become more and more professional and more complex. Very sophisticated airplanes can be downloaded for free or purchased for a reasonable price. Many of them includes all bells and whistles in a way that "flightsimming" is no "game" any more, and for many among us it becomes more and more a "real flight simulator" as used in the real flying world. Some have built very real looking cockpits with instrument panels with every switch and control in its right place; others like myself are still using their joystick and keyboard.

As I have a small computer desk, it is not so handy to use the keyboard and my joystick together especially for flightsimming. A cockpit is overloaded with devices to be set, numerous switches have to be used, many settings must be executed via keyboard entries and with a joystick with a scrubby eight buttons for all the remaining commands, it seems to be impossible to do this in any user-friendly way

In P3D4 and previous simulators, some, but not all, commands have already been to the joystick buttons by default and. these can modified by the Assignments options in P3D. By selecting a command from a list and defining a button of your joystick, the activation of this button will execute the selected command. There is also an option that repeats the command as long as the button is pushed.

I have an 8-button joystick but the simple default joystick programming of the eight buttons was not sufficient anymore. I need more commands, more sophistication on my joystick. So I searched for a solution: because my joystick was absolutely necessary the only option is then to eliminate keyboard entries as much as possible.

Lucky there are still some smart guys on this world, guys like Pete Dowson. Pete is well known for his excellent FSUIPC.DLL add-on module for MS flight simulators. This module makes it possible to correct some flaws in FS and to enhance FS, and must be considered as 'a must have' for the whole FS community. But FSUIPC includes also many features that the modal user can use to his advantage. One of these features for licensed FSUIPC users is "joystick and keyboard button programming".

Originally, Pete has provided this feature for owners of Goflight and Epic devices but this can also be used for your joystick too! I have written this guideline on the request of Pete because we realize that only a few FS users use this powerful tool as intended. I will try to explain the marvellous things you can do with this superb programming tool. A few weeks ago I didn't realized it myself, but now, oh boy!

I will explain some programming tricks I'm using in the button programming of my own Sidewinder Force Feedback Joystick 2 from Microsoft.

The following documentation is needed before you can start the programming:

First: A fully user-licensed FSUIPC installed in your copy of P3Dv4.

Second: the "FSUIPC for Advanced Users" manual [*the one now before you*]. Please, read very carefully the chapters concerning keyboard and joystick button programming, especially the section about compound buttons.

Third: the "List of FSX and P3D Controls" document which will have also been installed for you in your FSUIPC Documents folder – or, better, the TXT file version generated for you in that same folder by FSUIPC which then matches your installed version of P3D4 ("***Controls List for P3D4 Build xxxxx***")

To be sure that all commands will be executed the way you have programmed them, almost all the default programming of the buttons in Flight Simulator must be removed. I have removed them all, except the Hat button programming.

I use the two buttons on the left of the joystick pedestal to set conditions for the selection of commands assigned to the six other buttons. I also include the tricks you can use if you would use three buttons to set conditions.

Let us start with the first case. The two buttons used to define a condition, are labelled "7" and "8" on the joystick. The lowest button label on the joystick is being "1". For programming however the button numbering starts with button "0" for the button with label 1, "1" for button 2 etc. And so, again in our case, the conditions are programmed by button "6" and "7". Four possibilities are created by the button status, pushed or released, of a combination of two buttons:

1. button 6 and 7 are both up,
2. button 6 is down and button 7 is up,
3. button 7 is down and button 6 is up
4. button 6 and button 7 are both down.

The status of these two buttons together with an action of one of the other six buttons can be used to program a flight simulator command. In fact we can now assign up to 4 commands per button or 24 commands to the six remaining buttons (even 48, because we can program a function if one of the "action" buttons is going down and another when the same button goes up again)

The following can be done with a combination of three buttons:

1. button 5, 6 and 7 are all three up
2. button 5 is down, 6 and 7 up
3. button 6 is down, 5 and 7 are up
4. button 7 is down, 5 and 6 are up
5. button 5 and 6 are down, button 7 is up
6. button 5 and 7 are down, 6 is up
7. button 6 and 7 are down, button 5 is up
8. button 5, 6 and 7 are all three down

There are now 8 possibilities in the combination and 5 remaining buttons which gives 40 and up to 80 commands that could be assigned to these 5 buttons.

As specified in the section on Button programming, earlier, two kinds of commands can be generated: use the button combination to simulate a hit of a key combination on the keyboard, or use the joystick button combination to generate an "internal" FS command. A list of all the possibilities for these commands can be found in the "List of FSX and P3D Controls" document.

Let us take a few rules out of a button programming as examples:

        3=CP(-0,6)(-0,7)0,3,C65615,0
                ...
                ...
        9=CP(+0,6)(-0,7)0,3,C65769,0

In these both cases the active button (the button that is generating the command) is button 3 (with label 4 on the joystick). In the first case the command "65615" is generated when button 6 and 7 are up and button 3 is going down. C65615 will generate an "Elevator Trim Up", the same command as the default joystick button programming. The "CP" syntax defines that the command will be only executed once, even if the button 3 is holding down.

