GFdisplay: GoFlight® display handler for FSUIPC

by Pete Dowson, © 27th November 2000

Support Forum: http://forums.simflight.com/viewforum.php?f=54

Version 1.30 of GFdisplay.exe

Note: All my Windows based software is always available in the latest versions from http://www.schiratti.com/dowson**.** (Selected modules are also available elsewhere).

This is *not* my web site (I have none) but the list is there courtesy of Enrico Schiratti

This package contains the following parts:

GFdisplay.exe	The program itself, version 1.30
GFdisplay.doc	This document: please read it! (Word 97 format)
GFdisplay.pdf	This document: please read it! (Acrobat format)
GFdisplay.ini	Sample INI file, ready programmed as described
FSUIPC_Buttons.ini	[Buttons] sections for cutting and pasting

NOTE: GoFlight is a registered trademark of GoFlight Inc., the makers of many useful add-on modules for Microsoft Flight Simulator (and others). Their website is www.goflightinc.com and you should pay a visit there from time to time to make sure your GoFlight software is up to date. In particular, GFdisplay, like FSUIPC and WideClient, uses GFdev.dll, which will be installed by the GF package into the same folder as the GFconfig program. Do *not* put that DLL into the FS Modules folder!

Contents

Introduction to GFdisplay

Ever since version 3.20 of FSUIPC, GoFlight switches, buttons and knobs have been programmable in FSUIPC. My WideFS package was also adapted to recognise their operation on Networked PCs and transmit them to FSUIPC as well. All this complemented, rather than replaced, the provisions made for the GF modules by GoFlight Inc's own software for Flight Simulator. However, using FSUIPC for these left the operation of the indicator lights and numeric and alphanumeric displays either unused or incorrect.

After searching for a while for a way to rectify this in FSUIPC itself, I came to the conclusion that direct display handling just was not suited to the way FSUIPC operates. The changes that would have been needed to allow it to control displays over a Network were so great that I was never really likely to have time to complete it even if I had started it.

So, the idea of a small, simple, separate utility program, just to drive the displays according to user parameters, was born. I worked on it in what little "spare" time I had, and this is the result. It is simple, and small, and efficient, *but* unfortunately this is at a price. The parameters to drive it are not very user friendly. In fact they will look a bit cryptic, so will take quite a bit of explanation. That is the purpose of this document, and the many usable examples provided.

One day, maybe, I will have lots of time and will be able to make a user-friendly front-end, to generate these "cryptic parameters" automatically from, possibly a nice set of dialogue or menu windows, or possibly some Basic sort of language. But, to be honest, that sort of software isn't my scene and it would bore me to tears. So, in the mean time, I offer the program as it is, with hopefully enough fully annotated examples so that those who wish to can still make good use of it.

Installation and Configuration

All there is of GFdisplay—the EXE itself and the INI—are simply placed into a folder, and the EXE is run, either from a shortcut, or by double-clicking the program in Explorer, or by having it automatically loaded by FSUIPC or WideClient, or even Windows (via the StartUp folder).

You will need version 3.465 of FSUIPC, *or later*, and preferably 6.465 of WideFS, *or later*. It will work on clients with earlier versions of WideFS, but not as well. You will need to be a fully registered user of FSUIPC to use GFdisplay on the same PC as FS—but then you must have full user registration to use the GF button programming in FSUIPC in the first place, and this present program merely complements that facility.

It can be run before FS or WideFS is running, or after. It can be left running, in the background, all day. It will not be doing anything when there's no connection to FS, except looking for that connection, and will sit minimised in the task bar with its title saying "connected" or "Waiting" as applicable.

Apart from the (eventual) connection to FS (whether via WideClient or direct on the FS PC), the only other need it has is to find the GoFlight driver module. This is called GFdev.dll, and should either be in its normal installation place (wherever GFconfig got installed), or, if you prefer, in the same folder as GFdisplay itself. Since both FSUIPC and WideClient also use this module (to detect switch operations), you should already have this correctly accessible by now in any case.

The configuration of GFdisplay is via its INI file, and this takes us directly to the display programming. The sections following with provide instructions by way of usable fully annotated examples, whilst the actual specification of the parameter formats will be delayed to an appendix, for reference use only.

You can edit GFdisplay.INI and save it whilst the program is running, and it will pick up the revised version and use it within a few seconds. This is useful when developing and testing. If there are any errors, an error message will be written back to the INI file, against the line in error, so don't forget to read it afterwards and check.

The same things don't apply to FSUIPC.INI I'm afraid. Make a copy *before* submitting it to FSUIPC. If it finds anything in error it will ignore it and you will simply lose the line—the subsequent lines will be renumbered, which can cause confusion. So *always* keep a safe copy elsewhere. To get FSUIPC to reread the Buttons section of the INI file you can change aircraft (reloading the same one doesn't always work because of FS's caching).

Button programming in FSUIPC is easier to debug and check if you enable Button logging (in FSUIPC's Logging page) when testing.

Note that if you are driving GF modules on the same PC as FS (i.e. not via WideFS), then if you are programming all of your GF modules on the PC using FSUIPC and GFdisplay, you should remove the GFdev2k4.dll (or equivalent for FS2002) from the FS modules folder. You *can* mix them, have some driven by the Go-flight software and some by mine, by simply not assigning any function in GFconfig, but be aware that the GF drivers may occasionally blank the displays GFdisplay is supposed to be controlling. Go-Flight are aware of this and a newer version of Gfdev2k4.dll may well fix this slight oversight.

Display programming

All of the GFdisplay programming is via standard Windows-type INI file editing, just as any advanced button programming is in FSUIPC. In fact, to make the examples here fully usable, they deal with both: the programming of the displays in GFdisplay.ini *and* the programming of the buttons in FSUIPC.ini. You will, with some little adaptation (like renumbering the lines on occasion) be able to extract those parts you want and paste them into your FSUIPC.INI file—assuming, that is, you would like to have the displays driven as I am going to demonstrate.

Where applicable, the examples will have two target uses, at the same time: FS's default avionics (radios, autopilot etc), and, *automatically* when it is running, the Project Magenta MCP/autopilot. I've chosen PM not only because I am an avid user, but also because it is the only add-on which uses FSUIPC offsets in a fully documented way. I know many would like to drive other advanced panels like those from PSS and PMDG with the displays correctly functioning, but I'm afraid it is often just not possible to get the data out for display, or if it is, it is not publishable for legal reasons. However, if you *can* find the data you need, you should be able to adapt the examples here to suit.

Outline of INI file structure

The INI file is divided into a number of sections each with titles in the form [name of section].

The first section is entitled [GF Connections] and is used by GFdisplay (not by you) to log things like Gfdev.dll found, and how many GF devices of each type are connected. For example:

```
[GF Connections]
GFDev=DLL found
GF45=1
GFLGT=1
GFMCP=1
GFRP48=1
```

There's a single section for "conditions"—these define any tests that may need to be carried out in any of the other sections in order to decide what to do. Things like deciding whether to display FS MCP values or PM MCP values would be via a test defined here. They are collected out separately partly to save repetition everywhere else and partly to keep each programmed action relatively simple. (Hmm!)

Then there's one section for each GF device you have, or even may have (one day). Sections for devices not currently connected are simply ignored. These sections are the ones that really define what is happening. They have names in the form [<devicetype>.<device number>], where the device types are

GF45
GF46
GFATC *(support not tested)*
GFLGT
GFMCP
GFMCPPRO
GFRP48
GF166
GFP8
GFT8

And the devices are numbered from 0 upwards when there are more than one of any type. So,

```
[GF166.2]
```

would be the heading for the section defining the treatment for the *third* GF166 device connected to this system.

Within the [Conditions] and specific device sections there are sequences of statements, and these are always executed is a specific order. This is important to realise, because in many cases what happens later in a sequence will depend on what happens earlier.

The following sections analyse a typical INI file, section by section. The file, ready for use, is included in this package along with FSUIPC [Buttons] entries ready by adaptation and cutting and pasting into your own FSUIPC.INI file. Of the eight GF module types currently supported the only ones without examples explained here are the GF45 (the GF46 is so similar) and the three simple ones—GFP8, GFT8 and GFRP48 (simple in terms of displays anyway). These have been generously left as an exercise for the reader. ;-)

[Conditions]

Here I will go through the Conditions section present in the accompanying GFdisplay.INI file, explaining each line or group of lines in turn. To fully understand all of this it will help if you had reference to the FSUIPC offsets list, available in the FSUIPC SDK www.schiratti.com/dowson, and the Project Magenta Documentation website, where you will also find the PM Offset list, also useful here (see www.projectmagenta.com/resources/docs.html).

```
0=X500 U16 ;PM's MCP is active if this is non-zero
```

Okay, starting simple. First, notice that conditions start at number 0. They are executed in ascending numerical order. Later conditions can refer to earlier ones, but not vice versa.

