

WideFS: Network application interface for Microsoft Flight Simulator

by Pete Dowson, 28th July 2006

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

Version 6.70

WIDEFS IS NOT FREE!

WideFS 6 will *not* run on Flight Simulator without FSUIPC also being installed, and at least Version **3.53** of FSUIPC is required. WideFS **must** be registered with FSUIPC, using an Access Key, before it will work. You can purchase an access key from http://secure.simmarket.com/product_info.php?products_id=539. Full details about purchasing access keys and registering both FSUIPC and WideFS are also to be found in the early pages of the FSUIPC User Guide. You can save money if you purchase keys for FSUIPC and WideFS at the same time.

This package contains just the following parts:

| | |
|----------------|---|
| WideServer.dll | The server module itself, for the FS Modules folder (<i>Version 6.70</i>) |
| WideServer.ini | Initial configuration for the server |
| WideClient.exe | The client program, running on client PCs (<i>Version 6.70</i>) |
| WideClient.ini | Initial configuration for the clients |
| WideFS.doc | This document, in Word 97 format |
| WideFS.pdf | This document, in Acrobat format |

Note: All my Windows based FS software is always available in the latest versions from <http://www.schiratti.com/dowson>

IMPORTANT NOTES

This is NOT a program for linking several copies of Flight Simulator, nor will it allow scenery or cockpit views on multiple PCs. It is purely for running EXTERNAL applications on a separate PC whilst still communicating with FS as if on the same PC. For such applications to work through WideFS they need to access Flight Simulator through the interface provided and defined by FSUIPC (or originally, for FS98 programs, FS6IPC).

Also be sure to match versions of WideClient and WideServer. This is often essential for correct connections, as the protocol being used is developed and may be subject to subtle changes in each new version.

Introduction: What is WideFS?

There are now many add-on programs written to interface to Microsoft's Flight Simulator. Nearly all of these exchange information with FS by using an interface devised many years ago (originally for FS95, the first Windows version of FS) by Adam Szofran. This interface provided methods for programs running outside the Simulator, but in the same PC, to read various values from FS and write different values back. This enabled a wealth of programs such as moving maps, weather control, external instrumentation, extended autopilots and even flight management systems to be developed.

The interface, using a technique known as "Inter-Process Communication" (IPC), was extended by Adam to operate on FS98 (using a module called FS6IPC—the Microsoft internal name for FS98 was FS6). But this was the last in that line as Adam was understandably recruited by Microsoft to work from within instead of without. <G>

It was during this time that I put together my first PC network. An excellent package called 'WidevieW' became available, written by Luciano Napolitano, which allowed several installations of FS on different PCs across a Network to show different outside views from the same user aircraft in flight. I used this as a basis for making Chris Brett's EFISv2 run on a separate PC from the Flight Simulator to which it was interfacing. This was done by fooling the remote copy of WidevieW into thinking it was running with FS, whereas it was actually a program of my devising (the 'client'), and similarly another program intercepted the data at the Simulator end (the 'server'). This package was released as freeware under the name WideEFIS, for obvious reasons.

Not much later I learned a little more about network programming—but not much. Mostly I just re-used the simpler code examples in the Microsoft development kits. I used the IPX/SPX protocol because it looked simpler to understand, was fast in operation, and also because WidevieW used it too. Applying all this resulted in WideFS, a package consisting of two parts:

1. **WideClient.exe**, a program which on the one hand provides what looks like an exact replica of the FS6IPC interface, fooling FS application programs into interfacing with it thinking that it was FS98, and on the other transmitted these exchanges and received data back across the Network, using the SPX (and later TCP) protocol.
2. **WideServer.dll**, a Flight Simulator module, installed into and actually running as part of FS98, which serviced the messages and requests from WideClient and interfaced directly into the same data inside FS as did FS6IPC.

When FS2000 came out I wrote a new IPC module for that simulator, but kept not only the same interface as that devised by Adam, but also the same variable access. This was deliberate, to allow programs written for FS98 to run with FS2000 unchanged. I also made it run on FS98 itself, and extended it to CFS1, CFS2 and FS2002, keeping as far as possible complete compatibility with that FS98 standard. Since this was now a ‘universal’ IPC interface for Flight Simulator I called it FSUIPC, breaking away from Adam’s specific FS version implications.

Nowadays FSUIPC has taken over the interfacing role inside FS, and WideServer interfaces to it, so both parts are needed even if all the application programs are running on client PCs. This is because with FS2000 and then more so with FS2002, access to many of the internal parts of FS, which was so easy in FS98, has become very difficult. Internally it is no longer a matter of simple memory offset and size management and mapping, but often also involves calling procedures elsewhere in FS, and following pointer chains into data private to other modules. Rather than duplicate such complex code, it is now all concentrated into FSUIPC, freeing the WideFS parts to deal predominantly with the efficient running of the Server to Client (or Server to many Clients) exchanges.

So, does this answer the question, ‘what is WideFS?’ Maybe, if you’ve followed its history well enough. Otherwise, for now, just consider WideFS as a package that enables you to run Microsoft Flight Simulator applications on computers which aren’t even running Flight Simulator.

And **please** don’t confuse it with WidevieW. The similarity in the name is because of its history, as described. Both packages deal with spreading the simulation experience across several PCs in a network—hence the ‘Wide’ part. But you use WidevieW for multiple views, to see around you, and you use WideFS for additional instrumentation, moving maps, and weather control programs, running on separate PCs and lessening the burden of having too many processes on the all-important Simulation PC itself.

Good flying!

Installation

Follow these steps:

On the Flight Simulator PC

1. Place WideServer.dll and WideServer.Ini in the Flight Simulator Modules folder, which is inside your main FS folder.

That's it. FS will load it automatically for you.

Note that you must also have FSUIPC.dll (version 3.53 or later) in the Modules folder. WideFS is totally dependent upon that module, which is available as a separate download. For full use of all current WideFS features you should be using FSUIPC version 3.617 or later.

When you run FS, go to the FSUIPC options (Modules menu, FSUIPC entry), and **Register** your WideFS by entering the access key. (You *did* get one, didn’t you? If not, use the link given above).

For full details of this process, please see the FSUIPC User Guide. Note also that you have to close down and restart FS in order for the registration to become effective. You don’t have to repeat this each time, though. Just keep a safe copy of the FSUIPC.KEY file (from the Modules folder). You may have to re-register if you re-install Windows, or want to use FS on another PC. But you are not restricted to using WideFS in only one FS installation.

On each client PC

(i.e. all the other PCs in your Network on which you are NOT going to be running FS itself, but on which you want to run some application which interfaces to FS).

1. Place WideClient.Exe and WideClient.ini in any desired folder (I recommend either in the same folder as your FS utility, or a dedicated folder called WideClient).
2. Also install here any bitmap you want WideClient to display as background for your utilities (i.e. replacing the FS panel for which they may have been designed). If you do want to use a bitmap, you will need to edit the WideClient.ini file and add the bitmap filename using the "Background=filename" parameter. (See the section on WideClient parameters, below).
3. If you are using GoFlight equipment, and wish to connect this to the Client PC and program the buttons and dials in FSUIPC's "buttons" page, you either need to do the full install of GoFlight's recent driver release (the GFConfig program included), *or* extract the GFDev.dll from that and place this in the same folder as WideClient.exe. Alternatively, if you have a full GoFlight installation on your Server PC, find the GFconfig program folder, and copy over the GFDev.dll from there into the WideClient folder. Note that earlier versions of GoFlight's installation did not install GFDev.dll, so you may need to get an update first—a suitable version can be found on my Support Forum too.
4. Drag a shortcut for WideClient onto your desktop.

Now, when you have your Network correctly configured, you can either run WideClient (from the shortcut), THEN your utility program, or you can, for more convenience, edit the WideClient.ini file and enter the full pathname for your utility in the Run1= parameter line. WideClient will then start your application for you. Up to three programs can be started in this way—see the section on WideClient parameters later.

Configure your Network

WideFS uses TCP protocol (part of TCP/IP) by default, but also supports UDP (also part of TCP/IP) and SPX (part of IPX/SPX). Just ensure the TCP/IP protocol is installed on all PCs (as it will normally be by default) and you will be set. IPX/SPX may be more efficient but it is difficult to recommend these days as Microsoft do not seem interested in maintaining it nor making it easy to use.

The Server automatically operates with any clients which connect, whether they be using TCP, UDP or SPX. You can mix them, to suit the specific individual client PCs. Better, leave the protocol choice out of the Client INI file altogether and just use "ProtocolPreferred= ..." in the Server INI to select your choice. Details of these parameters are given below.

To use TCP/IP in a client you shouldn't need to do anything to either Server or Client configurations providing both are running Windows XP or 2000. However, if you find the connection is not happening, or if you have more than one Server and it is connecting to the wrong one, you need to add this line to the [Config] section of the WideClient.ini file (add the [Config] section too if your ini file hasn't even got one!):

```
ServerName=NameOfServer
```

The ServerName is the name you gave to the PC on which you will be running Flight Simulator with WideServer. If you prefer you can use the Server's IP address instead of the name, thus:

```
ServerIPAddr=192.168.0.3
```

But I'd recommend only ever using names. If both Name and Address are provided only the latter is used. Note that if the IP Address for the server is not one of those treated by Windows as 'local', you may find WideClient attempting to connect via your Internet connection. I'm not really sure how all that works, but you should be safe if all your networked PCs have assigned IP addresses in the form "192.168.0.N" where N runs from 1 to 255. To assign IP addresses (at least in Windows 98—NT and so on may be different)—open up the Network Properties dialogue, find and highlight the line mentioning "TCP/IP" bound to your local Network adapter (e.g. "TCP/IP->3COM Ethernet Adapter"), then click the Properties button. In the IP Address tab, select "Specify an IP Address", then set the IP Address to 192.168.0.N (N numbering your PCs) and the subnet Mask to 255.255.255.0.