However, by holding down the "6" button ("7" on the MS joystick) and activating button 3, FSUIPC will generate a "65769, Propeller Pitch Increment", command. This command is not a default joystick button command, but a command that, if it was not programmed that way, had to be entered by a button combination on the keyboard.

By defining the button combination with a "CR", the command will be repeated until the action button is released again, which is in our application more advantageous. And in fact, the repeat function is used on both commands:

        2=CR(-0,6)(-0,7)0,2,C65607,0
        3=CR(-0,6)(-0,7)0,3,C65615,0
                ...
                ...
        8=CR(+0,6)(-0,7)0,2,C65771,0
        9=CR(+0,6)(-0,7)0,3,C65769,0

The buttons "2" and "3" are used here to trim up/down (rule 2 and 3) with button 6 and 7 up. The same buttons, but now with button 6 activated while button 7 is up, controls the propeller pitch.

I assigned another two commands to the same "2" and "3" buttons; also I programmed the combination with the 7 button for "Mixture Incr" and "Mixture Decr" in rule 19 and 20:

        19=CR(-0,6)(+0,7)0,2,C65775,0
        20=CR(-0,6)(+0,7)0,3,C65777,0

I must emphasise here that FSUIPC uses the *status* (up or down) of the buttons in the compound combinations (+/-j,b) (+/-j,b) for a condition, but the *changes* of the button status, in fact "the pushing" or "the releasing" of a joystick button for the activation of the command, which is valid for one whole scan, meaning the check of all following button programming rules. This is important to remember.

On the MS joysticks, button 2 and 3 are very well placed for using them as increment and decrement functions and a lot of commands could be attached to them. However, we have used already 3 of the four condition statuses. So if we only

use the combination of two buttons and like to attach much more commands to these buttons we have to find another way.

First of all, with all preceding versions up to and including 3.14, FSUIPC allowed compound combinations of the status of up to two buttons, and not more than two buttons, to create a condition. In the newer versions the status of 16 buttons can used to create a condition—but the explained tricks will still be valid.

In fact, the button programming does not work directly with the buttons because FSUIPC stores the status of a button in a "flag", an internal storage space, during a process cycle or scan of all programmed button rules. The programmed conditions use these flags to define the result status of the programmed condition.

FSUIPC saves now the status for up to 32 buttons of up to 16 joysticks, which means 512 flags for 512 buttons!

From these 512 flags, only 8 are used for the 8 buttons of my joystick and the rest of these flags likes to be wasted space. Not entirely! Because Peter has provided some commands to set, toggle or reset the flags, even if they are not "connected" to a button. So an instruction can be used to set or reset a flag and to use the flag afterwards in a condition. And because there is no connection to an existing button, the status of the flag is entirely dependent of the programmed instructions that are given for that particular flag.

What we are going to do now is to make ONE flag reflecting the condition of the TWO buttons, so that this flag can be used together with the status of another button, to create another condition. To do this, I use the following tricks:

        ; Flag 10 follows keys (-6 AND -7)
        ;
        0=CU(-0,7)0,6,C1003,10
        1=CU(-0,6)0,7,C1003,10

When FS is started and the module FSUIPC.DLL is activated, all flags are reset. To be sure about the setting of flag 10 we have to "play a bit" with buttons 6 and 7. Playing a bit with these buttons at the beginning of our flight does not generate commands, because both buttons are "dead" buttons and they will not sent commands to FS (this is the same as the shift keys on your computer keyboard which are doing nothing on their own but only functioning together with other keys). The above rules are assuring that flag 10 will be set when both buttons are up or are going up:

    Rule 0:   when button 7 is up, and button 6 goes up, set flag 10

    Rule 1:   when button 6 is up, and button 7 goes up, set flag 10

The following rules are setting flags when one of the both buttons is going down. In these cases however we have to reset flag 10:

        ; Flag 11 follows keys (+6 AND -7)
        ;
        2=CP(-0,7)0,6,C1004,10
        3=CP(-0,7)0,6,C1003,11
        4=CU(F+0,11)0,6,C1004,11
        ;
        ; Flag 12 follows key (-6 AND +7)
        ;
        5=CP(-0,6)0,7,C1004,10
        6=CP(-0,6)0,7,C1003,12
        7=CU(F+0,12)0,7,C1004,12

The explanation of these programming rules is:

    Rule 2:   if button 6 goes down and button 7 is still up, reset flag 10.

    Rule 3:   if button 6 goes down and button 7 is still up, set flag 11 (remember that the action of the active button can be seen by all the following rules in the same scan).

    Rule 4:   when flag 11 is set and button 6 goes up, reset flag 11.

Flag 11 follows now the status of button 6 (up or down) while button 7 is up.

    Rule 5:   if button 7 goes down and button 6 is still up, reset flag 10.

    Rule 6:   if button 7 goes down and button 6 is still up, set flag 12.