Breaking the line into its (few) components:

X500 is a reference to an FSUIPC offset, identified in hexadecimal. Offset 0500 is non-zero if the Project Magenta MCP is running, and this indication is used throughout to decide whether to drive things for FS defaults or for Project Magenta.

Note that this is actually an abbreviation for "X500 !=0" which means "if offset 0500 is not equal to 0". GFdisplay assumes the offset test is for "non-zero" if no test is specified explicitly.

U16 tells GFdisplay that the offset reference is to a 16-bit unsigned value, or a "WORD" in Windows terms.

All the rest, after the ";" are merely comments.

In order to test this condition the shorthand C0 is used "Condition 0". This tests the condition for truth. To test for it being false instead the shorthand is !C0.

```
1=X4F0 U16 M8000 ; PMs speed is Mach
```

Now this introduces one more element. As before, the X4F0 U16 part indicates the test is for offset 04F0, a 16-bit unsigned 'Word', being non-zero, but this time the M8000 modifies the test.

M8000 specifies that the value shall be subject to a 'Mask' first. The value in 04F0 is logically ANDed with the mask value (also in hexadecimal) before the result is tested.

This condition is simply testing for the setting of one bit (hex 8000, or bit 15, the top-most bit in the 16-bit word). This is actually a flag indicating that Project Magenta's MCP has its Speed display operating in Mach, not IAS.

```
2=X7E4 U32 ; FSs speed is Mach -- see condition 46 too
```

Well, you've met these parts already. This tests for the 32-bit value at offset 07E4 being non-zero. 07E4 is the FS Autopilot Mach hold flag, which is zero when mach hold is off, non-zero when on.

```
3=X4F0 U16 M0002 ; PMs V/S mode
4=X3300 U16 M0100 ; FS LOC acquired
5=XC4E U16 ; CRS nonzero?
6=X4E2 U16 ; PM HDG nonzero?
```

No new comments should be needed for these—check any offsets in the lists where you don't understand. From now on I will omit lines not needing further comment. Please refer to the actual enclosed INI file for the omissions.

```
7=X7CC U16 *360 /65536 ;FS's HDG nonzero?
```

Something new here. FS's offset 07CC is a 16bit value that contains the Autopilot Heading bug value, but in special FS units where the complete range of 360 degrees uses the whole 65536 value capacity of a 16 bit unsigned value.

*360 is part of the conversion to degrees. It simply says that, before testing, multiply (*) by 360.

/65536 completes the conversion. It says divide (/) by 65536.

These two arithmetic operations are the only ones supported, and you can only have one of each. Multiplication is done before division no matter in which order they are entered.

```
9=XBEC U16 =16383 ; Nose Gear Down
10=XBF0 U16 =16383 ; Right Gear Down
11=XBF4 U16 =16383 ; Left Gear Down
```

These lines introduce another type of test. Instead of the simple “not equal to zero” (or just non-zero), these entries show values being tested for equality to a *decimal* number. You can also do this with hexadecimal numbers. The =16383 here would be =X3FFF in hexadecimal. Note that the =X defines this as a value rather than an offset.

```
; offset 66C0 used as GF45/46 radio selector, 0-6 as follows:
12=X66c0 U8 =0 ;Radio selector 0=COM1
13=X66c0 U8 =1 ;Radio selector 1=COM2
14=X66c0 U8 =2 ;Radio selector 2=NAV1
15=X66c0 U8 =3 ;Radio selector 3=NAV2
16=X66c0 U8 =4 ;Radio selector 4=ADF1
17=X66c0 U8 =5 ;Radio selector 5=ADF2
18=X66c0 U8 =6 ;Radio selector 6=TPNR
```

Offsets 66C0–66FF are available for anything you wish. They are not and never will be allocated to any specific application program of internal FS values. In the examples we are discussing here several of these will be used in order to keep track of things being set by FSUIPC button programming and tested or used by GFdisplay.

Here, a single byte (U8, Unsigned 8-bit) is used to define what radio frequency will be displayed on a GF45/46 module. There are no new INI elements here to explain, but it is time to look at one line in the FSUIPC button programming which complements this set of tests (I will italicise FSUIPC parameters so we don’t get too confused with the GFdisplay parameters):

100=P125,0,Cx510066C0,x00060001

This defines button 0 on joystick 125 (the first GF46 on the FS PC will have this number in FSUIPC). This is the button on the left on the GF46. I’m using it to cycle through the frequencies above (COM1 to Transponder) by cyclically incrementing byte offset 66C0 by 1, cycling back to 0 after 6. All that can actually be entered in FSUIPC’s Buttons page in FS, but the explanation of the encoding here is in the FSUIPC Advanced User’s document.

Whilst we are here, please note that the same GF46 on a WideFS client PC will have 1000 or 2000 or 3000 (etc) added to its ‘joystick number’ in FSUIPC—the actual value depending on the Client number assigned by WideServer. You can see client numbers in the WideServer INI file, but it is generally easier to determine joystick numbers and button numbers for GF buttons by pressing them whilst watching FSUIPC’s buttons option page. If you will be cutting and pasting parts of the [Buttons] section supplied with this package into your own FSUIPC.INI file, just be aware that for Client-attached GF modules you will have to change the joystick numbers in each relevant entry, and usually by adding a thousands digit only.

```
; GF166 support includes both radios and an offset viewing debug mode
; Offset 66C4 has flag bits as follows
;    bit 0 (1) = debug offset view if 0, debug type view if 1
;    bit 1 (2) = debug decimal display if 0, hex display if 1
;    bit 2 (4) = radio if 0, debug offset mode if 1
;    bit 3 (8) = standby shown if 0, radial shown if 1
;    bit 4 (16)= set by Centre button press, Cleared when released and if L or R pressed
;    bit 5 (32)= set for MCP Mach display instead of IAS (FS MCP)
; Offset 66C1 used for debug mode offset type (0-12)
; Offset 66C5 used as radio selector, 0-5
; Offset 66C2 (2bytes) is offset in debug mode
```

I’ve pulled these comments from the file into the text here in order to mention something rather important. In order to provide a decent demonstration of the versatility of the GF166 and the supporting FSUIPC and GFdisplay facilities, I have provided my own programming for it, which is probably rather over the top for most users. I have programmed it to not only provide a 6-way dual display radio, but also to provide a way of displaying FSUIPC offset values in a number of formats. This has helped debug some of the code itself, so I refer to it as “debug mode”.

You will see from the comments reproduced above that this involves offsets 66C1 (U8), 66C2 (U16), 66C4 (U8) and 66C5 (U8), and, furthermore, the byte 66C4 actually contains 6 different flag bits. Economy like this can be important with limited numbers of offsets available.

I’ll explain more about these developments when we get to the GF166 section.

```
34=X66C1 U8 <7 ;Fixed point
35=X66C1 U8 <9 ;Numeric
```

Here is the first use of a test other than not equal (!) and equal (=). The <7 section tests that the value is less than 7. As well as ! (not equal), = (equal), < (less than), > (greater than) you can have <= (less than or equal) and >= (greater that or equal).

```
46=!C1 !C2 X66C4 U8 M20 ;Mach not IAS
```

We end the section on Conditions with a condition that does actually refer to other (previously defined) conditions:

!C1	tests condition 1 (the one for PM’s speed being Mach) for false
!C2	tests condition 2 (the one for FS’s Mach hold being enabled) for false

The C46 condition is therefore checking that neither PM nor FS are in Mach mode *and* that flag bit 5 in 66C4 is set.

Okay. No questions? Good. On to the specific devices:

[GF46.0]   Multiple radio display and setting

The supplied GFdisplay.ini file does actually include a section for GF45.0 as well, but they are so similar I will only examine the GF46 one here. The FSUIPC.INI parts would also be similar apart from the lack of a button to cycle through the frequencies—you’ll need to program a button elsewhere for this if you want to use the GF45 in the way I’ve programmed it.

Most of this is straightforward:

```
Needs=V16 B E A
```

This parameter tells GFdisplay what needs to be set or available in FS in order to light up the units displays. This example actually needs everything currently tested by GFdisplay, so it is a good example to go into details with:

V16	says that the battery Voltage must be at least 16.
B	says that the Battery must be switched on.
E	says that the Electrical subsystem must not have been ‘failed’.
A	says that the Avionics must be switched on.

If any of these listed conditions is not true, the device is blanked.

```
B=1
```

This sets the Brightness level for the device. Possible values are 0 (minimum) to 15 (maximum). I set it to 1 which I find easily bright enough (the LED elements are very large), though it does make the illumination rather uneven.

If you wish you can define the brightness via offset values and conditions, just like display values. The only uses I can think of for this would be to program a button or knob to adjust the brightness manually at any time, and/or automatically by referring to FSUIPC offset 115E (time of day indicator) so that it dims for night flying.

```
D0.1=C12 ="COM1"
D0.2=C13 ="COM2"
D0.3=C14 ="NAV1"
D0.4=C15 ="NAV2"
D0.5=C16 ="ADF1"
D0.6=C17 ="ADF2"
D0.7=C18 ="TPDR"
```

The D lines define actions for a specific display. Each GF module with displays has these numbered from 0. In the case of the GF45 and GF46, display 0 (D0) is the left-hand alphanumeric display and D1 is the right-hand numeric display.

