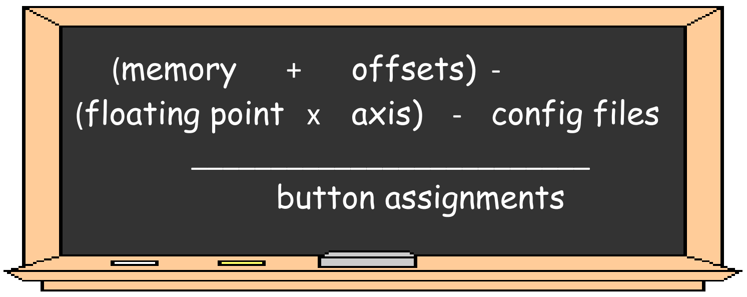
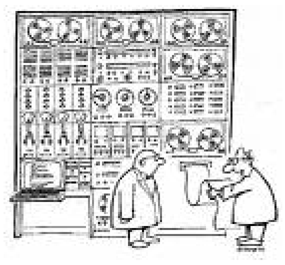
So what is FSUIPC and these Offset thingys ?

So you've got no programming skills and you've only just discovered that your PC has a calculator... but you're not stupid.. after all you went to school, you know how an engine works and heck you even know how a jet engine works, roughly !

But as you start to read all of the documentation that you've found on the net for Flight sim and Interfacing, it all starts to blur and sound like the most complicated thing on the planet – ever.



=



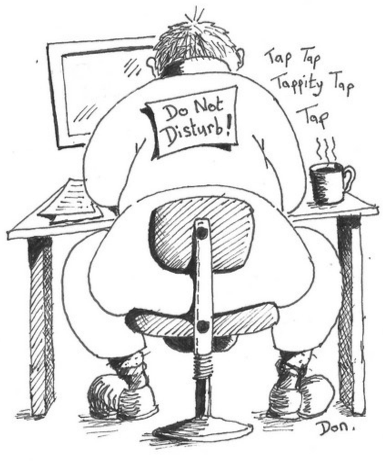
eventually it all starts looking like *"mem #^:,floating*%:, offset @:] blah blah blah"* !!

So I'll stop you there.. and try to clear a few things up.

In the beginning...

So you've heard of **FSUIPC** and people keep on mentioning 'offsets', but what are they and what do you need to know about them ?

FSUIPC is a utility developed by Pete Dowson and originally created by Adam Szofran in the form of FS6IPC. In simple terms FSUIPC allows external applications to access many of the Microsoft Flight Simulator internal variables from simple data like pitch, roll, altitude and heading right through to weather and acceleration values.



In a Nutshell

The FSUIPC module reserves a 65535 byte block of memory and then populates part of this memory with various pieces of data extracted from FS. External applications then have read and write access to the data in this block of memory for their own use.

You need to know about offsets and memory if you are going to be interfacing with Flight Sim via your only application or maybe via some I/O interfacing software or perhaps you want to start enhancing your sim setup by adding a few extra buttons.

A flat pack 'Offset' is born

“Each piece of FS data is placed in the memory block at a certain location. This is it's '**offset**'.”

You can think of the memory block as a huge empty shelf.

...... And this shelf is divided into 65535 compartments.

................And each compartment represents a single byte.

FSUIPC then takes individual pieces of flight sim data and places then in separate compartments in the memory block.

So put in simple language it's like saying :
“I've put the landing light data in compartment **42**”