Incidentally, assigning specific IP addresses to your PCs in this way also reduces periodic but regular stutters in Flight Simulator, caused by the Network drivers querying the Network to get an address assigned (or something like that).

Please be aware of possible problems, especially when using TCP/IP, which may be introduced by Routers and Firewalls, whether hardware or software. Some firewalls (ZoneAlarm is one) will need you to give WideClient permission to connect to WideServer, and maybe vice versa. This may be needed each time you update the WideFS program, as it will be seen as a change and therefore a new potential danger. Do not think that closing the front-end of such programs solves this—it may make it worse. Best to officially tell the firewall that you are happy with the WideFS components. You may also find the web site www.helmig.com useful in solving Network problems, whether with IPX/SPX or TCP/IP.

What's this about UDP?

UDP is simpler and faster than TCP, and is installed on your PC as part of the TCP/IP package. It differs from previous protocols supported by WideFS in that it is "connectionless"—data is sent to specific addresses (by IP address), but there are no checks on whether they get there and no error recovery to ensure they do. This is why it is faster, and also why I've not supported it till recently. With the block sequence and checking now in WideFS you should at least be able to detect if it is going wrong.

Note that the method of selecting the protocol to be used is changed. The old WideClient "UseTCPIP" parameter is gone. If that was previously set to "Yes" (the default) then nothing replaces it if the Server isn't specified, and the protocol is chosen automatically by the Client at run time, trying TCP, SPX (the part of IPX/SPX actually used) and UDP in turn.

With a WinXP server and a WinXP client, you can add "ProtocolPreferred=XXX" (XXX being TCP, SPX or UDP) to the [Config] section of the WideServer.INI file, and those clients not otherwise committed will use that protocol if possible. This gives you a quick way of changing the protocol on all computers in the Network, for performance comparisons.

If you wish to select a specific protocol on any client, just add "Protocol=XXX" (XXX being TCP, SPX or UDP) to the [Config] section of the WideClient.INI file

Connection arrangements

With the automating of WideClient linking to WideServer on Windows XP and 2000 systems, there are several permutations according to the manually settable options, as follows:

Case A: Server not specified in Client INI (In other words, no ServerName, ServerIPAddr or ServerNode parameters).

In this case both the WideServer PC and the WideClient PC must support mailslots (i.e. run Win2K or XP), and the "AdvertiseService" parameter in the WideServer INI must *not* be set to "No" (it defaults to "1", for 1 second intervals).

Sub-case A1: Protocol is set in the Client INI:

This protocol is the one that will be used by the Client. If it isn't installed you'll get an error.

Sub-case A2: Protocol is not set in the Client INI:

The protocol tried first is the one specified by the Server in "ProtocolPreferred". If there is none, TCP is assumed, but in either case, the client will try others if the initial one is not installed (in order TCP ... SPX ... UDP, cyclically).

Case B: Server is specified in Client INI (In other words, at least one of ServerName, ServerIPAddr and ServerNode parameters are provided).

In this case the Client can work with or without Server mailslots operating to this Client PC. This depends on whether the protocol is set:

Sub-case B1: Protocol is set in the Client INI:

Broadcasts from the Server are ignored altogether. This protocol is the one that will be used by the Client. If it isn't installed you'll get an error. If the Client cannot find the specified Server you'll get an error.

Sub-case A2: Protocol is not set in the Client INI:

Broadcasted mailslots from the Server are needed. Only broadcasts from the specified server will be accepted—this prevents the client from connecting to a different Server when there are more than one.

The protocol tried first is the one specified by the Server in "ProtocolPreferred". If there is none, TCP is assumed, but in either case, the client will try others if the initial one is not installed (in order TCP ... SPX ... UDP, cyclically). This is true no matter which Server identification system is used (name, address or node).

Running WideFS

WideClient can be up and running even when there's no WideServer program elsewhere on your network to serve it. It will simply keep trying to connect until one is ready. Similarly it will re-connect if a server is closed and re-opened.

Likewise, WideServer is very tolerant of clients starting and stopping. Each server can serve many clients, but each client only talks to one server.

Simply run Flight Simulator on the server machine (it will load WideServer), and run WideClient (plus your utility program(s) on your other machines. That is all there is to it! Apart from some possibly displays in Flight Simulator's title bar there will be no other sign in Flight Simulator that WideFS is doing its job, and you should not notice any affect on frame rates at all.

With FS2002 and FS2004 you can set a limit on the Frame Rate which FS will keep to. I have found that WideFS operates much more smoothly and efficiently if you use this to prevent FS running away with the processor. With a reasonably fast machine, say 2 GHz and more, try frame rate limits of 30–40 (20–30 for FS2004), but reduce this on slower machines, down to about 15–20 on 1 GHz PCs. For FS2004 on machines of 3 GHz or better, and with no slow or overloaded client PCs, you could try running with the frame rate limiter set to unlimited. This may give rather uneven performance, but this may not be too noticeable, depending on the client PC capabilities.

Note that the number of current connections is shown in the title bar of Flight Simulator. This may fluctuate if responses to clients aren't currently being sent. This is because the client assumes that a lack of response for many tries means a disconnected link, so opens a new one. The server detects disused links after a while, and closes them, but there's an overlapping period whilst more connections may actually be open than there are clients.

Some programs (Chris Brett's EFIS and John Hnidec's Moving Map are examples) place their windows *within* the Simulator window, or now, in this case the WideClient window. But most utilities don't actually need WideClient visible at all—you can minimise it (or simply set an INI file parameter to do this).

However, taking EFIS as the example, when it is up and running, it will place its Windows inside the blank grey window provided by WideClient, or over an optional background bitmap which you can set up via the INI file. You will want to resize the windows to suit your screen and desired layout of EFIS. Click on the FMC button to get the FMC window and position that as you like. I find the LARGE size of the main EFIS window best. The parameters in the files as provided suit an 800x600 resolution screen setting. The window sizes and positions will be remembered for next time.

Once you have everything running you can read the sections towards the end of this document, detailing the WideServer and WideClient INI file parameters, to see what adjustments you may wish to apply to get the best performance for your set up. However, please note that many hours have gone into optimising the performance and the defaults, in general, are the result. If you experiment and make things worse, just delete all the changes you made and the programs will reset the defaults.

Questions and Answers on how WideClient handles data

Q: Is WideFS a simple passthrough conduit to FS, or does it cache data locally?

A: WideClient maintains a memory map of all of the locations ever requested since it started running. When the values are requested by the client applications, it gets data from there and gives it to the client in a direct response. If there are data items which have not been requested before, it also sends appropriate requests to WideServer, whilst supplying the default value in its memory to the applications (this would be zero). There is an option in the INI (**WaitForNewData**, see below) which actually stops this return being made until WideServer has actually sent the newly requested data—this is actually enabled by default with a 500 mSec timeout. See the DOC.

Except for Write requests from clients, the Network traffic is totally controlled by WideServer, which maintains details of all data items requested by each client, separately, and monitors these for changes. This latter is done at FS frame rates. Only changes are sent out. If a connection dies or closes, the list for that client is cleared (by both ends) and the process starts over.

Q: Is there any recommended polling frequency for applications using WideFS?

A: No. It doesn't really matter, but if you are operating something graphical to run at FS speeds then you probably should try to match average frame rates, for smoothness of your displays. Except on really powerful machines (those capable of running FS with ease), WideServer usually works better if you limit the FS frame rates to a bit less than its average performance in any case (as mentioned elsewhere in this document), so you would, say, set the limiter to 35 or 30 or 25 fps (according to processor) and poll WideClient at that sort of frequency.

Of course, if your program does a lot of processing or heavy graphics, or is sharing the client PC with other such applications, you might not be able or want to achieve such a frequency.

Since WideClient is supplying all values from its memory, directly, there's no benefit from splitting your data requests into more or less frequently needed items. WideServer will be sending all the changes anyway. If you poll some less frequently you will just be skipping some changes. Of course it is *not* the same for writes to FS. They need more consideration as they will all result in LAN blocks to WideServer and require FS processor time when there.

Q: Is there a log setting I can use to see if I am thrashing WideFS?

A: You don't have any control over the Network operations, excepting how you write things. Certainly you should optimise writes. Don't keep writing the same values to the same places, only write what you need to write, and don't write that frequently that things bog down. But when you are reading you are not affecting anything on the Network. Provided you don't actually ask for any data you don't need (for once you have it will be monitored for changes and all changes sent), then it doesn't matter. However, don't take that sort of optimisation to extremes either. If you ask for every other byte only, for instance, the overheads of comparing them all separately and blocking them all up with their own red tape, will far exceed the saving from not sending the intervening bytes which you don't really need. In other words, try to apply some common sense.

Of course, if your program is also supposed to run well on the FS PC you have to consider the affect you may have on FS's performance. But there are a lot of other factors there apart from calling the IPC interface.

Error reporting

Unless there is something really bad preventing WideClient from running, it will normally keep trying. Any errors it gets are described in text form in a Log file ("WideClient.log") which you will find in the same folder as the EXE file you are running. Please check there if you see WideClient still waiting for a connection when you believe it should be connecting.

Except for an incorrect or missing Registration, a serious error which prevents WideServer from offering a service will result in a message box appearing whilst Flight Simulator is getting ready. Pressing OK should allow you to continue using FS, but the WideFS link will not be operating. If you configure a "hot key" to restart WideServer (see **RestartHotKey** below, in the section on WideServer.ini options) then you can make WideServer try the whole initialisation sequence again. But really if the error is that serious you will probably have to close FS and sort out the network error being reported in the WideServer.log file (find this in the FS Modules folder).