All seven of the above lines are defining possible displays for the D0. They are processed in numerical order of the second number, “.1” to “.7”. Processing for this display stops when the first such line is actually found applicable—that is whatever conditions applied in it are “true”.

Note that the sequence starts with .1, not .0. If a display is simple enough to only need one line, you just omit the .n part. So `D0= ="TEST"` would simply and unconditionally display TEST. (Note that this literal assignment can be abbreviated to `D0="TEST"`).

The condition tests C12–C18 are testing the radio selector, the byte at 66C0. The same conditions are applied for the actual frequency, displayed in D1:

```
D1.1=C12 X34E R2
D1.2=C13 X3118 R2
D1.3=C14 X350 R2
D1.4=C15 X352 R2
D1.5=C16 X34C R1
D1.6=C17 X2D4 R1
D1.7=C18 X354 R0
```

These define the FSUIPC offset containing the value to be displayed, and a “display format” value, Rn.

R0	a BCD radio frequency with no fractional part—the transponder (XXXX).
R1	a BCD radio frequency with one fractional digit—an ADF (XXXX.X).
R2	a BCD radio frequency with two fractional digits—COM and NAV (1XX.XX).

The only remaining things to consider for the GF46 example are the FSUIPC.INI entries, including the one for the button already seen:

```
100=P125,0,Cx510066C0,x00060001
```

This cycles offset 66C0 through 0–6, thus selecting which radio is being displayed. But, fof course, this must also change what the dual concentric knobs do. Each radio has its own set of FS controls to adjust its fractional and integral parts up or down. This makes for a *lot* of entries in the FSUIPC INI file. See the sample file provided for the complete list (lines 100–212). For our explanation here I will take only one radio, the COM1, and only one adjuster, the outer one (for integral changes):

```
101=B66C0=0 P125,14,C1030,0
102=B66C0=0 U125,14,C1030,0
103=B66C0=0 P125,15,C1030,0
104=B66C0=0 U125,15,C1030,0
105=B66C0=0 P125,13,C1031,0
106=B66C0=0 U125,13,C1031,0
107=B66C0=0 P125,12,C1031,0
108=B66C0=0 U125,12,C1031,0
```

This section defines the actions for fast turn left (button 15), slow turn left (14), slow turn right (13) and fast turn right (12). There are identical entries for fast and slow because there are no separate fast and slow adjustment controls. There are also identical entries for “Press” (P) and release (Up or “U”), because on each “click” these knobs either make or break. Without programming both you’d need two clicks per change.

This produces the eight entries for what are really only two controls: the FSUIPC-added “COM1 use whole inc” (1030) and “COM1 use whole dec” (1031). The added FSUIPC ones have to be used because the FS built-in controls operate only on the standby values, and with the GF46 (and GF45) only having one usable frequency display, we don’t have the ability to deal with those.

Note that we are using the FSUIPC offset testing facility here. “B66C0=0” refers to the Byte at offset 66C0 and tests it for zero, the value indicating COM1.

The other eight lines dealing with COM1 (lines 133–140) are the ones dealing with the inner, fractional adjuster.

With seven different radio functions you can now see why so many lines in the INI are needed. 7 x 8 x 2 = 112, plus just the one line for the button to select the radio!

[GFLGT.0]   Landing Gear, Trim, Flaps

I thought we’d take the easiest, shortest, one of the bunch next. Just 3 LED indicators for Gfdisplay to deal with—but I nice easy start for indicator control. We’ll program the three controls in FSUIPC too.

```
Needs=E B
```

With the battery on we have the Gear indicators lighting in all except an electrical failure. That probably isn’t very accurate, but it certainly won’t relate to the avionics switch.

```
L7=XBEC U16 =16383 ;Nose Gear Down
L6=!C9 XBEC U16 !=0 ;Nose Gear Moving
L5=XBF4 U16 =16383 ;Left Gear Down
L4=!C11 XBF4 U16 !=0 ;Left Gear Moving
L3=XBF0 U16 =16383 ;Right Gear Down
L2=!C10 XBF0 U16 !=0 ;Right Gear Moving
```

The first thing to note is the new line type. “Ln” refers to “LED” or “Light” number n. Each GF device numbers them in rather different ways. The three gear LEDs on the GFLGT are in fact operated as 6 lights—3 red and 3 green. You can figure out from the list above which is which. There’s no Light 1.

There’s nothing else new here, but note the condition tests on !C9, !C10 and !C11. These are for the same values *not* equal to 16383. This is only necessary because we want to light the “moving” LED (red) whenever the FS value indicates neither up (0) nor down (16383). That’s two tests, but only one is allowed per line. So we use a Condition for the other test. This is often a useful device and will be seen again.

The FSUIPC programming for the LGT is mercifully short too:

```
250=P165,4,C65607,0
251=U165,4,C65607,0
252=P165,5,C65607,0
253=U165,5,C65607,0
254=P165,6,C65615,0
255=U165,6,C65615,0
256=P165,7,C65615,0
257=U165,7,C65615,0
```

Those are the trim wheel “buttons” (4–7 fast “down” to fast “up”), with the usual treatment for rotaries to make every tick count.

```
258=P165,2,C65759,0
259=P165,1,C65758,0
```

The Flaps lever is off, or operating buttons 1 or 2. I’ve not selected “repeat” here because the rpeeats are too fast and make it difficult to select a specific flap setting.

```
260=P165,0,C66079,0
261=U165,0,C66080,0
```

The gear up and gear down controls in FS used specifically, so that the lever stays in sync. Don’t use the “toggle” controls with latching switches like this, only with buttons.

[GF166.0]   Multiple radio, plus ‘debug mode’ offset viewer

The example for the GF-166 presented here is more complicated than that for the GF-46 given earlier for two reasons:

1. The GF-166 has two displays capable of showing radio frequencies rather than just the one, and is obviously modelled on the standard Bendix-King type of radio with “in use” and “standby” values. The centre button is exactly right for the “swap” button, and one of the others is pressed into use as a “Radial” selection button—this is only used for a NAV radio, and replaces the standby display with the current VOR radial, or “---” if there’s no reception.
2. For my own use, and presented here really only as an example of more complex application of the facilities available, I have also provided a “debug” mode on the GF-166. This replaces the left display with an FSUIPC offset (in hexadecimal) and the right display with the value it currently contains. Both the offset and the type of value can be selected.

Okay. Let’s look at the GFdisplay.INI file entries for the 166:

```
; Debug offsets use
D0.1=C36 !C19 X66C2 U16 DX40 ; In offset display mode, left side is offset in hex
```

In the supplied INI file, all of the “debug” mode elements are listed before the radio use ones. The Condition 36 is:

```
36=X66C4 U8 M04 ; Flag for Debug mode, else Radio
```

which simply tests a bit (bit 2, value 4) in the local offset 66C4. We will see later how we use buttons to change that flag via FSUIPC. C36 is used for all the debug mode lines, !C36 for all the radio mode lines.

The first Display, D0, is the left-hand one. In Debug mode this will either contain the offset being viewed – a 4 digit hexadecimal number (like “66C4”), *or*, if we are selecting a number format, it will display that instead.

!C19	tests bit 0 (value 1) in 66C4. This is used to switch between offset and number type selection. !C19 means offset.
X66C2	is where we will maintain the actual offset value (i.e. the address, like 66C4) being displayed or adjusted.
U16	simply says this value is an unsigned 16-bit ‘word’.
DX40	defines how the value is to be displayed.

Before this we only saw simple literal string displays, like =“COM1”, and the different Radio displays, Rn. Most numerical values will, however, be displayed in decimal or hexadecimal with so many integer digits and so many fractional digits. In this case the display is specified as hexadecimal (X) with 4 digits and no fraction.

The actual offset being displayed, as contained in 66C2, is adjusted in this mode by the dual concentric knob on the GF166. The inner part adjusts the lower two digits and the outer part the upper two, with “fast mode” changing the higher digit in each pair. The FSUIPC.INI entries for this will be seen later.

```
D0.2=C36 C21 ="u 16" ; Else shows selected type
D0.3=C36 C22 ="s 8"
D0.4=C36 C23 ="u 8"
D0.5=C36 C24 ="s 16"
D0.6=C36 C25 ="u 16"
D0.7=C36 C26 ="s 32"
D0.8=C36 C27 ="u 32"
D0.9=C36 C28 ="F 32"
D0.10=C36 C29 ="F 64"
D0.11=C36 C30 ="str"
D0.12=C36 C31 ="r 0"
D0.13=C36 C32 ="r 1"
D0.14=C36 C33 ="r 2"
```

I’ve shown all these here, although they are simple variations on a theme, so you can see the list of number formats the debug mode actually supports. First, though, notice that these are dependent only on conditions C36 (true for debug mode), and one of the range C21–C33. The latter are all simply tests of the value in offset 66C1, acting on values there of 0–12.