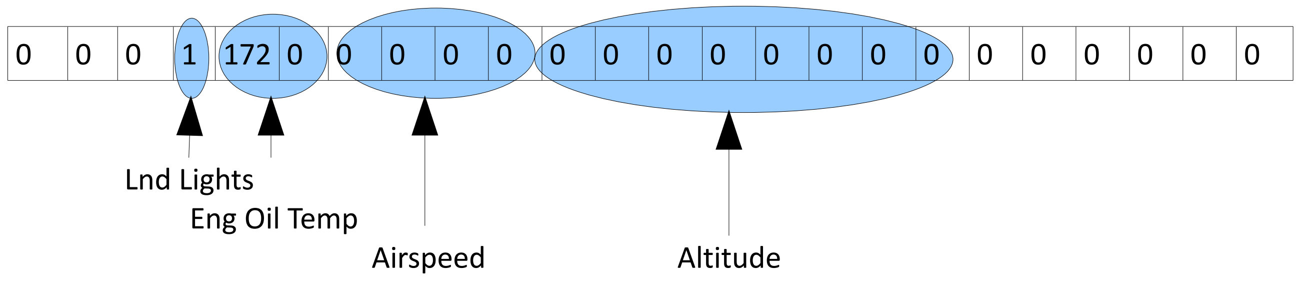
If you wanted to read the landing light data you would read the value in compartment **42**.

Offsets are usually specified in Hex format :
byte **42** in decimal = Offset **002A** in Hex

Written down in a table it might look like this:

Description	**Offset**	**Size**
Landing Lights	002A	1
Engine Oil Temp	002B	2
Airspeed	002D	4
Altitude	0031	8

And this is how the memory block would look.
Byte 40   41   42.....



For those wanting to read the values from the memory block, you would normally want to read the data into the same sized variable as it was initially written.
By this I mean that you would read an 8 byte altitude value into an internal variable defined as 8 bytes. Otherwise you will not have all of the data bytes need to correctly format the value, and the value that you read will be incorrect.

The bottom line is :

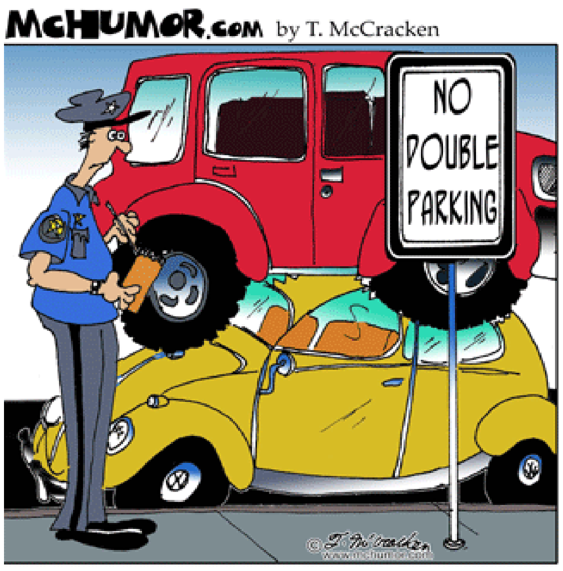
FSUIPC stores data from Flight Simulator in a memory block.
Each piece of data is assign a number of bytes depending on the size of the value that needs to be stored.
The starting location of the data is noted and known as it's 'offset' in the memory block.

This memory block is then made available to external applications. WideClient also sends a copy of this memory block across your network to other client pc's.

Which Offsets can I use ?

All of the memory values that are extracted from Flight Sim and populated by FSUIPC are listed in the FSUIPC SDK.
The usage of each offset is clearly stated in the documentation.

- Some FSUIPC offsets are 'read only' and cannot be written to.
- Some offsets can be written to but not intended for this purpose and your data will be overwritten during the next FSUIPC data load.
- Some offsets are read/write and you can write a value to these offsets to modify a function in FS.



Therefore you should only used the offsets publisged in the FSUIPC SDK as documented or you might get some unexpected results.
Othwise you can use virtually any spare offset in the range.

<u>But Remember.....</u>

If your application writes data to an offset that is already being populated by FSUIPC or one of your other FS addons it will probably get overwritten.

How many offsets do I require ?

If you are reading values from FSUIPC then you will need to reference the documentation for the product that is populating that offset to understand how many bytes to read.

If you want to use FSUIPC to store your own value you will need to decide how many bytes it requires.

Ask yourself - “Do I need to store...
a large number, negative number or floating point 'decimal' number ?”