If you have not yet registered WideFS with FSUIPC, the server will not start but will not have any adverse affect on FS operations. You will see a message in the WideServer.Log file reporting the registration problem.

Just about all of the errors that you are likely to see there will be due to network problems. I cannot give a precise list of messages with their meanings here—WideFS is merely reproducing the error number it gets from Windows software ("WinSock"), and the text is also direct from Microsoft's own documentation. If you get stuck with any real errors, check through ALL the points listed in the trouble-shooting section below.

If you write to me with any problems to look at (and I sincerely hope that this won't be needed, as everything I know is written down here somewhere), please attach a Zip file containing both the WideServer and WideClient LOG and INI files.

Notes for trouble-shooting

If you want to use IPX/SPX then be warned: the Windows software is not so user-friendly in this area, and seems to be getting less so. One must assume that Microsoft is neglecting this protocol these days. In particular, if either or both Server and Client PCs are running under Windows NT or its successors Windows 2000 or XP, then some additional configuration is almost always needed.

These steps may also prove useful with Windows 95, 98, 98SE and ME. Please try ALL of them before seeking more help. I'm afraid I really know very little about Networks as such, and all the hints listed here are actually contributed by other users.

1. NETBIOS may need to be enabled for IPX/SPX

You may need to enable NetBIOS over the IPX/SPX connection. Why? Sorry, I don't know, but this seems to be especially important in Windows 95 or 98 if you are running a mixed Windows network.

To do this in Windows 95/98, go into the Network application in Control Panel, select the IPX/SPX protocol, and click Properties. There will be a tab entitled NetBIOS. Select that, and then make sure “I want to enable NetBIOS over IPX/SPX” is selected.

I am not sure if this is also needed in Windows NT, 2000 or XP, nor exactly how you'd do it in those systems. But it is something to check if you cannot get a connection. One successful user has reported that on his working set up NetBIOS is *not* enabled. This is under Windows 98, so this may be an important difference to the above recommendation.

2. Server Node may need specifying for IPX/SPX

Run Flight Simulator with the WideServer.dll module installed, and then look at the WideServer.Log file that you'll find in the Modules folder. Very near the start there should be a line reading something like

```
17248 ServerNode=0.0.49152.1264.9490
```

The actual numbers won't be the same, but you should find the line okay. Now insert the parameter 'ServerNode= ...', quoting exactly the same numbers after the =. into the [Config] section of each WideClient.ini file you are planning to run with IPX/SPX protocol to link to this Server.

On Windows XP there appears to be a little problem in identifying the correct ServerNode sometimes. Windows XP seems to add something called an IPXLoopbackAdapter, which has its own Node, and it is this which WideServer sometimes sees first. When this happens, WideServer tries to recognise it and you may get Log entries like this:

```
88218 ServerNode=13330.61389.0.0.512
88218 *** WARNING! *** This ServerNode is likely to be incorrect! Running IPXROUTE to get list ...
88218 IPXROUTE config >"G:\FS9\MODULES\WideServer.ipx"
90375 ... The correct value will be one of these:
90375 ServerNode=0.0.3072.8310.62813 ("Local Area Connection")
90375 ServerNode=0.0.37578.21024.21313 ("NDISWANIPX")
```

In this case the first ServerNode found was, indeed, the one for the Loopback Adapter. The correct one in this case was the one for the Local Area Connection. (Please don't ask me what an NDISWANIPX is, as I have no idea!)

Note that you will have to re-check this if you change LAN adapters in the Server PC, or make other similar configuration changes. Also, please see item 7 below, concerning another potential problem with some LAN drivers providing the wrong Node identifiers (though this may now be resolved by the IPXROUTE selection shown above).

3. Network Address may need setting for IPX/SPX

I have heard from one chap using a mixed network (Win2000 and Win98) who needed to do a bit more before it would work. To quote him:

“On the Windows 98 PC, under Network Properties, IPX properties, there is an Advanced settings tab. In that list there's something called Network Address. That must be set to a number other than zero. I just typed 13579 (or something like that), rebooted, and now it works.”

I'm afraid I can't add anything to that.

4. Ensure IPX/SPX is associated with only one device on each PC

Another user reported that “I had to turn off the IPX protocol on the cable modem network card making sure the one for the LAN was enabled”, so it seems likely that IPX/SPX, when added to more than one device, can produce routing problems.

Generally, when you add a protocol in the Network part of the Control Panel, Windows adds it without exception to *all* installed network-capable devices. You will need to go through and remove it from everything but the Network adapter.

5. Connections and Sockets

Check that the parameters 'Maximum Connections' and 'Maximum Sockets', which you'll find under the protocol's Properties—Advanced, are both set to a larger number, like 128. The default limits in Windows ME (in particular), and possibly other versions of Windows, are definitely too low!

6. Frame Type

One user reported that the 'Frame Type', under IPX/SPX Properties—Advanced, should be set to Ethernet 802.3. This shouldn't really be needed—I've always had mine on Auto—but if all else fails it is certainly worth a try—but see item 10 below too!

7. Network drivers don't always work

One instance has been reported of a presumably buggy LAN adapter driver, a 3COM one for Windows XP, providing incorrect Server Node data to WideServer (see item 2 above). Using the ServerNode parameters logged gave no connection. Reverting to the slightly earlier 3COM drivers actually provided and installed by Windows XP fixed this.

Of the five numbers making it up, the first two identify the specific Network Address. If you've only got one then these numbers are normally both 0—but see also item 3 above. The other three constitute the physical address of the adapter card itself, and appears to be called either explicitly the 'physical address', or sometimes the 'MAC address'.

There are ways of verifying this address. These are different for the various Windows versions. On my Windows 98SE installations I can either use the Shareware program SiSoft Sandra Pro, looking at its "Network" information, or use the Windows-supplied DOS program IPConfig (run this in a DOS window with the parameter /ALL). On Windows XP (and presumably also earlier NT versions) you can get the information via the program IPXroute.exe.

Please also see item 10 below, which details other problems that can arise in getting the correct ServerNode values.

The address you get will actually consist of six values in hexadecimal notation. To take the example from section 2 above:

17248 ServerNode=0.0.49152.1264.9490

Each of the last three numbers can be expressed in hexadecimal, so:

C000 04F0 2512

When these are stored in 6 consecutive 8-bit 'bytes' the two halves of each 16-bit number get reversed, due to the way Intel processors store numbers:

00 C0 F0 04 12 25

This is how this particular physical address (or 'MAC' address) will be seen in those other applications and parts of Windows.

8. Network adapters don't seem to like sharing Interrupts

A lot of the problems I had with my Network in the early days turned out to be all due to the way either the BIOS or Windows had configured my system. The Network card was sharing an interrupt (IRQ) with another card.

The symptoms of the problems were odd. Most things worked fine, but there were strange things going on with WideFS-connected applications. I eventually found that the data transferred across the Network was suffering from occasional corruption—mostly the odd *extra* character being inserted. I proved this by transferring some very large files across, using drag-and-drop in Windows Explorer. Comparing them afterwards showed errors. Not many: about one character in a few million. But enough to make the Network untrustworthy and to occasionally do odd things to applications.

Eventually, after quite a bit of hair loss (and I didn't have much in the first place!) I found it was IRQ sharing. More hair was then lost trying to correct this, by things like shuffling the PCI cards around, trying to change allocations in the BIOS, and in Windows too. Sorry, I really cannot explain how to do this stuff here—it was all guesswork, trial and error. Mostly error.

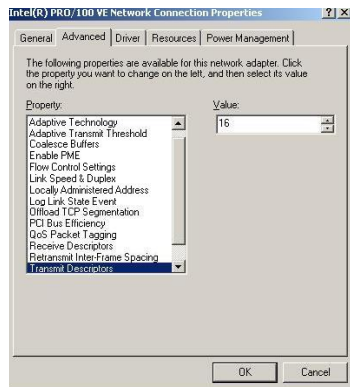
9. Network adapters can go wrong!

One other point to realise: LAN adapters are sometimes faulty. Out of seven known brands I've had three failures over as many years. It is sometimes worth changing them, or even swapping them between machines, to either eliminate or confirm this as a possible reason for problems. Symptoms may vary from no connection at all, to unreliable or very slow operation, to actual freezing during otherwise normal operation.

Other notes for trouble-shooting

I received a solution for one odd symptom from a user with a severe slow-down using TCP/IP. It looked like block coalescing was the cause, despite the fact that both WideServer and WideClient turn this off inside the Windows drivers. It turned out that the Intel LAN adapter, which was integrated into the PC motherboard (in this case a Gigabyte 8IG1000 Pro) had the coalescing algorithm coded into its firmware! The supplied driver configuration utility, when the user dug deep enough, presented options to turn it off. This is one more thing to check, should you have such problems. I would never have thought of it!

For this particular motherboard, this was the secret. Find the entry for Network Adapters, right click on the LAN adapter and select Properties then Advanced tab. You should get something like this:



As you can see, in the Pro/100 VE there are way more than the usual 3-5 settings available! These include Adaptive Technology, Adaptive Transmit Threshold, Coalesce Buffers, PCI Bus Efficiency, and Transmit Control Block parameters, all of which effect data store-and-forward and coalescing. In the case of this adapter, setting Adaptive Technology to 0 turns out to be the magic bullet that turns it all off!

If this is wrong the symptoms are a choppy flow of data, in one or possibly both directions, and significant lag and jitter in the data stream.

One user had severe problems with both Client and Server logging messages about “send() done after n retries ...”, disconnections and re-connections, “Send() request depth over 100!”, and even some missing and corrupted blocks (sumcheck errors). He solved this problem himself and reported this:

“I’m pretty sure that the QOS support and an ‘auto’-setting in the network card’s connection properties caused the described behaviour. QOS support was removed and the network card set to 100 MBit Full Duplex, et voila: everything runs fine!”