There is no need to test condition C19 in these (the condition which selects the number format display in the left-hand side instead of the hex offset) because if it were *not* set, the first line, “D0.1=…” already examined would have been operative, and as pointed out earlier, once a usable display has been found in the sequence for the display (Dn.1, 2, 3, …), the rest are not processed.

Briefly explained, the number formats supported are:

S 8:	Signed 8-bit byte	F 32:	Floating point 32-bit (“float”)
U 8:	Unsigned 8-bit byte	F 64:	Floating point 64-bit (“double”)
S 16:	Signed 16-bit word	STR:	Character string (ASCII)*
U 16:	Unsigned 16-bit word	R 0:	Transponder setting XXXX
S 32:	Signed 32-bit double word	R 1:	ADF frequency XXXX.X
U 32:	Unsigned 32-bit double word	R 2:	COM/NAV frequency 1XX.XX

*Note that the program does its best to display non-decimal characters legibly, but because of display limitations some letters may be ambiguous or omitted.

The value in 66C1 selecting the format is adjusted by either part of the frequency adjustment knob, as shown later.

Now we come to the ‘clever’ bit:

```
D1.1=C36 C34 !C20 IX66C2 I66C1 D00 ; Decimal fixed point numeric
D1.2=C36 C34 C20 IX66C2 I66C1 DX00 ; Hex fixed point numeric
D1.3=C36 C35 !C20 IX66C2 I66C1 D02 ; 2 fractional places
D1.4=C36 C35 C20 IX66C2 I66C1 DX02 ; ditto in hex
D1.5=C36 IX66C2 I66C1 ; Strings and radios
```

Here we introduce two new facilities: indirect offset and indirect value type.

IX66C2 tells GFdisplay to obtain the offset from an offset. In this case it gets the needed offset from 66C2. It knows 66C2 must be an unsigned 16bit value because all offsets have to be such.

I66C1 similarly tells it to get the value type from offset 66C1. Value types are encoded as a byte value, with, tes, you’ve guessed it, the values 1 to 12 which we place in 66C1. The default, if the value is outside this range, is “U 16”, which is why we had an extra entry for “U 16” earlier—the D0.2 line. When FS is first started, 66C1 will be zero, so that will be the initial format.

The other conditions tested here are simply to control the actual display format—remember, the Ixxxx part tells it the type of value it is reading from the offset, but it doesn’t necessarily tell it the display format (strings and radios excepted, as shown in the D1.5 line).

D00	is a decimal number with the integer part only showing, to as many places as will fit
DX00	is similar but hexadecimal
D02	is a decimal number with variable integer digits but two fraction digits.
DX02	is similar but hexadecimal

Conditions C34 and C35 just differentiate between fixed point/floating point, and numeric/“strings and radios” respectively. C20 is the hexadecimal/decimal display mode selected by the user (buttons, later) and flagged with bit 1 (value 2) in 66C4.

Two of the debug modes are indicated back to the user via the LED indicators:

```
L1.1=C36 X66C4 U8 M01
L2.1=C36 X66C4 U8 M02
```

The left LED (L1) shows the status of the left display (lit for value type shown, extinguished for offset shown). This is controlled by the left button, so it is an appropriate indication.

The right LED (L2) shows the status of the right display (lit for value in hex, extinguished for decimal). This is controlled by the right button, so it also is an appropriate indication.

Note that both are still conditional on C36. The LED indicators are used differently when radio mode is in operation.

Okay, on to the radios:

```
; Radio use
D0.15=!C36 C37 X34E R2 ;COM1
D0.16=!C36 C38 X3118 R2 ;COM2
D0.17=!C36 C39 X350 R2 ;NAV1
D0.18=!C36 C40 X352 R2 ;NAV2
D0.19=!C36 C41 X34C R1 ;ADF1
D0.20=!C36 C42 X2D4 R1 ;ADF2

D1.6=!C36 C37 X311A R2 ;COM1 sby
D1.7=!C36 C38 X311C R2 ;COM2 sby
D1.8=!C36 C39 !C43 X311E R2 ;NAV1 sby
D1.9=!C36 C40 !C43 X3120 R2 ;NAV2 sby
D1.10=!C36 C39 C43 C44 XC50 U16 *360 /65536 D30 ;NAV1 radial
D1.11=!C36 C39 C43 !C44 ="---"
D1.12=!C36 C40 C43 C45 XC60 U16 *360 /65536 D30 ;NAV2 radial
D1.13=!C36 C40 C43 !C45 ="---"
D1.14= ="" ; Else right display blanked
```

This is all pretty much as you've seen before. The !C36 condition is needed because it's the debug mode/radio mode condition. The C37–42 conditions select the specific radio, of the six I am supporting here (sorry, no transponder this time). The left-hand display (D0) is uncomplicated—it always displays the current in-use frequency. The right-hand-side (D1) is made more complicated by:

1. I am optionally showing the current radial for VORs, in place of the standby frequency, and
2. The standby frequencies for ADFs aren't supported, so the right display needs blanking in those cases.

Still, there's nothing new in any of these lines. The radial/standby switch is by bit 3 (value 8) in 66C4, tested by C43. This is toggled by the right-hand button, as we will see. Conditions 44 and 45 are there to check for a valid VOR signal, because if there isn't one it would be misleading to show a radial. If there is no VOR signal, the radial display is "---".

```
L1.2=!C36 X66C5 U8 M01 ;Left LED lit for #2 radio, else #1

L2.2=!C36 C39 X3300 U16 M0002 Fslow ;Right LED, active NAV1
L2.3=!C36 C40 X3300 U16 M0004 Fslow ;Right LED, active NAV1
L2.4=!C36 C41 X3300 U16 M0008 Fslow ;Right LED, active ADF1
L2.5=!C36 C42 X3300 U16 M0800 Fslow ;Right LED, active ADF2
```

Finishing off the display part of the GF-166 radio treatment, these deal with the use of the two indicator LEDs.

The left one is used to differentiate between #1 and #2 in each of the three radio types, COM, NAV and ADF. This is because, although it is easy to see what type of radio it is from the frequency display (COM 118–136, NAV 108–117, ADF different format nnnn.n), it cannot otherwise be seen wther it is COM1 or COM2, NAV1 or NAV2, or ADF1 or ADF2. The left hand LED tells you – #1 if not lit, #2 if lit.

Because 66C5 is used as the radio selector, and goes (cyclically) from 0 to 5, the lowest bit (bit 0) tells us whether it is radio #1 or #2. That is what is tested here to light the indicator.

The right-hand LED is used to show valid VOR or ADF reception on the selected NAV or ADF radio. This is by appropriate bits in the FSUIPC offset 3300. The LED is flashed slowly if this is true.

Fslow	tells the program to flash the display slowly
Ffast	tells it to flash it faster

We don't have another example here, but you can flash any display, not only the LED indicators.

Okay, now let’s look at the button programming in FSUIPC to accomplish all this. First, debug mode:

```
; GF166.0
401=B66C4&5=x4 P141,14,Cx510066C3,x00FF0001
402=B66C4&5=x4 U141,14,Cx510066C3,x00FF0001
403=B66C4&5=x4 P141,15,Cx510066C3,x00F00010
404=B66C4&5=x4 U141,15,Cx510066C3,x00F00010
```

Lines 401 to 416 are all about adjusting the hex offset when in debug mode (bit 2 in 66C4 set) and the offset is displayed (bit 0 in 66C4 clear). Thus the test “B66C4&05=4”, which tests both conditions in one test. This is a useful trick, as FSUIPC only supports a single offset condition test. Always try to arrange your flags so they can be tested together like this.

The outer knob (buttons 15 fast left to 12 fast right) is used to change the higher byte of the offset, in 66C3, whilst the inner (buttons 11 fast left to 8 fast right) is used to adjust the lower byte, 66C2. The fast mode adjusts by 16 (hex 10) whilst the slow mode adjusts by 1.

```
421=B66C4&05=5 P141,14,Cx510066C1,x000C0001
422=B66C4&05=5 U141,14,Cx510066C1,x000C0001
423=B66C4&05=5 P141,15,Cx510066C1,x000C0001
424=B66C4&05=5 U141,15,Cx510066C1,x000C0001
```

When the variable type is displayed on the left (bit 0 in 66C4 set), the same knobs are usined simply to cycle the value in 66C1 through the values 0–12. The inner and outer knobs do the same in this case.