For info:

- ***1 byte** can store a number between 0 and 255 (this could also be an ascii char)*
- ***2 bytes** (known as a WORD or SMALLINT) can store a number between 0-65535 unsigned or -32767 – 32767 signed*
- ***4 bytes** (known as a DWORD or INTEGER or FLOAT32 or SINGLE depending on the usage) can store a whole number between 0-4294967296 or decimal values*
- ***8 bytes** (known as a FLOAT64 or DOUBLE) can hold large decimal values.*
- *If you need to store a string character this is held as it's ASCII character code 0-255, therefore a single character is stored in a byte.*

Sim-Avionics Specific Information

As a base for Sim-Avionics offsets I usually recommend starting with the 5300-53FF range. These 255 bytes have been 'reserved' in FSUIPC for this project and no other official application should be using them. - *But it doesn't really matter which offsets you use as long as there are no conflicts.*

Most current interface solutions are able to read/write with FSUIPC, so a typical setup to map a battery switch might be:

Step 1:	Press a button in your sim
Step 2:	This is detected by your I/O card and it's software writes a '1' to a specified offset.
Step 3:	In the Sim-Avionics FSUIPC_IO.ini file you have assigned the specified offset to the Battery Switch.
Step 4:	The Server reads the specified offset and enables the Battery Switch.

Assigning Switch Inputs

There are 2 methods of using FSUIPC to control system switches in the server.

The first is the **Direct Offset** method. Meaning each function monitors a specific offset.

To assign the offsets to the BATTERY switch and ADIRU switch you must edit the FSUIPC_IO.ini

```
[FSUIPC_INPUTS]
BATTERY_SWITCH=5320
ADIRU_SWITCH=5321
```

This tells the server to monitor offsets 5320 and 5321 for the Battery and ADIRU switch positions. A value of 0 = Switch OFF and 1 = Switch ON.

The second method is via a **MultiFunction Offset** this is also assigned in the FSUIPC_IO.ini.

```
[FSUIPC_MULTI_FUNCTION]
MULTI_FUNCTION=53FE
```

This time instead of assigning a different offset for every function you just assign the MultiFunction offset. (2 byte WORD)

The server then monitors this offset and functions are trigger depending on the value set at this offset.

704 = BATTERY SWITCH = OFF
705 = BATTERY SWITCH = ON
722 = ADIRU SWITCH = OFF
723 = ADIRU SWITCH = ON

Once the function has been trigger the server sets the MultiFunction offset back to a value of 0 ready for the next command.

A full Multi-Function list can be found in our User Manual

Assigning Outputs

Just like the switch inputs an FSUIPC offset can be assigned to an output function.
This is again made in the FSUIPC_IO.ini file.

After the FSUIPC_INPUT section there is an FSUIPC_OUTPUT section. And it's here you can assign an offset to a function.

Please note: at this stage the output value represents the status of assigned system and not the overhead light state. To clarify, on an aircraft overhead, a light usually illuminates when a system is inoperative or there is a failure.

In this example offset 53C0 will = 1 when the battery system is ON and = 0 when the battery system is OFF.

```
[FSUIPC_OUTPUTS]
BATTERY=53C0
```

This is likely to be the opposite of what you want your battery 'light' to do on your overhead panel.

Therefore you can either allow for this in the code for your interface solution or use some of the additional assignment options in the FSUIPC_IO.ini file.

```
// Item = offset $invert (1 or 0) b(bit 0-7)
// BATTERY=53C0                  (if Battery = ON then offset 53C0 = 1)
// BATTERY=53C0$1                (if Battery = OFF then offset 53C0 = 1)
// BATTERY=53C0b2                (if Battery = ON then offset 53C0 = 4)  (bit 2)
// BATTERY=53C0b00000100         (if Battery = ON then offset 53C0 = 4)  (bit 2)
// BATTERY=53C0$1b3              (if Battery = OFF then offset 53C0 = 8) (bit 3)
// BATTERY=53C0$1b00001000       (if Battery = OFF then offset 53C0 = 8) (bit 3)
```

```
[FSUIPC_OUTPUTS]
BATTERY=53C0$1
```

This would invert the offset value. Now 53C0 will = 0 when the battery system is ON and = 1 when the battery system is OFF.