    Rule 7:   when flag 12 is set and button 7 goes up, reset flag 12.

In this case flag 12 follows the status of button 7 (up or down) while button 6 is up.

Now follows a more tricky part because we want to make a "follower" for button 6 and 7 down, (if we wouldn't use a combination with both buttons down, then, in any case, we have to include rule 8 and 11 to be sure of a resetting of flags 11 and 12 when the conditions in the above rules aren't valid any more):

```
; Flag 13 follows key (+6 AND +7)
;
8=CP(+0,6)(F+0,11)0,7,C1004,11
9=CP(+0,6)0,7,C1003,13
10=CU(F+0,13)0,7,C1004,13
;
11=CP(+0,7)(F+0,12)0,6,C1004,12
12=CP(+0,7)0,6,C1003,13
13=CU(F+0,13)0,6,C1004,13
```

Even if we do our very best, it's nearly impossible to push two buttons at the same time, so we have to disable the resulting flag setting of these rules in rule 8 and 11 because the program loop will detect that the conditions as specified in rule 3 or 6 will be satisfied before the second button is activated:

Rule 8: If button 6 is down and flag 11 is set (because we were faster with button 6 as with button 7) and button 7 goes down, reset flag 11.

Rule 9: If button 6 is down and button 7 is down, set flag 13.

Rule 10: If flag 13 is set and button 7 is released, reset flag 13. This programming rule acts if button 7 is released before button 6. In that case you would think that rule 3 is back in the game, but that's not true: FSUIPC doesn't react on the status of the "active" key but on the change of his status: "button down" is actually meaning "button goes down", "button up" is actually "button goes up". And because there is no change in the status of button 6, rule 3 is not activated.

Rule 11: If button 7 is down and flag 12 is set (because we were faster with button 7 as with button 6) and button 6 goes down, reset flag 12.

Rule 12: If button 7 is down and button 6 goes down, set flag 13.

Rule 13: If flag 13 is set and button 6 is released, reset flag 13. This programming rule acts if button 6 is released before button 7. Here also the same remark as for rule 10, but now regarding rule 6.

Now these flags can be used to assign real Microsoft Flight Simulator functions to the remaining buttons:

```
; IF -6 AND -7 (Flag 10)
;
14=CR(F+0,10)0,0,C65588,0          ;repeat break
15=CP(F+0,10)0,1,C65570,0          ;toggle gear
16=CR(F+0,10)0,2,C65607,0          ;repeat trim pitch up
17=CR(F+0,10)0,3,C65615,0          ;repeat trim pitch down
18=CP(F+0,10)0,4,C65758,0          ;increment flaps
19=CP(F+0,10)0,5,C65759,0          ;decrement flaps
;
; IF +6 AND -7 (Flag 11)
;
20=CP(F+0,11)0,0,K192,1            ;voice key for CS727
21=CP(F+0,11)0,1,C65751,0          ;toggle landing lights
22=CR(F+0,11)0,2,C65771,0          ;repeat decr. mixture
23=CR(F+0,11)0,3,C65769,0          ;repeat incr. mixture
24=CP(F+0,11)(F+0,20)0,4,C66390,0  ;AND +F20 toggle wing fold (Drag chute on CS F104)
25=CP(F+0,11)(F-0,20)0,4,C66391,0  ;toggle tail hook (Drag chute on CS          Mig21)
26=CP(F+0,11)0,5,C65589,0          ;toggle air break
```

A little more about rule 24 and 25: I am a fan of Captain Sim airplanes, but the CS team uses a different command for the drag chute on the Mig21 as for the drag chute on the Starfighter. I decided to use a flag (which I program later on) to generate a different command for the same button, depending on the status of that flag. Here is another trick that I am also using for the Yak 3 of Captain Sim airplanes: the default animation of the rear-view mirror uses two different commands for the activation and deactivation of the mirror. This is a rather weird because this is a toggle command. The next trick allows me toggle the mirror with one button:

```
36=CP(F+0,13)(F-0,30)0,0,C66294,0          ;Incr Concorde Visor (activate rear-view mirror)
37=CP(F+0,13)(F+0,30)0,0,C66295,0          ;Decr Concorde Visor (deactivate rear-view mirror)
38=CU(F+0,13)0,0,C1005,30                  ;Toggle flag 30
```

The combination of these three rows is used to switch the button command from incr to decr and visa versa, each time the 0 button is activated while button 6 and 7 are both down.