That’s it for now. If I get any other suggestions I’ll add them here.

WidevieW compatibility

First, please note that WidevieW is by Luciano Napolitano, and is NOT part of WideFS

If you have enough PCs on your network, you can still run two or more copies of Flight Simulator linked by WidevieW, giving you side views or maps. Only the main (flying) PC will run WideServer. You cannot run WideClient on a PC which is also running Flight Simulator—after all, its job is really to replace it and interface to it across the network connection instead.

Note that the port numbers used by WideServer and WideClient default to 8002 and 9002, so avoiding clashes with Wideview's default. For very large networks running more than one flying copy of Flight Simulator, WideServers with different port numbers can link to WideClients with matching port numbers, but there can only be one WideClient on any one PC, as this provides the interface expected by applications and there’s no way to differentiate between two or more.

FSUIPC Button Programming for Client Buttons

With a user-registered FSUIPC version 3.20 or later, and WideFS 6.22 (or later) in both Clients and Server, all Windows, EPIC and GoFlight buttons will be recognised on Client PCs and transmitted to FSUIPC for programming *unless* this action is explicitly turned off in the Client.

This action is automatic. You need do nothing to enable it—except, for GoFlight only, installing the Gfdev.dll as mentioned in the WideClient installation instructions earlier. For WideClient there is only one controlling parameter:

ButtonScanInterval=20

This parameter, in the [Config] section of the WideClient.ini file, controls the rate at which WideClient scans EPIC and Windows joystick buttons. GoFlight buttons are not scanned by WideClient—those are dealt with by the Gfdev.dll. However the parameter is still relevant since if you set this to 0 (zero), it switches the whole facility off.

The units here are in milliseconds. You can scan faster by reducing the value, but bear in mind that the frequency of button changes seen by the Server will be limited by the Network frame rate in any case, and this is pretty well tied to the FS frame rate by WideServer. The 20 millisecond default is generously allowing up to 50 changes per second in any case.

Buttons from different clients are differentiated by WideServer according to client names. When WideServer sees a new Client connecting, it makes a record of its name in the WideServer.ini file, assigning it a number. For example:

```
[ClientNames]
1=LEFT
2=CENTRE
3=RIGHT
4=NOTEBOOK
```

Each time it reconnects it can then check and assign it the same number. This provides consistency in button numbering—the “joystick” number displayed by FSUIPC in its Buttons page will have 1000 added for Client 1, 2000 added for Client 2 and so on.

There is a limit of 32 client names. You aren’t likely to reach that (I hope!), but if you change PCs or rename them, the list could build up with redundant names of clients no longer connected. WideServer has no way of knowing that the Client which used to be called “OLDPC” is effectively the same one as that called “NEWPC”. So when you change or rename PCs, please check the list here and edit the names so that they still reflect where things are correctly. You need to do that when FS is not running.

WideServer INI file options

Here all the user features of the server's control parameter file are described. Normally you won't need or want to mess around with many of these parameters. There are all placed into the 'WideServer.ini' file, which goes into the Flight Simulator modules folder along with the WideServer.dll module itself. The filetype 'ini' merely refers to the file as an Initialisation file—reflecting the fact that it is *only* read by the program on first loading. Changes should only ever be made when the program is not running.

Note that WideServer does *not* generate an INI file with all these parameters included. If they are omitted, as most of them will be most of the time, then they assume their default meaning, as documented here.

The [Config] section

This section of the INI file is where the physical configuration and other rather technical parameters are placed. With rare exceptions you should not need to change any of these. The main exception is the possibility of *adding* a “**ProtocolPreferred**” line if you wish—please see the discussions above on protocols.

ProtocolPreferred: This is omitted by default, leaving the choice up to clients, with TCP being defaulted throughout. If you are using Windows XP or 2000 throughout you can force uncommitted clients to choose a specific protocol by setting this Server parameter to one of **TCP**, **UDP** or **SPX**.

RARELY CHANGED PARAMETERS

AdvertiseService=Yes: By default, on Windows XP, WideServer “advertises” itself by sending out the Server details using a Mailshot broadcast. WideClients running on a Windows XP client PC will be able to receive these and so connect automatically. If there are several Servers, any clients without explicit Server details in their configuration files (the INI) will simply connect to the first one they see.

The mailshots are sent every second for the whole time WideServer is running. The extra loading on the Network is unlikely to be noticed, but if you want to reduce it or eliminate it altogether, just change this parameter. “No” or “0” will stop it, whilst a value from 1–10 will make the interval that number of seconds.

Port=8002: This controls the port number to be addressed. Only clients using the same Port number are served by this server. You can have several WideServers running on a network, each using a different Port number. However, you can of course only have one copy of WideClient in each machine, as its job is to ‘pretend’ to be an FS6IPC.DLL-equipped copy of FS98 (or FS95 or FS2000) so that FS6IPC-aware applications will use it.

The default of 8002 is chosen only to avoid the two ports (8000 and 8001) used by WidevieW.

Port2=9002: This controls the Port number used for the file transfer option.

AutoUpdateTime=13: This sets the *minimum* time between AutoUpdates, in milliseconds. The default value of 13 will stop the individual data frame rate for any client exceeding 75. Lower values than 13 can be set, but there is a limit based on the way the DLL works, and also the capacity of the LAN. 100Mbit LANs may cope better with lower times. But watch out for FS frame rates: there needs to be some time left over for the simulation—and of course the application at the Client end needs some time too!

RestartTime=10: This feature allows you to make WideServer automatically close down its network serving action and restart it (just as if it had been freshly loaded) when there have been no connections for at least the specified number of seconds.

If you have problems on your network starting up the WideFS clients then this may help unclog Windows' sockets software (?). You may also find it useful to use the ‘Restart Hot Key’ facility, described below.

To disable this feature set the RestartTime to 0.

AutoRestart=0: This operates a facility to automatically restart the Server (forcing all clients to reconnect) if a low frame rate (below 5) is experienced for the specified number of seconds (with a minimum of 5). The frame rate here is the total frames per second managed for all connected clients added together, so an average of less than 5 fps for over 5 seconds indicates a problem. However, this attempt at fixing it is a bit of a sledgehammer and in general should be avoided. The default setting of 0 switches it off.

NoStoppedRestarts=Yes: Only set this to ‘No’ (to allow restarting of all clients after extended stoppages on the FS PC) if you experience any crashing problem otherwise, or problems whereby client applications do not properly resume when you've been visiting FS menus for extended periods.

SendTimeout=15: This is the maximum time (in seconds) for which the Server will continue to try re-sending data to a client when each attempt is blocked for any reason. After this time has elapsed with no success in sending it data, the client is disconnected. You can disable this action by setting the timeout to zero.

MaximumBlock=4096: This parameter controls the block sizes used by WideServer when sending altered data out to client applications. It shouldn't be set less than 512, nor larger than 16384. The default of 4096 seems to suit most applications and the higher speed (100 mbps) Networks, and the effect of the limitation is usually all over after the first few seconds of an application starting. After that only changed values are re-sent so the limit is seldom reached, except possibly with TCAS applications and high numbers of moving AI aircraft in FS2002. If you are using a slow Network connection (10 mbps, USB, Serial or Parallel) then you may find 2048 better.

The [User] Section

RestartHotKey=: This option allows you to define a hot key which you can use in Flight Simulator to force WideServer automatically close down its network serving action and restart it (just as if it had been freshly loaded). If you have problems on your LAN with WideFS clients apparently stalling then this may help unclog Windows' sockets software (?).

The format for specifying the hot Key is 'keycode,shift states', for example:

RestartHotKey=78,11

specifies Shift+Ctrl+N (N for "Network"). The first value determines the main key required. This is a Windows 'virtual keycode'. A list of these is given in each of my FS "Controls" packages (there are separate ones for FS98, FS2000 and FS2002, but these codes are the same for each). These packages are available separately.

The second value determines additional shift states needed, as follows

| | |
|--------------|-------------------------|
| omitted or 8 | key on its own |
| 9 | Shift + |
| 10 | Control + |
| 11 | Shift + Control + |
| 12 | Alt + |
| 13 | Shift + Alt + |
| 14 | Control + Alt + |
| 15 | Shift + Control + Alt + |

Note, however, that not all combinations will work with all keycodes. The values can also be 0–7 instead of 8–15. The addition of '8' here is merely to provide compatibility with the way key presses are specified in Flight Simulator's CFG files. And especially note that the use of the 'Alt' key in many combinations is problematic and will tend to invoke FS's menus. Avoid this key if at all possible.

AllowShutdown=No: Set this to Yes *only* if you want to allow client programs to shut down your Flight Simulator computer by writing a special value to the IPC interface. (Details for programmers are in the FSUIPC SDK -- look for offset 3320).

AllowShutdown=App can be used instead if you just want WideServer to close FS down, leaving the PC itself up and running. Of course any applications featured in WideServer's "Close" parameters (below), will also close.

AutoShutdown=No: Set this to 'Yes' if you want WideServer to send shutdown messages to clients when Flight Sim is closed normally. Set it to 'Apps' if you want WideServer to tell clients to close down only any "CloseReady" applications when FS is closed normally. (The RunReady and CloseReady parameters for WideClient are described later).

ShutdownHotKey=: This option allows you to define a hot key that you can use in Flight Simulator to get WideServer to actually write the special "shut down" value to the IPC interface. This will be obeyed by all those WideClients with appropriate **AllowShutdown** parameter settings, and then also by WideServer itself if so configured. Using this you can close FS down and have the Client PCs also close at the same time (or a bit earlier, in fact, to ensure they see the WideFS message).