In your interface solution you can now say something like :

```
if 53C0 = 1
Battery_Light = ON
else
Battery_Light = OFF
```

Don't Panic!

Using Offset - “bits”

A 'BIT' is a single number or a 'virtual' switch. It can either be OFF = **0** or ON = **1**
A 'BYTE' contains 8 bits **00000000**

In normal decimal notation we count in 10's. Each column is a multiple of 10 and each digit can by 0-9 As a number gets more zeros added to it we multiply by 10, so... One, Ten, One Hundred, One Thousand etc

Binary work slightly different as a bit can only have a value of 0 or 1.
As a bit moves left in the 'byte' it's value doubles, then just like in decimal you add up all of the 'ON' bits to calculate it's decimal value.

Value	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	**1**
2	0	0	0	0	0	0	**1**	0
3	0	0	0	0	0	0	**1**	**1**
13	0	0	0	0	**1**	**1**	0	**1**
75	0	**1**	0	0	**1**	0	**1**	**1**
255	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

so what does this mean ?

A single byte or fsuipc offset can hold a value of 0-255, so if you want to hold a value like a battery voltage 0-24v you would need to use the whole byte to store this value, but if you want to hold basic ON/OFF states for switches you 'can' assign each of the 8 'bits' separately.

The result is exactly the same as assigning a whole byte to a function.
An advantage is that you require less offsets.

To assign an offset bit to a function you use the additional assignment option **'b'.**
'bits' are numbered from ***right to left*** **0-7**

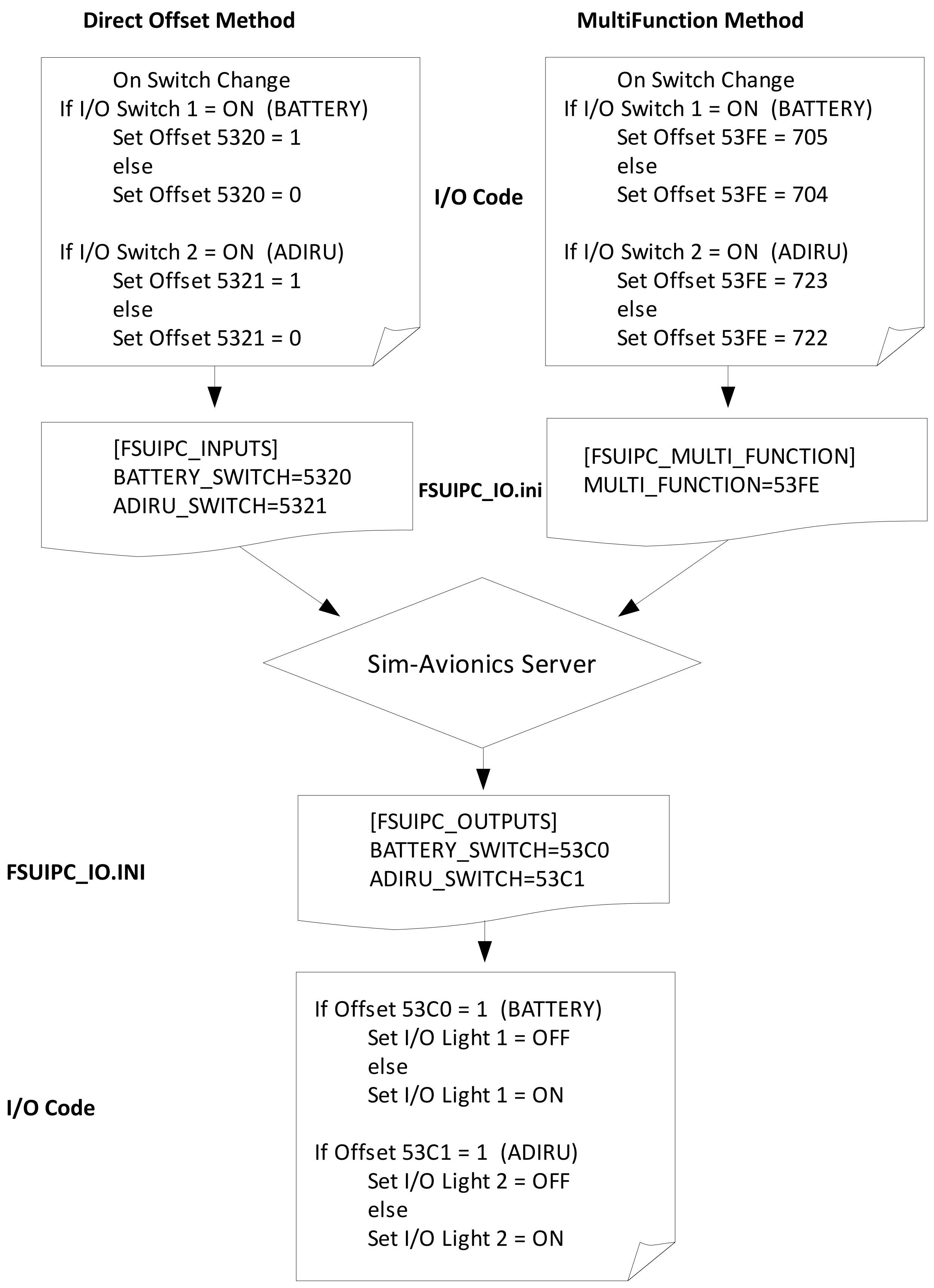
```
[FSUIPC_INPUT]
BATTERY_SWITCH=5320b0
ADIRU_SWITCH=5320b4
```