The buttons in debug mode are controlled by:

```
449=B66C4&14=x4 P141,1,x0D0066C4,x00000001
451=B66C4&14=x4 P141,2,x0D0066C4,x00000002
```

The test “B66C4&14=x4” here is checking that bit 2 is set (for debug mode) *without* bit 4 also being set. We’ll see, below, that bit 4 (value 16, or hex 10) is used to indicate that the Centre button is held pressed—this is used to swap modes! (I bet you’ve been wondering about that all this time?).

Button 1, the left one, here toggles bit 0 in 66C4, changing between offset displayed 9bit clear) and variable type (bit set).

Similarly button 2, the right one, toggles bit 1, determining whether the value is shown in decimal or hex.

Okay. This is getting a bit long-winded, isn’t it? So let’s rush through the small number of remaining lines:

```
440=P141,0,cX050066C4,x00000010 ; Mark centre pressed
```

The centre button is used for two things: to swap frequencies in radio mode, and, in *either* mode, to act as a “shift” along with one of the other buttons to select which mode the unit should switch to. This dual purpose for one button makes it a little complicated.

The 440 line, above, sets a flag in 66C4 (bit 4, in fact, hex 10), when the centre button is pressed. A later line:

```
448=U141,0,cX090066C4,x00000010 ; Clear centre pressed flag
```

clears that flag when the button is released.

```
442=W66C4&714=x0010 U141,0,Cx01003123,x08 ;Swap COM1
443=W66C4&714=x0110 U141,0,Cx01003123,x04 ;Swap COM2
444=W66C4&714=x0210 U141,0,Cx01003123,x02 ;Swap NAV1
445=W66C4&714=x0310 U141,0,Cx01003123,x01 ;Swap NAV2
```

These operate the “swap” button, to exchange standby and in-use frequencies. Not that this is done, not when the button is pressed, but when it is released (“U”). It has to be this way because if it is pressed and held whilst one of the other two buttons is pressed, this swaps modes, not frequencies.

The conditions (W66C4& …) are another example of combining several tests into the one allowed. “66C4&x14” needs to be x10 to indicate that (a) the centre button was pressed *without* another button—explained in a moment—and (b) the mode is radios not debug. The high part of the Word at 66C4 (i.e. offset 66C5) actually contains the radio type—i.e. COM1 through to ADF2. Only the COM and NAV radios have standby frequencies for swapping, however.

```
450=B66C4&14=x0 P141,1,Cx510066C5,x00050001
452=B66C4&14=x0 P141,2,Cx0D0066C4,x00000008
```

These two deal with the use of the left and right buttons in radio mode (66C4 flag bit 2 not set). The left button selects the radio by cycling 66C5 from 0 to 5. The right button simply toggles 66C4 flag bit 3, which is used for NAV radios to indicate Radial mode. When the NAV radial is displayed in place of the standby frequency, the frequency adjustment knob operates on the in-use frequency, not the standby one. This adds yet more entries later!

```
453=CP(+141,0)141,1,Cx050066C4,x04
454=CP(+141,0)141,1,Cx090066C4,x1B
455=CP(+141,0)141,2,Cx090066C4,x1F
```

These operate the actual switch between radio mode (66C4 bit 2 clear) and debug mode (66C4 bit 2 set). The button conditional, (+141,0) makes these lines conditional on the Centre button being pressed still.

Lines 453 and 454 are obeyed when the centre button is held whilst the left button is pressed. This *sets* bit 2 (enabling debug mode), and clears all the other mode bits so they start off defaulted.

Line 455 is for the pressing of the right button when the centre one is held pressed, and sets radio mode. All flags including bit 2 are cleared.

Notice that they clear bit 4 (hex 10), which effectively stops the frequency swap when the centre button is released.

All the rest of the lines (456–583) are solely concerned with the frequency adjustments in radio mode. For the standby frequencies (COM, and NAV in non-radial display mode) the FS controls are used. For the in-use frequencies (ADF and NAV in radial display mode) the added FSUIPC controls have had to be used.

[GFMCP.0] MCP for both FS autopilot and Project Magenta

(See appendix 2 for a user-supplied example, based on this, for the GF-MCP Pro)

I've got this working and automatically switching between FS's autopilot and the Project Magenta MCP according to whether it detects the latter running (our Condition 0, C0, see right at the start).

You may notice that, in comparison with FS's autopilot, the PM MCP values do not always update so readily as you are adjusting them with the rotaries. This is particularly true when using the facilities for control of the PM MCP values through the inc/dec bits it provides in the PM offsets (it is these which used by the relevant added FSUIPC PM controls). Here I've done four things to make this smoother:

1. I run the PM MCP on the same PC as Flight Sim. Since the MCP is the "hub" of Project Magenta, I think this helps make things smoother all round. Of course it also helps if you have a powerful processor for FS too. With a Pentium 4 with hyper-threading you shouldn't notice any deterioration in FS's performance.
2. I changed the cycle timing parameter in MCP.INI from its default of 50 milliseconds all the way down to 10. With the version of PM's MCP currently on release this doesn't work (it has a minimum of 50), but I understand that the author will be relaxing this to allow values down to 5 to be specified.
3. I attached the GFMCP unit itself to the FS PC as well. You need to remove the Go-Flight driver module (GFMCP2k4.dll) from the FS modules folder. Be sure that you do *not* have the PM MCP configured to handle any serial-connected hardware devices (PFC, CPflight or Aerosoft MCPs), as when you do this the program concentrates on supporting the hardware directly and the response times for other switches and knobs gets far too slow. You should really only have one source of inputs for its operation
4. I programmed the GFMCP to update the PM MCP values "directly", or at least as directly as I could. According to the PM offsets documentation there are separate locations for *writing* (5406–540E) and *reading* (04E0–04E8). The latter are the actual values being displayed and, usually, used.

   I had to make a small change to FSUIPC to handle this. In some cases the values which are *read* in the 5406–540E locations are zero even though there are non-zero values being displayed. I think this is related to MCP initialisation. The values will certainly be zero initially in any case. Since I need to increment or decrement the value by reading it and changing it this isn't good, so FSUIPC now intervenes when these locations are read and if zero it provides the value read from the 04E0–04E8 area instead. (This only applies to the offset increment/decrement controls).

   A future version of the PM MCP may provide a single set of offsets for read/write, like FS. If so, the programming in the example may need some small amendments. Probably either the read-outs at 04E0–04E8 will need changing to refer to 5406–540E, or vice versa.

```
D0.1=C5 XC4E U16 D30 ;NAV1 CRS as nnn
D0.2=!C5 "360"
```

The “course”, whether in PM or FS, is the same. It is known as the OBS in FS, and there’s one for the NAV1 radio and another for NAV2. Here we program it from the NAV1 (VOR1) value.

The C5 condition tests for zero degrees. The internal value in FS runs 0–359, but the MCP displays 360 not 000.

```
D1.1=C0 C6 X4E2 U16 D30 ; PM's HDG as nnn
D1.2=C0 !C6 "360"
D1.3=!C0 C7 X7CC U16 *360 /65536 D30 ;FS's HDG as nnn
D1.4=!C0 !C7 "360"
```

Here we see the PM heading value (D1.1 and D1.2) and the FS heading value (D1.3 and D1.4) being set in the same way. In both cases 000 is changed to 360. In the FS case the value has to be converted from FS’s units.

```
D2.1=C0 C48 "" ; PM blank Spd
D2.2=C0 !C48 C1 X4E8 U16 /100 D02 ; PM's Mach as (n).nn
D2.3=C0 !C48 !C1 X4E0 U16 D30 ; PM's IAS as nnn
D2.4=!C0 C46 X7E8 U32 /65536 D02 ; FS's Mach as (n).nn
D2.5=!C0 !C46 X7E2 U16 D30 ; FS's IAS as nnn
```

In the PM case, C48 tests whether the PM MCP wants the speed window blanked (as when the speed is under VNAV control). The only other complication here is whether the speed to be displayed is IAS or Mach. The conditions C1 (PM) and C46 (FS) deal with that. We’ll see what affects C46 later.

```
D3.1=C0 C47 "" ; PM blank VS
D3.2=C0 !C47 X4E6 S16 *100 D-40 ; PM's V/S as (-)nnnn
D3.3=!C0 X7F2 S16 D-40 ; FS's V/S as (-)nnnn
```

The V/S is one of the few signed values we meet. Note the display format specifier “D-40” for a 4-digit signed decimal value.

```
D4.1=C0 X4E4 S16 *100 D-50 ; PMs Alt as (-)nnnnn
D4.2=!C0 X7D4 S32 *3.28084 /65536 D-50 ; FSs Alt as (-)nnnnn
```

Since the altitude value is, theoretically, signed, I’ve allowed for a signed value here, though in practice, in airliners at least, you’d never see a negative altitude on the MCP. However, the GF-MCP has allowed for the full 5 digits for the altitude (in feet) and the sign.