The format for specifying the hot Key is the same as above. I use Shift+Ctrl+E, so the setting is:

ShutdownHotKey=69,11

CloseAppsHotKey=: This is to define a hot key that you can use in Flight Simulator to get WideServer to send a different "shutdown" pattern via the IPC interface. This pattern asks for applications (only) to be closed—see the "AppOnly" option for **AllowShutdown** in the section on Wideclient, later. This will be obeyed by all WideClients with *any* **AllowShutdown** parameter setting other than 'No', and then also by WideServer itself if so configured—on the Server FS itself will be closed. Using this with client applications loaded and closed by the **RunReady** and **CloseReady** facilities means that the applications will close when you use the hot key to close FS, then start again when you start FS again. The WideClient program is left running on the client PCs so that this can take effect.

The format for specifying the hot Key is the same as above. I use Shift+Ctrl+C, so the setting is:

CloseAppsHotKey=67,11

Log=Errors+: This is a debugging aid which I may ask you to change if I need more information to help sort out a problem with application interfacing. The default will create a WideServer.log file in the FS modules folder which contains details of basic errors but otherwise just some starting and stopping information and (most important for me!) the version number.

Set to 'Yes', this parameter logs all simulator variable read and write calls (and the results) made through WideServer. This is really only of any use to application developers, so they can see how their program is operating once its requests have been through WideClient.

Other settings are 'No' (not recommended as you lose error data), 'Errors' (only showing error reports, no other useful stuff) and 'Debug' (which gives details of WideServer's dealing with Windows' Sockets too).

If you do switch on any type full logging (e.g. 'Yes' or 'Debug'), be sure to keep the session short. The log file gets *very* large! The 'Debug' setting is only used on request for possible help in resolving complex problems.

Monitor=: This can be used by developers to find out what is happening to any one particular area of the FSUIPC variables memory area—i.e. the "offsets" being written or read by WideFS applications. It is better than using "Log=Yes" just to sort out a known variable. The format is:

Monitor=<offset>,<size>

where the offset is in hexadecimal and the size is a number of bytes in decimal. If you only want to watch one byte, you can omit the "<size>" part. For example

Monitor=028C,1

makes WideServer log all network reads from and writes to this location, which happens to be the Landing Lights switch.

You can Monitor up to 8 different offset areas. Just list more <offset>,<size> parameters on the same line, thus:

Monitor=04E0,88,48F0,10,5400,512,5600,256,5B00,128

which means 88 bytes at 0x04E0, 10 bytes at 0x48F0, and so on.

For a full picture you should use the same parameter at the client end too—i.e. in the WideClient.ini file(s).

IgnoreSumcheck=No: Setting this to 'Yes' is a *last* resort in trying to get some server-client interaction on a system where the initially larger client request blocks are consistently failing with sumcheck errors. The proper solution is to fix the protocol on the LAN. (This facility dates back to Windows 95 days. There is some evidence suggesting that the IPX/SPX protocol is not sufficiently reliable on original Windows 95 releases to provide good blocks. Users are really advised to try to upgrade to Windows 98SE, or at least Windows 95 OSR2.1).

ShowCounts=No: Set to 'Yes' to show the total network read and write counts (in bytes) in the Flight Simulator title bar.

TitleBarUpdate=Yes: Set this to 'No' if you don't want any sign that WideServer is running. Sometimes other programs and add-ons can be affected by the changes that WideServer makes to the Flight Simulator title bar. With this set to 'No' these add-ons may run correctly.

Action1=, **Action2**=, etc: These tell the program to execute the given command line on receiving the corresponding 'Action' message from a WideClient. See the WideClient ini file notes for how to set this up.

These commands can have an extra parameter to make the program being run do so at HIGH priority (higher than Flight Simulator), and optionally Hidden (i.e. running invisibly). The latter does not work with all programs, however. **ONLY** use these facilities with programs that are run and completed quickly—loading control programs into an EPIC card is a good example. The extra parameter is "HIGH" or "HIDE", respectively, as in:

Action1=HIDE,"c:\epic\loadepic fs98prop"

Note that HIDE implies HIGH as well. It works with Loadepic. Running it at high priority improves the chances of a successful load somewhat—when Flight Simulator is running there can be more clashes at the Epic direct interface.

Run, RunIf, Close and CloseIf parameters

These are still incorporated into WideServer for backward compatibility with users' existing set-ups, but these days it is much better to use the Run options provided in FSUIPC. If anyone needs information about them please ask on the Support Forum.

KeySend parameters

These are still incorporated into WideServer for backward compatibility with users' existing set-ups, but new users please use the FSUIPC Buttons page instead and assign buttons to the KeySend control in the drop down controls lists. The KeySend number (1–255) is entered as a parameter. You can also assign keys to KeySends in FSUIPC.

WideClient INI file options

Here all the user features of the client's control parameter file are described. Normally you won't need or want to mess around with many of these parameters. There are all placed into the 'WideClient.ini' file, which goes into the same folder as the Wideclient.exe program copy it is to be used with. The filetype 'ini' merely refers to the file as an Initialisation file—reflecting the fact that it is *only* read by the program on first loading. Changes should only ever be made when the program is not running.

The [Config] section

This section of the INI file is where the physical configuration and some rather technical parameters are placed. Very few of these need ever be changed by the user:

Protocol: Set this to one of **TCP**, **UDP**, or **SPX** if you want to specifically make this client use that protocol. TCP is defaulted in any case unless you are using Windows XP or 2000 on both Server and Client PCs and want to control the protocol from the Server (see **ProtocolPreferred** earlier).

ServerName=: Either this or the **ServerIPAddr** can be provided if the TCP/IP protocol is chosen and set by the **UseTCPIP** parameter. The server name is the one you assigned in the Network properties: on Windows 98 and probably others it's the "computer name" in the Identification tab. If you are using Windows XP on both the Server and Client PCs this shouldn't be necessary unless you have two or more Servers and need to keep the connection fixed to a specific one..

ServerIPAddr=: If you are using TCP/IP and you need to give the Server details, you can elect to specify the Server by its IP Address instead of its name. If you give both, only the IP address will be used. To use this you need to have assigned a fixed IP Address for your Server PC, as described earlier (in the "Configure Your Network" section). The format is four decimal numbers separated by points, for example:

ServerIPAddr=192.168.0.3

I recommend using only ServerName as this avoids any difficulties with IP address changes.

ServerNode=: This is only used with the SPX protocol, and can be omitted for Windows 95, 98 and (probably) ME installations, but it is often needed when Windows NT, Windows 2000 or Windows XP are being used, whether at the Server or Client end, or both. It may make initial connection more efficient on Windows 95/98/ME too.

If you are using Windows XP on both the Server and Client PCs it shouldn't be necessary to provide it as WideClient should receive this information automatically in one of WideServer's broadcasts.

If you do need to set it, the ServerNode is determined by the specific network adapter being used by the Server. You can get this most easily by running Flight Simulator with WideServer.dll installed, and then looking at the WideServer.log file (which you will find in the Modules folder, with the module itself). Near the beginning there should be a line containing

ServerNode=n.n.n.n.n

where the n's are decimal. Copy this part of the line into the [Config] sections in all the Wideclient.ini files you are using for programs which will connect to this specific server.

Be aware that there have been problems with some network drivers reporting the incorrect 'ServerNode' values to WideServer. See the trouble shooting section earlier.

RARELY CHANGED PARAMETERS

Port=8002: This controls the port number to be addressed. Only the WideServer running with the same Port number will be addressed by this WideClient. You can have several WideServers running on a network, each using a different Port number.

Note that you can only have one copy of WideClient in each machine, as its job is to "pretend" to be an FS6IPC.DLL-equipped copy of FS98 so that FS6IPC-aware applications will use it.

The default of 8002 is chosen only to avoid the two ports (8000 and 8001) used by WideviewW, in my earlier package "WideEFIS".

Port2=9002: This port is used for file transfers only.

Window=43,44,886,589: Don't alter this unless you "lose" the Window! It stores the size and position you last used for the main window.

Visible=Yes: Normally leave this to default. Other values are:

| | |
|-----|--|
| No | if you don't want any sign of WideClient (in this case, to terminate use CTRL-ALT-DEL) |
| Min | to start up minimized |

Max to start up maximized

Note that there are not many client programs that actually need a Window. EFIS by Chris Brett is one. You should leave this parameter to default to 'Yes' for EFIS, then size the WideClient window to fit the EFIS plus FMC windows, arranged nicely side-by-side, with the control buttons beneath. WideClient will remember the window size and position. (Similar considerations apply to Moving Map. You can arrange all EFIS and Moving Map windows within WideClient's window, and they will all be remembered by their respective programs).

WaitForNewData=500: Network response with new data timeout, in milliseconds. This applies only to requests made for data which is not yet "registered" with the server, as being needed by this client. The client deliberately waits, examining messages coming back from the server, for up to this number of milliseconds, or until it sees the new data it requested arriving.

A large timeout here helps guarantee that the application will see good data right from the outset (assuming FS is already running and the Server is able to respond in this time), rather than be supplied with zeroes from WideClient's own memory. Once the client knows that the data requested is registered with the server it no longer waits but services the application from its memory and relies on updates for changes from the server.

This means that, with a larger timeout, the application may be slightly jerkier initially, but obtains good data. If the slower start is not acceptable, you can omit the timeout altogether by setting this parameter to 0.

Note that if the connection to the Server is restarted at any time, for any reason, then all the data it was previously receiving is again considered un-registered, so the new data timeout applies again.

ApplicationDelay=0: A delay inserted into each call to WideClient made by each application program. This value, in milliseconds, applies to every request made by all your client applications. Since the Server is sending updates for requested data in any case, this timeout is only useful for limiting the processor time used by the application, to stop it hogging the client PC when there are other programs to run.