The next button programming rules in the INI file are:

```
; IF (-6 AND +7) = F12
;
27=CP(F+0,12)0,1,C65858,0                  ;toggle pitot-heat
28=CR(F+0,12)0,2,C65777,0                  ;repeat mixture decrement
29=CR(F+0,12)0,3,C65775,0                  ;repeat mixture increment
30=CP(F+0,12)0,4,K83,8                     ;keyboard "S" (next view)
31=CP(F+0,12)0,5,K83,1                     ;keyboard "SHIFT-S" (previous view)
;
; IF(+7 AND +8) = F13
;
32=CP(F+0,13)0,0,C66224,0                  ;autostart engines
33=CP(F+0,13)0,1,C66293,0                  ;toggle avionics on/off
34=CR(F+0,13)0,2,C65880,0                  ;increment heading bug
35=CR(F+0,13)0,3,C65879,0                  ;decrement heading bug
```

By using the combination of three buttons is the following can be accomplished:

```
; Flag 10 follows keys (-5 AND -6 AND -7)
;
0=CU(-0,6)(-0,7)0,5,C1003,10
1=CU(-0,5)(-0,7)0,6,C1003,10
2=CU(-0,5)(-0,6)0,7,C1003,10
;
; Flag 11 follows keys (+5 AND -6 AND -7)
;
3=CP(-0,6)(-0,7)0,5,C1004,10
4=CP(-0,6)(-0,7)0,5,C1003,11
5=CU(F+0,11)0,5,C1004,11
;
; Flag 12 follows key (-5 AND +6 AND -7)
;
6=CP(-0,5)(-0,7)0,6,C1004,10
7=CP(-0,5)(-0,7)0,6,C1003,12
8=CU(F+0,12)0,6,C1004,12
;
; Flag 14 follows key (-5 AND -6 AND +7)
;
9=CP(-0,5)(-0,6)0,7,C1004,10
10=CP(-0,5)(-0,6)0,7,C1003,14
11=CU(F+0,14)0,7,C1004,14
;
; Flag 14 follows key (+5 AND +6 AND -7)
;
12=CP(+0,6)(F+0,12)0,5,C1004,12
13=CP(+0,6)(-0,7)0,5,C1003,13
14=CU(F+0,13)0,5,C1004,13
;
15=CP(+0,5)(F+0,11)0,6,C1004,11
16=CP(+0,5)(-0,7)0,6,C1003,13
17=CU(F+0,13)0,6,C1004,14
;
; Flag 15 follows key (+5 AND -6 AND +7)
;
```

```
18=CP(+0,7)(F+0,13)0,5,C1004,13
19=CP(+0,7)(-0,6)0,5,C1003,15
20=CU(F+0,15)0,5,C1004,15
;
21=CP(+0,5)(F+0,13)0,7,C1004,11
22=CP(+0,5)(-0,6)0,7,C1003,15
23=CU(F+0,15)0,7,C1004,15
;
; Flag 16 follows key (-5 AND +6 AND +7)
;
18=CP(+0,6)(F+0,12)0,7,C1004,12
19=CP(+0,6)(-0,5)0,7,C1003,16
20=CU(F+0,16)0,7,C1004,16
;
21=CP(+0,7)(F+0,13)0,6,C1004,13
22=CP(+0,7)(-0,5)0,6,C1003,16
23=CU(F+0,16)0,6,C1004,16
```

Appendix 2: About the *Aircraft Specific* option and "*ShortAircraftNameOK*"

I have left this Appendix in for reference as I think it is still useful, but please note that this ini parameter is no longer available, and FSUIPC7 automatically uses *ShortAircraftNameOK=Substring* – this cannot be changed.

Note: this is a contribution from a user, to whom thanks is expressed.

There are these three choices in FSUIPC settings:

> **ShortAircraftNameOK=No**
>
> **ShortAircraftNameOK=Yes**
>
> **ShortAircraftNameOK=Substring**

Result: To get exactly the same settings for AXES, BUTTONS, KEYS and CALIBRATION for each plane repaint or variant.

The Short Aircraft Name in FSUIPC refers to the name in the Aircraft.cfg file under "title"

For example: Aerosoft DHC Beaver. There might be 7 variants or repaints

> aircraft.cfg \(flightsim.X)\title= Aerosoft Beaver DHC-2A 55-0682
> aircraft.cfg \(flightsim.X)\title=DHC-2A C-GSKY Beaver
> aircraft.cfg \(flightsim.X)\title= Aerosoft DHC-2A C-GSKY modern
> aircraft.cfg \(flightsim.X)\title=Beaver DHC-2A DQ-GEE
> aircraft.cfg \(flightsim.X)\title=DHC-2A DQ-GEE modern
> aircraft.cfg \(flightsim.X)\title= Aerosoft DHC-2A N299EE
> aircraft.cfg \(flightsim.X)\title=Beaver Aerosoft DHC-2A N299EE modern

Edit the FSUIPC.ini file:

Scenario 1: If "ShortAircraftNameOK=No"

Presuming that you have already assigned the axes, keys and buttons and calibrated the joystick for one of the above variants or repaints: in order to get the same settings for the rest of the above variants/repaints of the Aerosoft Beaver you would need to edit the FSUIPC.ini file and add 4 separate entries for each title name (exactly as above) under [Axes], [Buttons], [Keys], [Joystick Calibration] to ensure that all of the settings were exactly the same, ie 28 entries in all. Pretty tedious in fact— I had over 40 variants/repaints of this plane so I would have need 160 entries in the FSUIPC.ini file.