This shows a conversion with floating point values included. The multiply (*) and divide (/) entries can be integral or include fractional parts, as here, as needed. The 3.28084 value here is the number of feet in a metre.

```
L0.1=C0 X4F0 U16 M0020 ; PMs LOC -> NAV light
L0.2=!C0 C8 X7C4 U32 ; FS's NAV acquired
L0.3=!C0 !C8 X7C4 U32 Fslow ; FS's NAV (flash)
```

The NAV indicator (called VOR/LOC on a Boeing airliner MCP) is shown here left to PM’s LOC setting for PM, but operated Bendix-King style on FS’s autopilot. The standard Bendix-King autopilots flash the NAV indicator when the mode is engaged, and light it steady when it is captured. This is the condition tested by C8.

```
L1.1=C0 X4F0 U16 M0080 ; PMs HDG
L1.2=!C0 X7C8 U32 ; FS's HDG
L2.1=C0 X4F0 U16 M0200 ; PMs Speed
L2.2=!C0 !C2 X7DC U32 ; FS's IAS -> SPD
L2.3=!C0 X7E4 U32 ; FS's Mach -> SPD
L4.1=C0 X4F0 U16 M0008 ; PMs ALT
L4.2=!C0 X7D0 U32 ; FS's ALT
```

I’ve left the above lines in the document for completeness, but the need no extra comment (by now you’ve seen it all!). Note that there’s no Light 3 on the unit.

```
L5.1=C0 X4F0 U16 M0010 ; PMs APP
L5.2=!C0 C4 X800 U32 ; FS's APP LOCKED
L5.3=!C0 !C4 X800 U32 Fslow; FS's APP Pending (flash)
```

This is similar to the treatment of the NAV (VOR LOC) indication above. For FS use the APP indicator flashes whilst engage, and lights steady when captured.

Note that on a standard Bendix King autopilot, there would be a GS (glideslope) LED, and this would flash until captured, then go steady. If you’d prefer the indication be to confirm GS acquisition, change Condition 4 to read:

```
4=X3300 U16 M0200 ; FS GS acquired
```

```
L6.1=C0 X4F0 U16 M0002 ; PM no BC, use for AP Master 2
L6.2=!C0 X804 U32 ; FS's FSs BC
L7.1=C0 X4F0 U16 M0001 ; PMs AP Master 1
L7.2=!C0 X7BC U32 ; FS's ap eng
```

The last two indicators are those for “A/P CMD” and “B/C”. On FS these correspond to the AP Engage and BC mode (back course) respectively. On PM I’m using the A/P CMD button for the Left A/P (APL or AP1) and the B/C button for the Right A/P (APR or AP2). This allows you to engage autoland in PM, which needs two A/Ps engaged once APP mode is set.

Note the omissions from a full airliner MCP include the AutoThrottle arming switch, LNAV, VNAV, and *either* FLCH or ALT HOLD, depending which you’d prefer on the GF-MCP. These omissions will need programming on another unit. In the FSUIPC button programming example included with the package I’m operating PM’s “FLCH” mode with the ALT HOLD button, as this is probably generally a lot more useful if you only have one to choose from.

Moving on to the FSUIPC button programming, there’s not a lot to say, although there *are* a lot of entries needed. That’s because, for every knob, you need 4 Press and 4 release (Up) entries for each of FS and MP. That’s 16 lines per knob, or 80 lines before looking at the simple push buttons and disconnect bar themselves.

Scanning quickly through the lines and picking out some which may need more explanation:

```
599=P149,3,C1005,3870
```

The very first one introduces the first complexity. Button 3 is the “Sel” button, the one which should switch to and fro between IAS and Mach for both the display and the speed hold, assuming the auto-throttle is armed.

Line 599 simply toggles the Flag associated with (non-existent) button 15,30 (joystick 15, button 30). The ‘3870’ value here comes from 256*15 + 30. This flag is tested later.

```
600=D07DC P149,3,C65915,0
601=D07DC P149,3,C1003,3869
602=D07E4 P149,3,C65891,0
603=D07E4 P149,3,C1003,3869
604=W0500=0 CP(F-15,29)(F+15,30)149,3,C65915,0
605=W0500=0 CP(F-15,29)(F+15,30)149,3,C65915,0
606=W0500=0 CP(F-15,29)(F-15,30)149,3,C65891,0
607=W0500=0 CP(F-15,29)(F-15,30)149,3,C65891,0
605=P149,3,Cx0D0066C4,x20
606=P149,3,C1004,3869
```

All these are dealing with the problems of IAS/Mach selection. Even then they don’t *quite* win over Flight Sim’s rather perverse way of providing these features.

Lines 600 and 601 apply if FS’s IAS hold mode is currently active (as indicated by the double word at 07DC). Line 600 selects Mach mode instead (i.e. the Sel button swaps modes). Line 601 *sets* the flag for button 15,29 (3860 = 256*15 + 29). This is then used in lines 604–607 so that the do nothing when the Sel button has made this switch—after all, there’s nothing left to do. Well, not quite. More in a moment.

Lines 602 and 603 do the same for a current FS Mach mode setting, toggling it over to IAS mode.

Lines 604–607 test the word offset 0500 for zero to make sure this is not PM, then, *if no switch was made before* (i.e. Flag 15,29 is *not* set), they set the mode, determined by the toggled flag 15,30 (see above), ***twice***.

“Why?” I hear you ask. Well, I ask myself the same question, now, because it doesn’t always (or often) work the way I intended. The idea was that, with no speed or mach mode enabled, I wanted to change the mode shown on the FS MCP. There are no FS controls to do this—the operation on the deafult panels appears to be purely a panel gauge programming thing. Clicking the IAS or Mach button with the mouse changes the display *and* sends the command to FS, but sending the command to FS does *not* change that display! Grrr!

The idea of sending the commands *twice*, not once, was to toggle the mode on, then off again. After all these are toggle controls. This should change the display be end up leaving the mode off, as we know it was before (because otherwise Flag 15,29 would be set).

Well, FS doesn’t appear to cooperate in this. It works okay if the auto-throttle is not armed, but not if it is armed. I have no idea why at present.

Finally, in lines 605 and 606 I toggle another flag bit in 66C4 (bit 5, hex 20, decimal 32), and clear Flag 15,29. The 66C4 bit is tested in Condition 46 (C46) which, as we saw above, is tested to see whether to display FS’s IAS or Mach hold values.

Compared with all of this horrible complication, the PM treatment of the SEL button is extremely simply. In fact it is all in line 619, which simply uses the FSUIPC PM control to press the “Sel” button.

```
609=W0500=0 CP(F+15,30)149,2,C65915,0
610=W0500=0 CP(F-15,30)149,2,C65891,0
```

A little left-over complication. The “Hold” button for speed needs to set the FS speed hold or mach hold according to that toggling flag operated by the SEL button.

```
614=W0500=0 CP(-149,8)149,7,C65580,0
615=W0500=0 P149,8,Cx030007BC,x00000000
```

Button 8 is the A/P disconnect bar. That is actually “on” for disconnection, so when button 7 is operated (the A/P CMD button), it should only switch on the A/P if switch 8 is off.

Line 615 switches off the A/P directly 9writing 0 to the FS offset) when “pressed”—i.e moved downwards.

```
616=W0500 P149,0,C2998,28
617=W0500 P149,1,C2998,25
618=W0500 P149,2,C2998,22
619=W0500 P149,3,C2998,23
620=W0500 P149,4,C2998,24
621=W0500 P149,5,C2998,29
622=W0500 P149,6,C2998,33
623=W0500 P149,7,C2998,32
624=W0500 P149,8,C2998,40
```

All these are for PM operation, and merely equate each of the 9 buttons to an MCP function for PM. The added FSUIPC control “2998” is the PM MCP controls (by parameter(control, and the parameters are values obtained from the list. The only one you may be interested in changing is the ALT HOLD one, button 4. I assign it to the PM parameter 24 (FLCH). If you’d prefer ALT HOLD just change that 24 to 30.

All the rest of the FSUIPC lines (625–712) are wholly concerned with the adjusting knobs, and things get repetitive and boring here. I will just pick out the odd example that may need explaining.

```
641=W0500=0 CP(F-15,30)149,20,C65897,0
…
649=W0500=0 CP(F+15,30)149,20,C65917,0
```

This conditional test on the 15.30 flag is the test for IAS versus Mach again. The FS controls are different.

```
673=W0500 P149,16,Cx62005408,x0167000A
```

Here’s a direct change to one of PM’s MCP values, in this case the heading at 5408. I’m changing these by 10 on the fast turns and 1 on the slow ones. There’s a similar pattern throughout lines 673–712. Not the maximum values included in each. Here it is hex 0167 (359).

```
681=W0500 CP(F-15,30)149,20,Cx22005406,x03E7000A
…
689=W0500 CP(F+15,30)149,20,Cx2200540E,x012C000A
```

Just as in the FS case, we need to change either IAS or Mach depending upon Flag 15,30. You can see here I set the limits to hex 3E7 (999) for speed and 12C (300 = 3.00) for Mach, but I don’t think this is critical. PM applies limits too.