Many FS6IPC/FSUIPC client applications are reasonably well behaved, however, and do timeshare well enough, so this timeout can be defaulted to 0. However, with some applications, if there is not a delay before returning to the application from each of its data requests, then the loop can be too tight and there may be some real difficulties in accessing other facilities on that PC, moving and sizing windows, and so on. This should never be a problem on Windows 2000 or XP, but may be so on Windows 98/ Me which are not so good at timeslicing.

NetworkTiming=5,1: This gives control over two quite critical periods. Both numbers are times in milliseconds. The first (defaulting to 5) specifies the minimum time to be given to the Application to allow it to read changes after WideClient has received each new frame. The second (defaulting to 1) specifies the minimum time to be given to the Network thread to allow new frames to be received and processed before acting upon each read or write request from the Application.

This is quite a balancing act and needs to be just right to achieve perfectly smooth operation. Ideally the Network thread should receive one frame which the Application can immediately process, and so on, but in practice frames are like buses, they run in bunches and sometimes not at all. These numbers allow experimentation with the way this is turned into apparent smoothness.

MaxSendQ=100: This sets a limit on the number of frames awaiting transmission to the Server. If the queue exceeds this number, the action specified by **WhenMaxSendQ** is taken.

WhenMaxSendQ=Recon: This specifies what to do if the send queue exceeds the maximum allowed by **MaxSendQ**. This can be either "Recon" to discard all the frames *and* reconnect to the Server, effectively starting afresh, or "Flush" which simply discards the pending frames. Note that if you have a Client reaching the default maximum of 100 you do have something badly wrong. For any session the maximum seen is logged at the end, when the WideClient program is terminated. Good values are 2–10. You won't see less than 2 normally, but a perfect setup will show a maximum of 2 on all clients.

SendScanTime=10: This sets the *minimum* time between frames being sent from this Client to the Server, in milliseconds. The default value of 10 will stop the individual data frame upload rate exceeding 100. If you get too much stuttering, or experience delays or reconnections on a client, it may be that one of the applications is writing values to FS too fast. You can increase the **ApplicationDelay** (above) but that will slow down reads as well as writes, or you can increase this value instead.

Priority=3,1,2: There are three threads in WideClient: the main one which is receiving requests for reads and writes from applications, a Network reception thread, and a Network transmission thread. Normally, the reception thread is run at a high priority with the transmission thread a little lower, but still higher than the applications. You can change this order here if you like. The three values are all 1, 2 or 3, and give relative priorities of Applications, Receives, Transmits, in that order—with 1 highest, 3 lowest. If you set them all the same there will be no priority differences.

PollInterval=2000: The number of milliseconds allowed between each message sent to the Server to confirm that this client is still active. When there is no other reason to send a message this will merely cycle through the FSUIPC offsets being read as a kind of 'refreshment'. WideClient otherwise only sends messages to the Server for Writes to FSUIPC offsets and for Reads of new offsets (new since the last connection or re-connection).

ResponseTime=18: The number of seconds elapsing with no message arriving from the Server before WideClient decides that the connection is lost and attempts to re-connect. Note that twice this time is actually allowed until such a timeout has occurred twice. After this the specified time is used. This initial leniency allows for longer delays at the Server during initialisation, both of FS and all of its attendant applications.

ButtonScanInterval=20: This parameter controls the rate at which WideClient scans EPIC and Windows joystick buttons. GoFlight buttons are not scanned by WideClient—those are dealt with by the Gfdev.dll. However the parameter is still relevant since if you set this to 0 (zero), it switches the recognition of buttons off altogether. The units are milliseconds.

ClassInstance=0: You can't normally run WideClient and FS, or two copies of WideClient, in the same PC, as they have the exact same Window Class. FSUIPC applications cannot differentiate between them. In fact both FS and WideClient prevent more than one such instance starting in the first place. However, there are two very unusual situations:

- (a) You want a WideClient running and talking to a server on another PC, whilst on the same PC as the Client you have a copy of FS running (possibly linked to the other by WideviewW), or
- (b) You want two WideClients talking to different Servers (by different Server Names or Ports).

For these situations, Wideclient allows you to change its Window Class name. For this to be of any use, the applications you are running will also have to be set to connect to the different Class name—this will not be possible with most standard FSUIPC connecting applications. In this regard, this facility is rather restricted to those with programming abilities.

To change the Class name used, simply set the parameter "ClassInstance=n" where n is a number in the range 0–99. The Class Name then becomes "FS98MAINnn"—i.e. the number is appended as two digits, 00–99. If you do this, be aware that most ready-made applications for FSUIPC will not connect.

The [User] Section

Background=: WideClient will display a user-supplied bitmap in its window instead of the featureless grey. This allows suitable backgrounds for utilities such as EFIS, EFIS98 and Moving Map to be designed. Specify the BMP file to be used by adding this line into the [User] section of WideClient.ini:

Background=filename.bmp

The filename can contain the complete path to the file: useful if it isn't stored in the load path.

Run1=, Run2=, ... Run9=: These tell WideClient to run the specified programs (specified by their full pathnames), as WideClient is initialising. For example:

Run1=f:\efis\Efisv2.exe

This would load EFIS Version 2. Other suitable clients are RWX5.EXE (Real Weather 5 by Jeff Wheeler and Steve Halpern), and Aeroview.Exe (the moving map free on the CDROM with Nick Dargahi's book). These are the ones I use and have tested, but any FS6IPC.DLL using program should run fine. If the program needs command-line parameters, these can be included by enclosing the whole value in double quotation marks ("") so that the spaces needed don't cause problems.

Delay1=, Delay2=, ... Delay9=: These optionally define delays, in seconds, to be executed after the corresponding **Run** parameter, above. Whilst the delay is operating WideClient is sleeping, so it will *not* attempt to make a connection during this time. The maximum delay which is set is 60 seconds.

Close1=Yes, Close2=Yes, ... Close9=Yes: Add these to ask WideClient to close the programs it loaded by **Run** when WideClient itself closes or when requested by an appropriate **KeySend**. This is performed by sending the program's Windows a WM_CLOSE message, so if it ignores these, or has no Windows defined, this won't work.

RunReady1=, RunReady2=, ... RunReady9=: These are identical to the **RUN** options above, except they are not actioned until WideClient is actually connected to WideServer.

DelayReady1=, DelayReady2=, ... DelayReady9=: These optionally define delays, in seconds, to be executed after the corresponding **RunReady** parameter, above. Whilst the delay is operating WideClient is actually still running. The maximum delay which is set is 60 seconds.

CloseReady1=Yes, CloseReady2=Yes, ... CloseReady9=Yes: Add these to ask WideClient to close the programs it loaded by **RunReady** when WideClient itself closes or when requested by an appropriate **KeySend**. This is performed by sending the program's Windows a WM_CLOSE message, so if it ignores these, or has no Windows defined, this won't work.

RunKey1=, RunKey2=, ... RunKey9=: These are identical to the **RUN** options above, except they are not actioned automatically at all, but only when a **KeySend** request to run them is received. This allows the programs to be loaded under explicit control from the server. See **KeySend**, below.

CloseKey1=Yes, CloseKey2=Yes, ... CloseKey9=Yes: Add these to ask WideClient to close the programs it loaded by **RunKey** when WideClient itself closes or when requested by an appropriate **KeySend**. This is performed by sending the program's Windows a WM_CLOSE message, so if it ignores these, or has no Windows defined, this won't work.

Close=: This parameter can list up to three Windows CLASS names, representing programs that should be closed when WideClient itself closes. Separate the Class names with commas.

For example, EFIS98's CLASS name is TToolWindow97, so you can set WideClient to close EFIS98 automatically when you close WideClient by the parameter setting:

Close=TTToolWindow97

To obtain Class names is not easy without programmer's tools, but the program's author can tell you. Class names for some of the possible Client applications are provided below, in the section on the **KeySend** feature. Project Magenta class names can be configured by parameters in the individual module ini files.

The **Close** parameter provides no facility to distinguish between two or more programs which use the same Windows Class name.

Actions=: This tells the program to create a Menu with the listed commands. Each command, when selected, sends a special message to WideServer. The first command executes the **Action1** line in WideServer's ini file, and so on. Use commas to separate commands. For example:

Actions=Jet,Prop,Heli

These three menu entries might be used in conjunction with these WideServer.ini parameters:

Action1="C:\EPIC\LOADEPIC FS98JET"
Action2="C:\EPIC\LOADEPIC FS98PROP"
Action3="C:\EPIC\LOADEPIC FS98HELI"

thus getting the server to re-load different Epic control programs on request from the client.

Log=Errors+: This is a debugging aid which I may ask you to change if I need more information to help sort out a problem with application interfacing. The default will create a WideClient.log file in the FS modules folder which contains details of basic errors but otherwise just some starting and stopping information and (most important for me!) the version number.

Set to 'Yes', this parameter provides a log of all simulator variable read and write calls (and the results) made through WideClient. This is really only of any use to application developers, so they can see how their program is operating once its requests have been through WideClient.

Other settings are:

No not recommended as you lose error data

Errors only showing error reports, no other useful stuff

and a range of really heavy logging options (Debug, Debug+, DebugAll, All, AllRx, PartRx) which will tend to only ever be used under instruction when trying to nail really obstinate problems.

If you do switch on any type full logging (e.g. 'Yes' or any of the "heavy" settings), be sure to keep the session short. The log file gets *very* large!

You can also have the full data logging switched by Hot Key: e.g.

Log=K1190

The number after the K here represents Shift >. The value is the Virtual Key number (see one of my FS Controls packages for a list) plus 1000 for Shift and/or 2000 for Ctrl. The logging state is then shown in the WideClient title bar.