> [Axes. Aerosoft Beaver DHC-2A 55-068]
> [Buttons. Aerosoft Beaver DHC-2A 55-068]
> [Keys. Aerosoft Beaver DHC-2A 55-068]
> [JoystickCalibration.Aerosoft Beaver DHC-2A 55-068]
>
> [Axes. DHC-2A C-GSKY Beaver]
> [Buttons. DHC-2A C-GSKY Beaver]
> [Keys. DHC-2A C-GSKY Beaver]
> [JoystickCalibration.DHC-2A C-GSKY Beaver]
>
> [Axes. Aerosoft DHC-2A C-GSKY modern]
> [Buttons. Aerosoft DHC-2A C-GSKY modern]
> [Keys. Aerosoft DHC-2A C-GSKY modern]
> [JoystickCalibration.Aerosoft DHC-2A C-GSKY modern]
>
> [Axes. Beaver DHC-2A DQ-GEE]
> [Buttons. Beaver DHC-2A DQ-GEE]

[Keys. Beaver DHC-2A DQ-GEE] [JoystickCalibration.Beaver DHC-2A DQ-GEE]
[Axes. DHC-2A DQ-GEE modern] [Buttons. DHC-2A DQ-GEE modern] [Keys. DHC-2A DQ-GEE modern] [JoystickCalibration.DHC-2A DQ-GEE modern]
[Axes. Aerosoft DHC-2A N299EE] [Buttons. Aerosoft DHC-2A N299EE] [Keys. Aerosoft DHC-2A N299EE] [JoystickCalibration. Aerosoft DHC-2A N299EE]]
[Axes. Beaver Aerosoft DHC-2A N299EE modern] [Buttons. Beaver AerosoftDHC-2A N299EE modern] [Keys. Beaver Aerosoft DHC-2A N299EE modern] [JoystickCalibration.Beaver Aerosoft DHC-2A N299EE modern]

Scenario 2: If "ShortAircraftNameOK=YES"

12 entries would be required to make sure all settings were the same

[Axes. Aerosoft [Buttons. Aerosoft [Keys. Aerosoft [JoystickCalibration.Aerosoft]	[Axes. DHC] [Buttons. DHC] [Keys.DHC] [JoystickCalibration.DHC]	[Axes. Beaver] [Buttons. Beaver] [Keys.Beaver] [JoystickCalibration.Beaver]

Explanation:

1. "Aerosoft" would pick all those entries in the title STARTING with "AEROSOFT", but NOT Aerosoft in any other part of the title.

2. "DHC" would pick all those entries in the title STARTING with "DHC" but not those with "DHC" in any other part of the title

3. "Beaver" would pick all those entries in the title STARTING with "Beaver" but not those with "Beaver" in any other part of the title

Scenario 3: If "ShortAircraftNameOK=Substring"

4 entries only, i.e. "DHC" in the FSUIPC.ini file would result in all variants having exactly the same settings – "DHC" is common to all titles.

[Axes. DHC] [Buttons. DHC] [Keys. DHC] [JoystickCalibration.DHC]

To summarise:

ShortAircraftNameOK=No	**One entry for each different title in the aircraft.cfg file**
ShortAircraftNameOK=Yes	**Picks up the starting part of the title in the aircraft.cfg file**
ShortAircraftNameOK=Substring	**Picks up any part of the title in the aircraft.cfg file**

Title in aircraft.cfg file	ShortAircraftNameOK=		
	No	**Yes**	**Substring**
title=Airbus A321 title=Airbus A321 Paint2 title=Airbus A321 Paint4 title=Airbus A321 Paint5 title=Boeing 737-400 title=Boeing 737-400 Paint1 title=Boeing 737-400 Paint2 title=Boeing 737-400 Paint3 title=Boeing 737-400 Paint4 title=Boeing 747-400 title=Boeing 747-400 Paint1 title=Boeing 747-400 Paint2 title=Boeing 747-400 Paint3 title=Boeing 777-300 title=Boeing 777-300 Paint1 title=Boeing 777-300 Paint2 title=Boeing 777-300 Paint 3	**Separate entry for each title**	**"Airbus": Would apply to all entries starting with Airbus.** **"Boeing" would apply to all entries starting with Boeing.**	**"A321": Any variant with A321 in the title.** **"Paint" Any variant with PAINT in the title.** **"737": Any variant with 737 in the title.**

Explanation: **ShortAircraftNameOK=Substring** Any text that is in any position in the "title" located in the aircraft.cfg file that is inserted in the ini file as above will result in the same settings for those aircraft. For instance choosing "737" ie **[Axes.737]** etc would result in all planes with 737 in the title having the same settings. Likewise choosing "Boeing" would cover all variants/repaints with Boeing in the title

To summarise if you had 20 variants/models/repaints with all different titles you would need 20 entries per section (80 in all) in the ini file. Using **ShortAircraftNameOK=Substring** you could cut this back to just 1 entry per section (4 in total).