Well, that’s it! That was the last of the examples. If you’ve made it this far, well done! You deserve a medal!

Appendix 1: Full specification of GFdisplay.INI parameters

The INI file contains the instructions to the GFdisplay program. These effectively form a sort of highly specialised programming language. It may look off-putting, but that's really because it is so flexible.

There are a number of sections, with names inside [] brackets, and each section contains lines with a number or name on the left of an '=' and a list of things on the right of the '='.

The identification on the left tells the program what this line is about—for example, which display or indicator light, and the list of "things" on the right are the instructions as to how that should be treated. Rather that "things" I'll call these **elements** from now on.

Each element in a line can be separated from the next by one or more spaces (or tabs or commas). Although, in fact, no such spacing is usually needed by the program, it *is* needed by you. It is important for readability. Throughout this document we are using a space as separator.

The program ignores anything after a semi-colon (excepting a semi-colon in a literal string "…"), so you are free to comment your lines so you can understand them later. You will have seen many comments in the preceding examples.

When processing the INI file, the program will check each line thoroughly, and will not accept any line in error. This will not stop other lines from operating. Lines in error will be written back to the file with an extra comment like this one (for example):

<<< Cannot use forward condition reference

This will be done as soon as the program is started, so your first job, after writing or amending the INI file, is to look at it again, searching for "<<<". You can fix the error and immediately replace the file. You do not need to erase the Error message—if the line is now good, GFdisplay will do that for you. You also do not have to stop the program and start it again. It will notice that you have updated the INI file within a few seconds, and will then re-process it.

Remembering these things about the GFdisplay program will help you debug and test, and revise and develop your GoFlight program a lot faster!

Unfortunately, FSUIPC isn't quite so kind with its button programming. At present, if it doesn't like a line it will simply discard it. I may do something similar to GFdisplay there soon, however. But FSUIPC will never automatically scan for INI changes and reprocess them automatically. That could affect FS performance too much. In order to make it re-read the [Buttons] sections ([Keys] too) you will have to change aircraft.

Sections

The INI file contains a [**GF Connections**] section, generated by the program itself, *not* by you, a [**Conditions**] section and one device section for each device you want to program.

Device sections have names generated by the device type (one of the following list) followed by **.n** where n is the device number, starting with 0. I don't know of any reliable way of identifying which is which other than by trial and error or, possibly, checking in GFconfig. I've never had more than one of any of the units so I can't help further.

The device types recognised are:

GF45
GF46
GFATC *(though support is untested)*
GFLGT
GFMCP
GFMCPPRO
GFRP48
GF166
GFP8
GFT8

Conditions

This section contains a list of 'tests', each of which is at any time either 'true' or 'false'. These tests don't do anything in themselves but can be used throughout the rest of the real unit programming, whenever decisions need to be made about what to display, how to display it, and so on.

Like most things in computers, Conditions are based on numerical values. Where no comparison is explicitly made it is implied to be “not equal to zero” (!0). Thus, the notion of ‘true’ applies for any non-zero result, whilst ‘false’ is only derived for a zero result. Fractions are ignored in this decision, so any result in the range

$$-1.0 < x < 1.0$$

is treated as zero and therefore ‘false’.

Condition lines are numbered from 0, and the maximum condition number is 254. The order is unimportant *except* where you use conditions in condition lines. To prevent recursive loops, a condition referred to in a condition must have a lower line (condition) number than the current one—i.e you can only refer backwards.

References to conditions are by

Cn for testing ‘true’, and **!Cn** for testing ‘false’, where **n** is the condition number (line number).

In all other respects, Condition lines are similar to the other programming lines for the units as defined below. In terms of a direct likeness they are similar to the Light program lines, because a Light (or LED indicator) is either ‘true’ (lit) or ‘false’ (extinguished) just like conditions. The only thing extra to Display lines is that they can contain literal strings or numbers for direct display.

Any **Condition** or other programming line can contain up to 8 ‘true’ conditions (**Cn**) and up to 8 ‘false’ conditions **(!Cn**). But note that conditions alone do *not* provide a ‘result’. In other words, to light an indicator you cannot simply have a condition or list of conditions. The conditions just tell GFdisplay whether to process the rest of the line. They do *not* result in a value for the line.

Needs and Brightness

There are two special optional parameters for each device. One to set the brightness level, and the other just to list special ‘needs’ for the device’s displays to show anything at all.

The Brightness specification is simply:

B=<value>

Where the <value> part is either a literal numeric, 0–15, or something which evaluates to a numeric—a reference to an FSUIPC offset. Using the free offsets (66C0–66FF) you can implement a brightness control on a spare rotary knob, or simply a single button implementing a cyclic increment on a Byte offset, with 15 as the limit.

The Needs specification is:

Needs=<list of needs>

Currently the available checks here are:

V<n>	The battery voltage must be at least <n> (an explicit numeric)
B	The battery must be switched on.
E	The electrical subsystem must not have been ‘failed’.
A	The Avionics must be switched on.

Other all-on and all-off type conditions may be added here in future.

Display and Light numbering

The Displays and Lights on the GF units are numbered from 0 upwards. With displays this seems to be from left to right, top to bottom. With Lights it varies. These are the ones I know:

GFLGT:
- 2 = right red
- 3 = right green
- 4 = left red
- 5 = left green
- 6 = nose red
- 7 = nose green

GF166:
- 0 = centre
- 1 = left
- 2 = right

GFMCP:
- 0 = NAV
- 1 = Hdg HOLD
- 2 = IAS/Mach HOLD
- 4 = Alt HOLD
- 5 = APPR
- 6 = B/C
- 7 = A/P CMD

GFMCPPRO:
0 = VNAV
1 = LNAV
2 = CMD A
3 = CMD B
4 = A/T Arm
7 = VOR LOC
9 = CWS A
10 = CWS B
14 = Flight Director Left
15 = N1
16 = SPEED
17 = LVL CHG
18 = HDG SEL
19 = APP
20 = ALT HOLD
21 = V/S
23 = Flight Director Right

The others all follow the principle of numbering left to right, 0–7 (maximum is 7 in all cases).

In the lines programming the devices and lights you normally identify the light or display on the left by:

L<n> = …	for example L1=1 switches Light 1 on unconditionally
D<n>= …	for example D2= ="PETE" displays PETE (as best it can) on display 2

This applies to a simple unconditional result, or one where the display is simple based on the content of an offset. Where you want to display different things in different circumstances, you need more than one line for the Light or Display. In this case you have a sequence of lines, numbered like this (for example:

```
D1.1= C3 C4 ="CASE1"
D1.2= C3 !C4 ="CASE2"
D1.3= !C3 C4 ="CASE3"
D1.4= !C3 !C4 =""
```

And so on. The program will process each line in order of the sequence number (.1 .2 .3 …), skipping all those for which the total result of the Conditions is FALSE, and processing the first it finds with either no conditions, or ones which evaluate to all TRUE. Once such a line is executed for this Light or Display, remaining lines in the sequence are skipped.

Note that in the sequence above all four possible combinations of the two Conditions have some action programmed for them, so aone of those lines will always be executed. If you had left off line D1.4, say, then if both conditions C3 and C4 are FALSE *nothing will be done with this Display!* In other words, it will simply be left in whatever state it was in on the last pass.

To avoid cases like that, where one or more sets of possible conditions are omitted, it is a good idea to have a final line with no conditions at all, acting as a default. Maybe it blanks the Display or extinguishes the Light, or displays an error. In the above example, the D1.4 line could be, more safely perhaps:

```
D1.4= =""
```

Elements

Here I present a reference list of all the different ‘element’ types you can use. Some are only valid in Display lines and others are mandatory in Light and Condition lines, but in general the ordering is not relevant and almost every element type is optional throughout.

Literals

By “literals” I mean values or strings that simply mean what they say. The literals supported are:

n	decimal number such 53, or 1.414, or .23. You can precede it with + or –.
='xxx'	string of up to 8 characters, not including the single quote (')
="xxx"	string of up to 8 characters, not including the double quote (")
STR'xxx'	string of up to 8 characters, not including the single quote (') (*alternative form*)
STR"xxx"	string of up to 8 characters, not including the double quote (") (*alternative form*)
=Xxxx	hexadecimal literal (or a comparison, depending on context)

Note that the ‘=’ or ‘STR’ preceding strings is strictly only needed when there are no other value-defining elements present. However, it is safer and more consistent to use them all the time.

Xxxxx: Value from FSUIPC offset

This provides the hexadecimal offset value for an FSUIPC value to be displayed or tested. For FSUIPC-provided values you need to refer to the list in the Programmer’s Guide (part of the FSUIPC SDK). For Project Magenta you can find a list of

FSUIPC offsets on the PM documentation website. Other projects using FSUIPC offsets may provide their own. You can also use the free user offsets 66C0–66FF for general purposes such as counters, selectors and flags, as extensively shown in the examples earlier.