Monitor=: Developers can use this to find out what is happening to any one particular area of the FSUIPC memory area—i.e. the offsets being written or read by the WideFS applications on this client. The format is:

Monitor=<offset>,<size>

where the offset is in hexadecimal and the size is a number of bytes in decimal. If you only want to watch one byte, you can omit the ",<size>" part.

For example

Monitor=028C,1

This makes WideClient log all Network reads from and writes to this location, the Landing Lights switch.

You can Monitor up to 8 different offset areas. Just list more <offset>,<size> parameters on the same line, thus:

Monitor=04E0,88,48F0,10,5400,512,5600,256,5B00,128

which means 88 bytes at 0x04E0, 10 bytes at 0x48F0, and so on.

For a full picture you should use the same parameter at the server end too—i.e. in the WideServer.ini file.

AllowShutdown=No: Set this to Yes only if you want to allow client programs (or the server's **ShutdownHotKey**) to shut down this client PC by writing a special value to the IPC interface. (Details for programmers are in the FSUIPC SDK—look for offset 3320).

AllowShutdown=App can be used instead if you just want WideClient itself to close down, leaving the PC itself up and running. Of course any applications featured in WideClient's "Close" parameters (above), will also close.

AllowShutdown=AppOnly is a further variation which leaves WideClient running (awaiting a re-connection from a reloaded FS), but closes down any applications which were loaded with "RunReady" or "RunKey" parameters and have a "CloseReady" or "CloseKey" entry too. When FS starts up again on the server PC, WideClient will reload the "RunReady" programs (but the "RunKey" programs will await the appropriate KeySend request).

NOTE that there is a related "CloseApps" facility that will perform the AppOnly function if the **AllowShutDown** parameter is set to anything other than 'No'. This facility can be instigated by a WideServer hot key, or by any application writing a different value to the IPC interface. Again, the FSUIPC SDK will provide programmers the details.

ShowRxFrameRate=No: Set this to Yes to show the frame rate—i.e. the number of frames per second received from the Server on this client. In general this will be very low (0 is less than 1, not absolute zero! <G>) when FS is not doing much, and, hopefully, something near to FS's frame rate when things are changing a lot. But this will depend upon the Applications. If there's only one Application and it is only reading the Time of Day (to the nearest second) then this will only change at most once a second, so the frame rate will hover around 0 or 1 no matter what is going on in FS. Performance summaries are shown at the end of the Log.

ShowCounts=No: Set to Yes to show the total network Read and Write counts (in bytes) in the WideClient title bar. In general it is more useful to use the reception frame count instead.

EFISkbfocus=No: *[Only suitable for use with the EFIS98 package on the Client]*

Set this to Yes if you are using Chris Brett's EFIS98 package on this Client, and wish to make the keyboard input mode in EFIS98 operate automatically, whenever the Client window has focus. Make sure EFIS98 is docked: do this after sizing the windows to suit. You can position them afterwards.

This option saves having to remember to press Shift+Space before selecting options by keyboard short cuts, or entering details into the FMGC or the Options dialogue. WideClient will do this for you, and will disable keyboard input whenever the Client window loses focus (e.g. to access some other program).

KeySend: Clients can receive requests from the Server. The requests are simply encoded by a reference number, in the range 1 to 255 inclusive. In the Server PC you use FSUIPC to program these requests. They can be assigned in the **Buttons** and/or **Keys** option pages in FSUIPC. Just find the added FS control called "KeySend" in the drop-down lists. The KeySend reference (1–255) is entered as a parameter.

This mechanism allows use of recognised buttons (including EPIC, GoFlight, and PFC.DLL connections) and server key presses to be relayed to Client PCs as KeySend encoded messages. It can also drive specific facilities in WideClient to run and close programs, or to send PTT on/off requests to Roger Wilco or AVC.

Basically, the idea is simple. To make something happen on a Client, a keypress is usually needed on that client. WideClient can deliver that keypress using this facility. But the keypress needed at the client cannot be programmed as such on the Server, because it would be delivered to FS on that PC instead.

To counter this, WideFS implements a special data value, the KeySend reference number (1–255). The number used is not relevant to anything, it is just an identifier, a reference. When a button or key on the Server is programmed for a KeySend number 'n', that number 'n' is broadcast to all Client PCs. The Client program watches for these. If it sees a KeySend reference number it has an entry for, it then acts upon it, if not it ignores it. Thus, different Clients can do different things with the same KeySend message, or the same things with different messages. The flexibility of doing things this way is enormous.

Currently KeySends can, with the specific exceptions dealt with below, only be used to generate Key strokes on the Client PCs. There are no facilities for mouse movement nor mouse-clicking. Please check Luciano Napolitano's site (www.wideview.it) for programs which handle mouse operations.

In the simplest form you specify the keyboard actions required for given KeySend reference numbers (1–255 as required) as shown in the following examples:

```
KeySend1=65,9
; Shift+A
KeySend2=8,11
; Shift+Ctrl+Backspace
```

```
KeySend255=112,12
; Alt+F1
```

Here the first value determines the main key required. This is a Windows “virtual keycode”. A list of these is given in the table below. The second value determines additional shift states needed, as follows

| | |
|----|-------------------------|
| 8 | key on its own |
| 9 | Shift + |
| 10 | Control + |
| 11 | Shift + Control + |
| 12 | Alt + |
| 13 | Shift + Alt + |
| 14 | Control + Alt + |
| 15 | Shift + Control + Alt + |

Note that not all combinations will work with all keycodes. The values can also be 0–7 instead of 8–15. The addition of 8 here is merely to provide compatibility with the way key presses are specified in Flight Simulator’s CFG files.

Use of the Alt key in many combinations is problematic. Try to avoid this key if possible.

Note that all these values operate the keystroke momentarily: i.e. they do a “key down” followed by a “key up”. If you want one KeySend event to press a key and a separate KeySend event to release it, then you will need to alter the shift state above as follows:

```
Shift state + 8 to Press the key
Shift state + 16 to Release the key
```

For example

```
KeySend1=65,17 ; Shift+A press
KeySend2=65,25 ; Shift+A release
```

These parameters make KeySend 1 and 2 press the Shift+A combination for as long as the action in the server need it to. You would probably program the KeySend entries in WideServer.ini to operate KeySend1 when a button is pressed and KeySend2 when it is released.

Take care when using these more advanced features not to get your client PC in a bit of a mess, with assorted key states stuck on. To release stuck keys, press them on the client’s keyboard—the Key UP codes should sort things out.

Okay. Here are the keycodes (though they may not all be usable):

| | | | | | |
|----|---------------------------------|-----|--------------------------------|-----|--------------------------------|
| 8 | Backspace | 68 | D | 102 | NumPad 6 (<i>NumLock ON</i>) |
| 9 | Tab | 69 | E | 103 | NumPad 7 (<i>NumLock ON</i>) |
| 12 | NumPad 5 (<i>NumLock OFF</i>) | 70 | F | 104 | NumPad 8 (<i>NumLock ON</i>) |
| 13 | Enter | 71 | G | 105 | NumPad 9 (<i>NumLock ON</i>) |
| 19 | Pause | 72 | H | 106 | NumPad * |
| 32 | Space bar | 73 | I | 107 | NumPad + |
| 33 | Page Up | 74 | J | 109 | NumPad - |
| 34 | Page Down | 75 | K | 110 | NumPad . |
| 35 | End | 76 | L | 111 | NumPad / |
| 36 | Home | 77 | M | 112 | F1 |
| 37 | Left arrow | 78 | N | 113 | F2 |
| 38 | Up arrow | 79 | O | 114 | F3 |
| 39 | Right arrow | 80 | P | 115 | F4 |
| 40 | Down arrow | 81 | Q | 116 | F5 |
| 45 | Insert | 82 | R | 117 | F6 |
| 46 | Delete | 83 | S | 118 | F7 |
| 48 | 0 on main keyboard | 84 | T | 119 | F8 |
| 49 | 1 on main keyboard | 85 | U | 120 | F9 |
| 50 | 2 on main keyboard | 86 | V | 121 | F10 |
| 51 | 3 on main keyboard | 87 | W | 122 | F11 |
| 52 | 4 on main keyboard | 88 | X | 123 | F12 |
| 53 | 5 on main keyboard | 89 | Y | 124 | F13 |
| 54 | 6 on main keyboard | 90 | Z | 125 | F14 |
| 55 | 7 on main keyboard | 96 | NumPad 0 (<i>NumLock ON</i>) | 126 | F15 |
| 56 | 8 on main keyboard | 97 | NumPad 1 (<i>NumLock ON</i>) | 127 | F16 |
| 57 | 9 on main keyboard | 98 | NumPad 2 (<i>NumLock ON</i>) | 128 | F17 |
| 65 | A | 99 | NumPad 3 (<i>NumLock ON</i>) | 129 | F18 |
| 66 | B | 100 | NumPad 4 (<i>NumLock ON</i>) | 130 | F19 |
| 67 | C | 101 | NumPad 5 (<i>NumLock ON</i>) | 131 | F20 |

| | | | | | |
|-----|----------------------|-----|------------------|-----|------------|
| 132 | F21 | 162 | Left Control ** | 190 | . > Key* |
| 133 | F22 | 163 | Right control ** | 191 | / ? Key* |
| 134 | F23 | 164 | Left 'Menu' ** | 192 | # ~ Key* |
| 135 | NumPad Enter or F24? | 165 | Right 'Menu' | 219 | [{ Key* |
| 144 | NumLock | 186 | ; : Key* | 220 | \ Key* |
| 145 | ScrollLock | 187 | = + Key* | 221 |] } Key* |
| 160 | Left Shift ** | 188 | , < Key* | 222 | ' @ Key* |
| 161 | Right shift ** | 189 | - _ Key* | 223 | ` ~ ! 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. |
| 222 | ' @ | is to the right of 186 |
| 192 | # ~ | is to the right of 222 (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 |

** These keys may or may not work, depending on the method used and the target program.

Directing Key Strokes more precisely

If the application that is to receive the keystroke is not a child window of Flight Simulator (i.e. of WideClient, which is substituting for FS in this case), the Windows keyboard focus may not allow the assigned keystroke to reach it. In this case you need to add another parameter, or maybe two, to each KeySend line, identifying the program to receive it.