APPENDIX 3: Handling VRInsight serial devices in FSUIPC

Introduction

First, please note that this section is all about FSUIPC's support for these *serial port* devices from VRInsight:

> "MCP Combi", "M-Panel", "CDU 2", "CDU", "Micro Prop Pit", "Micro Jet Pit", "Radio Stack", "Prop Pit", "Jet Pit", "Flight Monitor", "MFD", "GPS5", "MCP2 Boeing", "MCP2 Airbus"

These have become quite popular, being pretty good value for money. You can get a lot of functionality in a compact package. However, they are not recognised by Windows as "Human Interface Devices" (HIDs) and certainly not as "joysticks", and are therefore not normally seen in FSUIPC for Button or Switch programming.

In fact they are serial "COM" port devices, using USB connections with an FTDI chip based interface with a serial/USB port driver. Their interface to FS is managed by VRInsight's own driver "**SerialFP2**" (or a later replacement).

For many straightforward uses, SerialFP2 does a good job. However, it doesn't provide the flexibility for every purpose and with more and more specialised aircraft and other add-ons for FS doing their "own thing", a way to increase the functionality of the VRI devices was felt needed. This is especially the case where the devices have to resort to sending many keypress combinations, which can get rather fraught when so many other programs are also doing this.

The opportunity to make provisions in FSUIPC for the VRInsight range arose after the implementation of the serial port handling Lua library, "**com**", because now FSUIPC already contained a multi-device multi-threaded mechanism for easily reading from, and writing to, serial COM port devices.

Problems and Solutions

Compared to the GoFlight implementation in FSUIPC, which utilises a library module (GFDev.dll) provided by GoFlight for this purpose, there are some complications. With GoFlight the devices can, to some extent, be shared between the GoFlight driver assignments and FSUIPC assignments (though admittedly this can provide complications with displays and indicators). The VRInsight situation is rather different. There is no easy way for a user-level program like FS+FSUIPC to share the use of the same COM port with the VRInsight driver (SerialFP2). It could probably be done using something like the Eterlogic VSPE program as a "splitter", but this is not a general solution for users.

Therefore it first looked like it would have to be an either/or: you either use SerialFP2, or you use FSUIPC probably with a Lua plug-in to program the displays. That would means the plug-in must do a lot of work, much of it probably beyond the means of most users.

However, the Eterlogic VSPE ("Virtual Serial Port Emulator") does offer a good solution. It can provide any number of virtual serial port "Pairs": that is two 'pretend' COM ports which are linked. For example, COM9 and COM10 might be a "Pair". Whatever a program writes to COM9 can be read by another program on COM10 and vice versa. This is a facility I already promote the use of with my GPSout links via WideFS, for moving map applications.

The use of Virtual Serial Port pairs allows FSUIPC to sit between the VRInsight device and the VRInsight driver (SerialFP2). Then FSUIPC can divert some or all buttons and switches to uses determined in the FSUIPC options, and it can provide optional Lua plug-ins with opportunities to hook into both switch inputs from and display outputs to the devices.

Okay. So, if you are still interested, let's move on to the instructions for achieving this:

Setting up the virtual serial ports

First, please download the Eterlogic VSPE program:

<div align="center">

http://www.eterlogic.com/Products.VSPE.html

</div>

You will need to purchase a key for this program. ($25 U.S.). After installing it and registering it (you have to cut and paste the long key!), proceed as follows:

1. From the **Device** menu, select **Create**.

2. In the **Device Type** drop-down, select **Pair**, then press **Next** and **Finish**.

3. Repeat steps 1 and 2 for the number of VRI devices you want to connect this way.

4. Note down the pairs made. For example:

<p style="text-align:center">COM5 ⇔ COM6
COM7 ⇔ COM8</p>

5. In the **File** menu, select **Save As**, and save the configuration to some place with a file name you will know. For example, in **C:** with a name like

<p style="text-align:center">**ComPairs_56_78**</p>

to suit the configuration example I gave above.

6. Now close VSPE. By default the pairs will be destroyed. That's fine.

7. Find the short-cut to VSPE which the installer placed on your desktop. Right-click it, select Properties, then at the end of the stuff in "Target", and after a space add:

<p style="text-align:center">**-minimize –hide_splash C:\ComPairs_56_78.vspe**</p>

where you put your own path and configuration filename in place of '**C:\ComPairs_56_78**'

8. Now you have a choice. You can have this program start when Windows starts—just drag the short-cut, or a copy of it, into the Windows **Startup** folder. That's what I would do. The existence of all those extra COM ports does no harm when you are not using them, and you will be annoyed if you forget to start the program before you want to run FS.

Note that you must *not* start it by simply using a **Run** parameter in FSUIPC7 INI's **[Programs]** section. This will be too late for FSUIPC to open one end of the link for SerialFP2 to connect.

Configuring FSUIPC to handle VRI devices

Now we must edit the **FSUIPC7 INI** file. Find it in the FSUIPC installation folder—if you have Windows set to hide known filetypes it will look like just '**FSUIPC7**' with a file type of "**Configuration Settings**". Load it into a text editor such as NotePad—do *not* use WordPad or a word processor!