This example will set the battery switch when bit 0 = 1 and the ADIRU switch when bit 4 = 1

This is how the offset value will look as the two switches are changed:

Both OFF	**00000000** = 0 in decimal
Battery ON	**00000001** = 1 in decimal
ADIRU ON	**00010000** = 16 in decimal
Both ON	**00010001** = 17 in decimal

In Summary



```
        On Switch Change
If I/O Switch 1 = ON  (BATTERY)
        Set Offset 5320 = 1
        else
        Set Offset 5320 = 0

If I/O Switch 2 = ON  (ADIRU)
        Set Offset 5321 = 1
        else
        Set Offset 5321 = 0
```

```
        On Switch Change
If I/O Switch 1 = ON  (BATTERY)
        Set Offset 53FE = 705
        else
        Set Offset 53FE = 704

If I/O Switch 2 = ON  (ADIRU)
        Set Offset 53FE = 723
        else
        Set Offset 53FE = 722
```

```
[FSUIPC_INPUTS]
BATTERY_SWITCH=5320
ADIRU_SWITCH=5321
```

```
[FSUIPC_MULTI_FUNCTION]
MULTI_FUNCTION=53FE
```

```
[FSUIPC_OUTPUTS]
BATTERY_SWITCH=53C0
ADIRU_SWITCH=53C1
```

```
If Offset 53C0 = 1  (BATTERY)
        Set I/O Light 1 = OFF
        else
        Set I/O Light 1 = ON

If Offset 53C1 = 1  (ADIRU)
        Set I/O Light 2 = OFF
        else
        Set I/O Light 2 = ON
```