The value from the offset should always be qualified by a **type** as defined next, and can be operated upon by masks, multipliers and divisors, all described below.

A variation, to allow offset values to be obtained from an offset provided by another offset is:

IXxxxx

Where the hexadecimal offset 'xxxx' here tells the program where to find the offset to be used. The type still applies to the eventual value—the indirect offset must always be an unsigned 16-bit word.

Value types

When a condition or display value is to be derived from the contents of an FSUIPC offset, the **type** of value should always be declared. The element which does this will be one of these:

S 8:	Signed 8-bit byte	1
U 8:	Unsigned 8-bit byte	2
S 16:	Signed 16-bit word	3
U 16:	Unsigned 16-bit word	4 (also 0)
S 32:	Signed 32-bit double word	5
U 32:	Unsigned 32-bit double word	6
F 32:	Floating point 32-bit ("float")	7
F 64:	Floating point 64-bit ("double")	8
STR:	Character string (ASCII)*	9
R 0:	Transponder setting XXXX	10
R 1:	ADF frequency XXXX.X	11
R 2:	COM/NAV frequency 1XX.XX	12

You can also specify the type as being provided indirect, by a 'U8' offset value. To do this, specify the type as **Ixxxx**, where xxxx is the hexadecimal offset. The values to select each type are shown in the list too—1 to 12, defaulting with U16. This facility also features in one of the more complicated examples earlier.

Mxxxx: Mask

This element provides a hexadecimal value to be used as a mask on the value selected by an **Xxxxx** or **IXxxxx** element. This is most often used to test individual bits in an array of flag bits. The result of an offset value and a mask is the logical AND of the two.

*n Multiplier, /n Divisor:

These two allow conversions of the value obtained from the offset by simple multiplication and division. The values 'n' must be decimal literals (see earlier) and the computations are performed in floating point. Only one of each, at most, is allowed—you have to combine multipliers and combine divisors if you need more.

=, !=, <, >, <=, >= Comparison:

For Conditions and Lights you won't normally want a final computed value, so much as some condition on that value.

GFdisplay supports these conditions, which all operate on the value *after* the Mask, Multiplier and Divisor have done their work. The literal values here can be any of those defined earlier except strings.

=literal	equality
!=literal	inequality
<literal	less than
<=literal	less than or equal
>literal	more than or equal
>=literal	more than or equal

Where no comparison is applied and a true or false result is needed (i.e. in Lights and Conditions), the !=0 comparison is assumed, but then any value in the range –1.0 < n < 1.0 is considered to be zero.

A"str" and AA"str": AdvDisplay String Comparisons

A special facility enables indicators and displays to be changed according to the messages being displayed via the "AdvDisplay" text area at FSUIPC's offset 3380. This is specifically aimed at Radar Contact users, but could relate to any program using the display facilities for options or menus.

The form **AA"str"** looks for an exact match for the given string of characters *anywhere* within the AdvDisplay text. If it is found, the result is the value 1 (i.e. True). If not, the result is 0 (False).

A"str" requires the string to occur at the beginning of a line. This would be the more usual form for Radar Contact where the facility is being used to light or flash indicators representing the options being presented. The characters at the start of the line will represent the button combination being prompted (e.g. "CS1" for Control+Shift+1). Using a Go-Flight P8 module the LED next to the button programmed for CS1 could be lit or flashed to show this option was currently available.

In both cases the maximum length string is 8 characters, any excess being ignored.

Dxxxx: Numerical display format:

Number display formats are defined by "D-Xifs" where

-: is omitted for unsigned numbers and hexadecimal, or for automatic signing by type

X: is omitted for decimal, present for hexadecimal (*no signs*)

if: are values giving the integral and fractional places. If the number of integer places is given as zero it is allowed to vary as needed, with preceding spaces if needed. If the number of fractional places is zero then only the integer part is shown.

s: is omitted for normal right-alignment in the display, or included as a decimal digit specifying the number of spaces in the display to be left *after* the value. This can be used to left-align or centre the value.

For signed numbers the place for the sign is not included in the counts. Remember that there are limited numbers of places in the displays. For the GF166, for example, there are 6 digits in each, so i + f + s must be less than or equal to 5 for signed numbers (allowing room for the sign), or 6 for unsigned. The decimal point doesn't use a digit position.

If a value is too large for the display, GFdisplay will show something like "8.8.8.8.8.8." to indicate overflow.

Examples:	DX40 for "66C4"
	D-14 for " 0.2345" and "-1.0567"
	D-50 for " 32767" and "-32768"

Rn: Radio frequency display format:

There are three special format codes for Binary-Coded Decimal (BCD) values used for frequencies and transponder codes:

R0	a BCD radio frequency with no fractional part—the transponder (XXXX).
R1	a BCD radio frequency with one fractional digit—an ADF (XXXX.X).
R2	a BCD radio frequency with two fractional digits—COM and NAV (1XX.XX).

Flashing and delayed switching

There are several ancillary codes which can be added to a Display or Light definition for special behaviours. These are:

Fslow	Flash the display slowly
Ffast	Flash the display faster

Both of the above can be used on any Display orLight. The rest are only applicable to Lights:

Fs*n*off	Flash slowly for *n* seconds, then leave it off. 0 < *n* < 60
Fs*n*on	Flash slowly for *n* seconds, then leave it on. 0 < *n* < 60
Ff*n*off	Flash fast for *n* seconds, then leave it off. 0 < *n* < 60
Ff*n*on	Flash fast for *n* seconds, then leave it on. 0 < *n* < 60
St*n*off	Display stays on steady for *n* seconds then goes off

These all apply only when, without the flash/delay option added, the display would have been enabled/lit.

Appendix 2: An example for [GFMCPPRO.0]

The example following was derived from the one in the main text for the GF-MCP, and tested with the updated version of GFDisplay, by Steve, known as steveo38 on the Forum. Thanks Steve!

```
[GFMCPPRO.0]
Needs=V16 B E A

B=10 ; Not too bright?

D0.1=C5 XC4E U16 D30 ;NAV1 CRS as nnn
D0.2=!C5 "360"
D1.1=C0 C48 "" ; PM blank Spd
D1.2=C0 !C48 C1 X4E8 U16 /100 D02 ; PM's Mach as (n).nn
D1.3=C0 !C48 !C1 X4E0 U16 D30 ; PM's IAS as nnn
D1.4=!C0 C46 X7E8 U32 /65536 D02 ; FS's Mach as (n).nn
D1.5=!C0 !C46 X7E2 U16 D30 ; FS's IAS as nnn
D2.1=C0 C6 X4E2 U16 D30 ; PM's HDG as nnn
D2.2=C0 !C6 "360"
D2.3=!C0 C7 X7CC U16 *360 /65536 D30 ;FS's HDG as nnn
D2.4=!C0 !C7 "360"
D3.1=C0 X4E4 S16 *100 D-50 ; PMs Alt as (-)nnnnn
D3.2=!C0 X7D4 S32 *3.28084 /65536 D50 ; FSs Alt
D4.1=C0 C47 "" ; PM blank VS
D4.2=C0 !C47 X4E6 S16 *100 D-40 ; PM's V/S as (-)nnnn
D4.3=!C0 X7F2 S16 D-40; FS's V/S as (-)nnnn
D5.1=C5 XC4E U16 D30 ;NAV1 CRS as nnn
D5.2=!C5 "360"

L2.1=C0 X4F0 U16 M0001 ; PMs AP Master 1
L2.2=!C0 X7BC U32 ; FS's ap eng CMD A
L3.1=C0 X4F0 U16 M0001 ; PMs AP Master 1
L3.2=!C0 X7BC U32 ; FS's ap eng CMD B
L4.1=!C0 X810 U32 ; FS's Auto Throttle Active
L7.1=C0 X4F0 U16 M0020 ; PMs LOC -> NAV light
L7.2=!C0 C8 X7C4 U32 ; FS's NAV acquired
L7.3=!C0 !C8 X7C4 U32 Fslow ; FS's NAV (flash)
L14.1=!C0 X2EE0 U32 ; FS's Flight Director Active
L16.1=C0 X4F0 U16 M0200 ; PMs Speed
L16.2=!C0 !C2 X7DC U32 ; FS's IAS -> SPD
L16.3=!C0 X7E4 U32 ; FS's Mach -> SPD
L18.1=C0 X4F0 U16 M0080 ; PMs HDG
L18.2=!C0 X7C8 U32 ; FS's HDG
L19.1=C0 X4F0 U16 M0010 ; PMs APP
L19.2=!C0 C4 X800 U32 ; FS's APP LOCKED
L19.3=!C0 !C4 X800 U32 Fslow ; FS's APP Pending (flash)
L20.1=C0 X4F0 U16 M0008 ; PMs ALT
L20.2=!C0 X7D0 U32 ; FS's ALT
L23.1=!C0 X2EE0 U32 ; FS's Flight Director Active
```