Add a completely new section:

```
[VRInsight]
1=<device>, <driver>
2=<device>, <driver>
```

Where those **<device>** and **<driver>** entries are serial port names. You need one line here for each VRI device. The order doesn't matter. The **<device>** entry gives the real serial port name for the device, and the **<driver>** entry gives a virtual serial port name.

You can assign any virtual pair to any device, but just one pair to one device. Then, for each device, you enter one of the pair's port names as **<driver>** here. The other one of the pair will be used by SerialFP2—you shouldn't need to worry about that if SerialFP2 is set to 'Auto', as it will find it.

As an example, supposing I have one VRI device on **COM3** and another on **COM9**. With my two pairs as set in the example on the previous page I could have:

<p style="text-align:center">1=**COM3, COM6**
2=**COM9, COM8**</p>

Then SerialFP2 would connect to the first via **COM5** and the second via **COM7**. Here are the connections which will be made:

<p style="text-align:center">**SerialFP2 ← → COM5 ← → COM6 ← → FSUIPC ← → COM3 ← → VRI Device1**</p>

<p style="text-align:center">**SerialFP2 ← → COM7 ← → COM8 ← → FSUIPC ← → COM9 ← → VRI Device2**</p>

Note that, if you didn't need SerialFP2 to drive your device, if it only had buttons and switches you were assigning in FSUIPC, or you were driving it with a Lua plug-in instead of SerialFP2, then you need not have a 'Pair' for it and you would omit the second port in the [VRInsight] parameters. I don't think this is likely to apply very often.

Running SerialFP2

Whilst you are editing the FSUIPC7.INI file, you should consider how you will be running SerialFP2. It must *not* be run *before* FSUIPC has grabbed the device's real port, or it will get it and prevent FSUIPC's access. Running it manually after starting FS is awkward for obvious reasons.

The best way is to run it from FSUIPC. For that you need it adding to the INI file's [Programs] section (add the section too if you haven't got one). For example, for two devices I would have this:

```
[Programs]
Run1=READY,CLOSE,d:\VRInsight\SerialFP2\SerialFP2.exe
Run2=READY,CLOSE,d:\VRInsight\SerialFP2\SerialFP2.exe
```

For two devices you need two copies of SerialFP2 running, and so on. By putting 'READY' here I am stopping it running before FSUIPC has got the port. CLOSE simply asks FSUIPC to close it when FS closes.

One final thing. Until you are sure you have things right, you might want to enable some special Logging in FSUIPC which will show what is going on in the SerialFP2 – FSUIPC -- VRI device chain. To log of all of the inputs and outputs, from all parties, you can use the **Log → Custom** feature, and enter the value **4**.

Okay. Now you should be ready. Make sure your VRI devices are switched on, then run FS.

If all goes well your VRI devices should initialise and start working normally. The FSUIPC Log file will, soon after the initialisation phase, show entries like this:

```
VRI port 1 "COM5" opened
VRI driver port 1 "COM2" also opened
```

For each pair listed in the [VRInsight] section of the FSUIPC7.INI file, and then, as each device is seen by the SerialFP2 driver (though probably getting mingled, as they are all multi-threading):

```
VRI COM2 ---> CMDRST [from VRI Driver]
VRI COM5 <--- CMDRST [to Device]
VRI COM2 ---> CMDCON [from VRI Driver]
VRI COM5 <--- CMDCON [to Device
VRI COM5 ---> CMDCON [from Device]
VRI COM2 <--- CMDCON [to VRI Driver]
VRI COM5 ---> APLMAST+ [from Device]
VRI COM2 <--- APLMAST+ [to VRI Driver]
VRI COM2 ---> CMDFUN [from VRI Driver]
VRI COM5 <--- CMDFUN [to Device]
VRI COM5 ---> CMDFMER [from Device]
VRI FMER ("MCP Combi") detected on port COM5
VRI COM2 <--- CMDFMER [to VRI Driver]
```

Note that FSUIPC here recognized "FMER" as being the MCP Combi.

If the SerialFP2 driver does not find the device it may need helping. Try setting it on 'AUTO' and making it retry. Once you have it working it sohuld be fine next time.

Programming buttons, switches and knobs

Once you've reached this stage you should find you can detect and program most of the VRInsight knobs and switches within FSUIPC's Buttons and Switches Tab. They'll have joystick numbers 256 and over. Some dials will look like 4 buttons—fast and slow in each direction. But some don't have the fast mode.

FSUIPC's Buttons tab only reacts to buttons when they switch from "off" to "on". For VRInsight devices this generally means two presses on buttons—unlike normal joystick buttons, holding them down does nothing useful. There's no indication available of this. You press and release for one indication, then do the same again for the next. Each time you do this it changes the button state from "off" to "on" and vice versa, alternately. If you want a button to do something every time you press it you need to program both the press and the release. Similar considerations usually apply to dials, which look "on" on one click and "off" on the next, and so on.