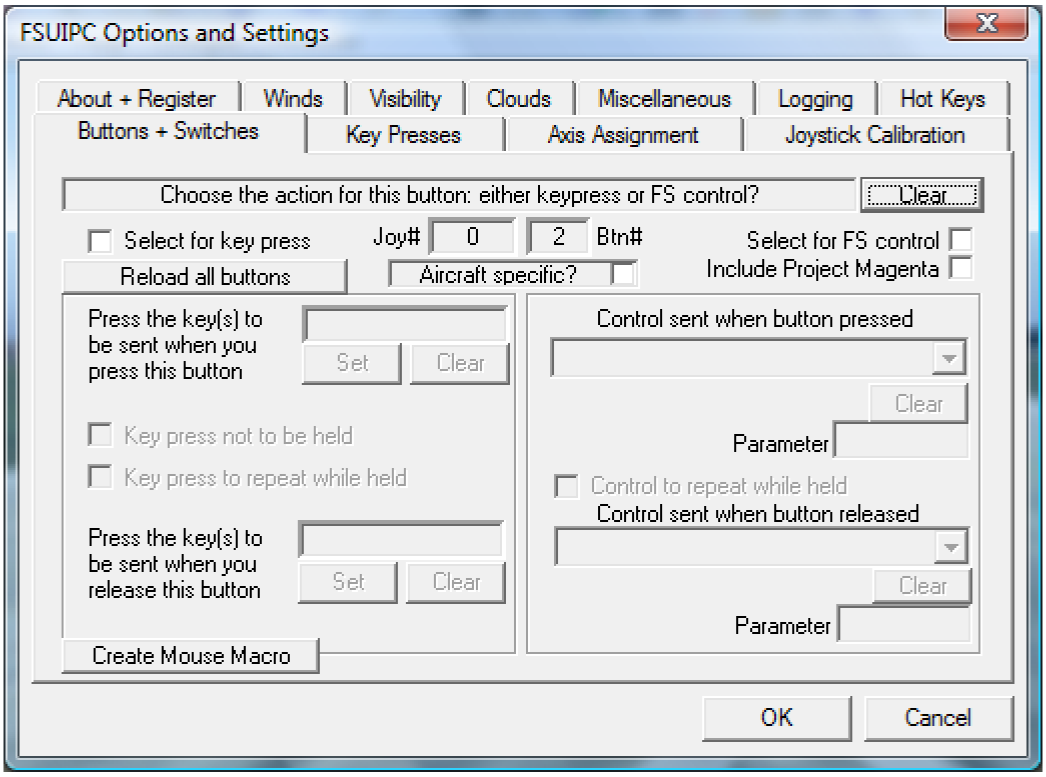
Using FSUIPC within FS to assign offset values to a Joystick Button

If you have dedicated hardware controller software like InterfaceIT from FDS or SIOC then you will assign functions from their own software. But if you have an interface that emulates joystick buttons or want to assign you current joystick buttons to functions then you can use the Buttons + Switches utility within the FSUIPC module.

To do this:
Start FS and under the Modules menu select FSUIPC.
Then select the "Buttons + Switches" tab.

Next 'press' the Joystick button you wish to assign to a function.
FSUIPC should detect the button press and display the Joy# and Btn# in the window.

In this example I have pressed button 2 on the joystick 0.



FSUIPC Options and Settings

About + Register | Winds | Visibility | Clouds | Miscellaneous | Logging | Hot Keys
Buttons + Switches | Key Presses | Axis Assignment | Joystick Calibration

Choose the action for this button: either keypress or FS control? [Clear]

☐ Select for key press   Joy# 0   2 Btn#   Select for FS control ☐
[Reload all buttons]   Aircraft specific? ☐   Include Project Magenta ☐

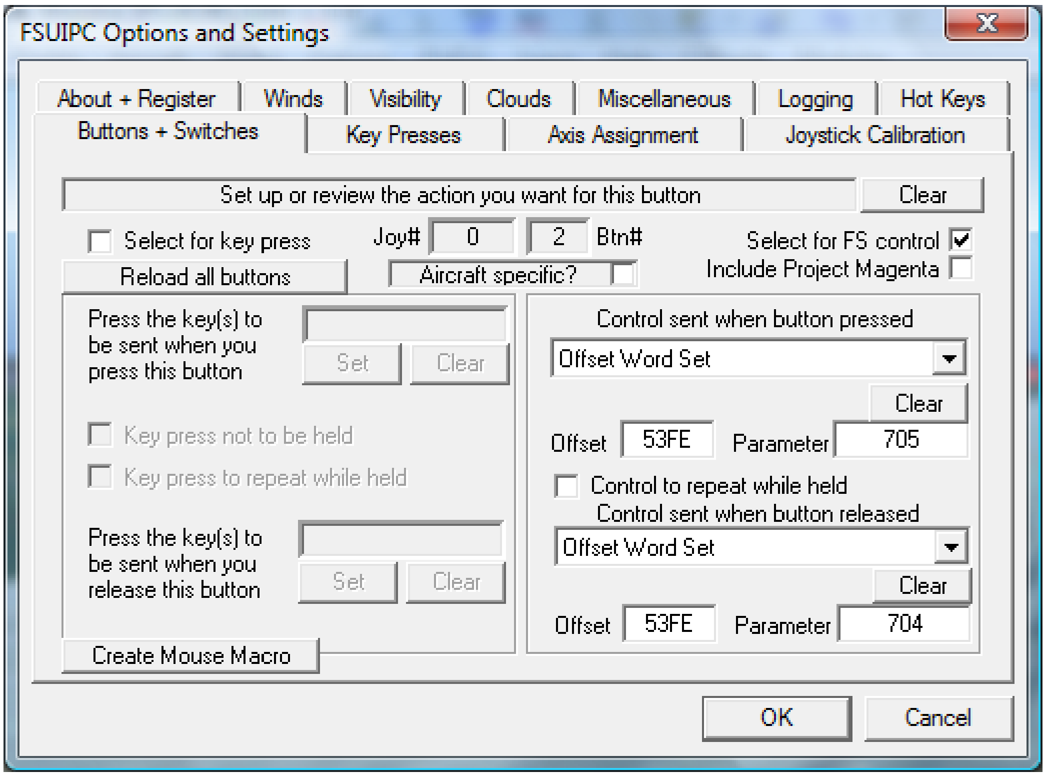
Press the key(s) to be sent when you press this button [Set] [Clear]
☐ Key press not to be held
☐ Key press to repeat while held
Press the key(s) to be sent when you release this button [Set] [Clear]
[Create Mouse Macro]

Control sent when button pressed [Clear]
Parameter
☐ Control to repeat while held
Control sent when button released [Clear]
Parameter

[OK] [Cancel]

You then need to fill in the details on the right hand side to assign an offset and value to the joystick button.

To do this select the “Select fo FS control” checkbox.


FSUIPC Options and Settings

About + Register | Winds | Visibility | Clouds | Miscellaneous | Logging | Hot Keys
Buttons + Switches | Key Presses | Axis Assignment | Joystick Calibration

Set up or review the action you want for this button — Clear

Select for key press ☐ — Joy# 0 — 2 Btn# — Select for FS control ☑
Reload all buttons — Aircraft specific? ☐ — Include Project Magenta ☐

Press the key(s) to be sent when you press this button — Set — Clear
Key press not to be held ☐
Key press to repeat while held ☐
Press the key(s) to be sent when you release this button — Set — Clear
Create Mouse Macro

Control sent when button pressed: Offset Word Set — Clear
Offset 53FE — Parameter 705
Control to repeat while held ☐
Control sent when button released: Offset Word Set — Clear
Offset 53FE — Parameter 704

OK — Cancel

To assign a MultiFunction Offset :
In the “Control sent when button pressed” box select “**Offset Word Set**” from the drop down list.

To assign a Direct Input Offset :
In the “Control sent when button pressed” box select “**Offset Byte Set**” from the drop down list.

Then in the “Offset” box enter the offset that you wish to update.
This could be the Direct or MultiFunction offset.

Next in the “Parameter” box enter the value you want the offset to have.
For a Direct Input offset this will usually be **1**. For the MultiFunction offset this will be the function value found on the multifuction list in the User Manual.

IF you want something to happen when you release the button then you can also fill in the “Control sent when button released” fields.

by Mark Hastings 2009