At present the radio buttons and knobs are not programmable in FSUIPC. They seem to operate quite well enough as they are. They will be overridable in Lua plug-ins, for those among you who wish to get into more advanced manipulation of the devices, but they aren't suitable for general re-allocation.

Once a button, knob or switch is programmed in FSUIPC it is hidden from SerialFP2 and, in fact, the log.

What else? What about the displays?

Good questions.

Everything that SerialFP2 can do with a device can also be done with a Lua plug-in using the facilities offered by the new "**com**" library and, with the aid of some extra parameters which go into the FSUIPC7.INI file, this can work with SerialFP2 taking its part too if you'd rather not have to re-program everything yourself.

The Lua package provided with FSUIPC contains full details of both the Lua programming side and how the FSUIPC INI file can be edited to make this all run seamlessly and automatically. Two relatively simple examples are included and explained:

- one to allow the MCP Combi Speed display and adjustment to work correctly in Mach mode as well as IAS mode, and

- one to swap the use of Inches for the altimeter BARO setting on the M-Panel for millibars (or hectoPascals if you prefer).

If you own the MCP-Combi or M-Panel devices you might want to try one of those now. Instructions are included in the Lua ZIP package.

APPENDIX 4: Running FSUIPC7 on an FS Client PC

It is now possible to run FSUIPC7 (v7.2.13 and later) on a client PC, that is, a 2nd PC where MSFS is not installed. This should allow FSUIPC clients to be ran on the client PC and allow them to communicate to the FS without the need for WideFS/WideClient. Note that this DOES NOT replace or duplicate the functionality provided by WideFS. When FSUIPC7 runs on a client PC, it will maintain its own offset area distinct from that of the offset area used by FSUIPC7 on the FS machine. However, most offsets should contain the same data, as they are populated by the data received from the FS, but be aware of this if/when writing to user offsets (e.g. using lua).

Note also that you do not have to have FSUIPC7 running on the FS machine to run FSUIPC7 in the client machine.

The following functionality in FSUIPC7 will not work (or is not available) in FSUIPC7 when ran on a client machine:
- no installer: you need to manually install/copy FSUIPC7 to the client machine
- no possibility of sending key presses to the FS (key presses CAN be received)
- no auto-start or auto-connection

To have FSUIPC7 running on a client machine, please follow the steps indicated below.

1. Configuring SimConnect on the MSFS computer
Locate your MSFS2020 **SimConnect.xml** file. For steam installs, this should be under
Your User Account\AppData\Roaming\Microsoft Flight Simulator
and for MS Store installs under:
Your User Account\Local\Packages\Microsoft.FlightSimulator_8wekyb3d8bbwe\LocalCache

Open the file in an editor, and add a Global (remote) IPv4 port by adding the following entry, replacing `'Your Local MSFS PC Address'` with the actual IP address:

```
<SimConnect.Comm>
        <Descr>Global IP Port</Descr>
        <Disabled>False</Disabled>
        <Protocol>IPv4</Protocol>
        <Scope>global</Scope>
        <Address>Your Local MSFS PC Address</Address>
        <MaxClients>64</MaxClients>
        <Port>7421</Port>
        <MaxRecvSize>4096</MaxRecvSize>
        <DisableNagle>False</DisableNagle>
</SimConnect.Comm>
```

You can also change the port number, but this MUST match the port number defined in the **SimConnect.cfg** in your client PC (see below). Note that you also may need to open this port in any firewalls that you may use (although I did not have to do this when I configured in my LAN).

2. Install FSUIPC7 in the client machine
To install FSUIPC7 on a client machine, create an FSUIPC7 folder (preferably not under Documents, Program Files, or any other windows-protected folder) and copy across the following files from your MSFS computer:
FSUIPC7.exe
FSUIPC7.key (if using a registered version)
You can also copy across your FSUIPC7.ini file, but it may be better to start afresh with this in your client PC.

3. Configuring SimConnect on the Client computer

To configure SimConnect on the client PC, create a file called **SimConnect.cfg** in your FSUIPC7 installation folder on the client machine with the following contents:

```
[SimConnect]
Protocol=IPv4
Address=Your Local MSFS PC Address
Port=7421
MaxReceiveSize=4096
DisableNagle=0
```

where **Your Local MSFS PC Address** is the same address as used in the client configuration (and the port also must, of course, be the same as the one used there).

4. Configure FSUIPC7 to run on the Client computer

If you have copied across your FSUIPC7.ini, open this in an editor and removed any unwanted assignments/profiles. If you have not copied across your FSUIPC7.ini, run FSUIPC7 once and exit - this will create a default FSUIPC7.ini file for you.

Open this file in an editor (e.g. Notepad++). Under the [General] section, add the following ini parameters:

       RunningOnClientPC=Yes
       UseSimConnection=0

and under the [WAPI] section (if using the FSUIPC7 WASM module on the MSFS computer) - create if necessary, also add the following:

       UseSimConnection=0

And that is all!

To use FSUIPC7 on the client machine, you must manually connect once MSFS is up and running.