If the program is one you are having WideClient load, using the **Run** or **RunReady** parameters, then the additional parameter can simply be the name of the parameter you used. For example:

```
KeySend1=65,9,Run1
```

```
and KeySend1=66,9,RunReady2
```

This is bar far the easiest and, usually, the most reliable way. However, if it is a program being run separately then you will need more information about that program, in particular the program's main Window class name, and where there's a chance of confusion, the title for the application window concerned. The next section goes into this in some detail.

CLASS Names

Windows class names either have to be supplied by the author of the program, or obtained by other programs such as the Spy programs that come with development packages like Microsoft's Visual C++. Here are the Class names for some (now rather old) FS utility applications. When there are more than two programs running with the same Class name, the title (from the title bar) is needed as well, as an extra parameter.

| | |
|------------------|---|
| FlightDirector98 | ThunderRT5Form |
| Project Magenta: | ThunderRT5Form,"PFD GLASS COCKPIT" |
| | (but PM modules now have programmable class names—see PM INI files) |
| Real Weather 5 | ThunderRT5Form |
| Aeroview | TestClass |

For example:

```
KeySend1=65,9,ThunderRT5Form,"PFD GLASS COCKPIT"
; Shift+A
KeySend2=8,11,ThunderRT5Form,"PFD GLASS COCKPIT"
; Shift+Ctrl+Backspace
KeySend255=112,12,ThunderRT5Form,"PFD GLASS COCKPIT"
; Alt+F1
```

Note that quotes " " are needed around the Window title when given. They are also needed around the Class name if it contains any spaces.

Running and stopping programs via KeySend requests

The KeySend facility can also accept any of these keywords after the KeySend<n>=:

RunKeyN, CloseKeyN, RunReadyN, CloseReadyN, RunN, CloseN

This allows you to program KeySends to allow *any* of the programs, known to WideClient through these keywords (see earlier), to be started or stopped by key or button press, from anywhere in the system. For the **Run** variants to work the program must have been specified by that parameter, for the **Close** variants to work, that Close parameter must be present with the “Yes” setting.

PTT (push to talk) for Roger Wilco, AVC and TeamSpeak

There are two special forms of the KeySend parameter which are specifically designed to operate Roger Wilco’s Push To Talk (PTT) action. They also work on the Advanced Voice Client (AVC)—but not, it seems, on TeamSpeak (more on that later).

These are:

KeySend<n>=RWon

KeySend<n>=Rwoff

NOTE THAT YOU NO LONGER NEED TO USE THIS METHOD PROVIDED YOUR FSUIPC IS 3.50 OR LATER

The standard FSUIPC controls “PTT Transmit On” and “PTT Transmit Off” now perform the functions automatically whether local to FS or on a WideFS client. The information given here is still correct but is here for completeness only.

Just select appropriate KeySend numbers instead of <n>, and define matching KeySend parameters for your PTT buttons in the FSUIPC Buttons programming options. So, if you define these lines in the WideClient.ini file:

KeySend1=Rwon

KeySend2=Rwoff

Then simply define KeySends in the FSUIPC Buttons page to do the same thing. To do this, go to FSUIPC Options, find the Buttons page, press your PTT button. Now look in the FS controls drop-down list for “KeySend” and then set the parameter for the KeySend control to the KeySend number. In this case, you would assign the button or key *press* the KeySend control and parameter = 1 (to do Rwon), and the key button or key *release* the KeySend control and parameter = 2 (to do Rwoff). The numbers tie up the KeySend controls to the matching parameters in the WideClient.ini file.

This works well with Roger Wilco Mark 1 and Mark 1c, and with all versions of AVC as far as I know. It should be okay with other versions of RW, but it hasn’t been tested with them.

TeamSpeak is different. It doesn’t accept the direct messages WideClient uses for RW and AVC. But it can be made to work as follows. [*Thanks are due to Lee Glover for helping work this out*]

Choose a single unadorned key for the PTT operation from the list above. By “unadorned” I mean it must have no shifts – i.e. no shift, control or ALT. It also needs to be a Key which won’t mess up any other program you have running on the client, because TeamSpeak does not *swallow* the key, it simply acts on it and lets it pass! A suitable key might, for instance, be ESCape, or F12. Let’s take F12 as the example:

KeySend1=123,16 ; Press F12

KeySend2=123,24 ; Release F12

Now set **UseSendInput=Yes** (this parameter is described below).

Program the PTT button in FSUIPC, as described above. Operate the PTT and assign it in TeamSpeak. That’s it.

PostKeys=No: Normally all **KeySend** operations are performed using keyboard playback facilities in Windows—or SendInput if that options is selected. These are generally more reliable and have the advantage of reproducing exactly the same sequence of keyboard-related messages that would occur if the real keys were being pressed. However, they do seem to have a problem with

keyboard focus, and, whilst WideClient does attempt to change focus to the target window if it doesn't already have it, this can sometimes cause problems, resulting in missed or ignored keypresses.

If you have the target window class name, as described above, or you know that the application docks itself as a child of FS (and so WideClient), you can tell WideClient to **Post** the key presses instead of playing them back like a recording. This needs no focus changing, and works, *provided* that you get the Window class name right, that it is unique, and that the target program is happily processing WM_KEYDOWN or KEYUP messages and doesn't care about WM_CHAR messages or the timing relationships between all these.

To post key presses just set **PostKeys=Yes**. This works well with Project Magenta's PFD displays, and probably also the MCP.

UseSendInput=No: If this is set to **Yes**, then for all undirected KeySend key presses, WideClient uses the Windows **SendInput** method. This cannot be directed—whichever program has the focus at the time will receive the keystrokes. But it has the advantage that it can provide something that can be detected by programs using keyboard scanning rather than processing Windows messages.

Note that, even if this is set to **yes**, all directed KeySends will still use record-playback or message posting.

SendKeyPresses=No: [Only for use when running FS2000 (or later), or CFS2, under Windows 98/ME/2000/XP on the Server PC]. If you set this to **Yes**, then any non-system key presses (i.e. anything not including the ALT key) received by WideClient will be relayed to FSUIPC and thence to FS/CFS2. This has limited uses these days. For Wideclient to see the key presses it *must* have the keyboard focus. **THERE IS NOW A BETTER OPTION!** See the next section for programming key presses on the Client to operate “virtual buttons” on the Server.

ButtonKeys: making use of FSUIPC's virtual buttons facilities

FSUIPC offers facilities not only for programming real buttons and switches, but also up to 288 “virtual buttons”, represented by bits in part of its array of offsets. These 288 buttons are regarded, like real buttons, as being 32 buttons, numbered 0 to 31, on each “joystick”, with nine such “virtual joysticks” numbered 64 to 72.

WideClient provides facilities, using the built-in Windows “hot key” facilities, to convert trapped key presses into changes in FSUIPCs range of virtual button offsets. The effectively allows any controls, keypresses or other actions supported by FSUIPC on the Server PC to be instigated by key presses from the clients.

This obviously has major attractions for those using keyboard encoders in their cockpits and wishing to attach these to Networked PCs. Because the key presses are trapped using the standard Windows hot key facilities, their use is independent of the current keyboard focus. The only restrictions are on the range of such keypress combinations which are valid, and the fact that each such hot key is only validly claimed by one application, and once in that too.

To use these facilities, add the section [ButtonKeys] to the WideClient.ini file. Then, for each virtual button you want to operate, add a line in the format:

Tn=<keycode>,<shifts>

where **n** is the virtual button number (in the range 0–287): 0 being Joystick 64 button 0, and 287 being Joystick 72 button 31 (remember, each joystick has 32 buttons).

<keycode> is the usual virtual keycode—see the list provided earlier. Note that not all of them are usable as Windows Hot Keys. If you provide an invalid one (or one already in use here or in other programs) the line will fail. In this case the WideClient LOG file will contain an error message identifying the line in error.

<shifts> are:

- 8 for no shift keys, just the plain key
- +1 for shift
- +2 for control
- +4 for ALT
- +32 for “Win” (the Windows key)

So, for example, the **shifts** value for Ctrl+Shft would be $8+1+2 = 11$.

You should note that these are similar to the shift values used elsewhere in WideFs, but they are not the same. The differences are due to Windows restrictions on its Hot Key facilities.

Note that with this format of entry, when the Hot Key is pressed, the relevant virtual button is toggled: i.e. if not set, it is set, and vice versa. From the point of view of programming in FSUIPC this would look like a "Press" on the first press, and a "Release" on the second press, and so on, alternately. This seems to be the most flexible, as Windows Hot Keys do not supply any separate 'press' and 'release' indications. However, if you specifically want to Press a button with one keystroke and Release it with another, use these formats:

| | |
|--|-------------------------------|
| Pn=<keycode>,<shifts> | to Press the virtual button |
| Rn=<keycode>,<shifts> | to Release the virtual button |

*[Note that in pre-releases, before 6.45, WideClient used these parameters with no **T**, **P**, or **R** prefix before the button number. Don't worry—if you already have made use of these facilities, WideClient will automatically convert those lines to the **T** format.]*

GPSout relay facility

WideClient will now receive GPS data from my GPSout module installed in the Server's FS, and relay it to a local COM port. To do this you need to use GPSout 2.58 or later in the FS PC with its port to "WideFS". Then, in the client's WideClient INI file add this section:

```
[GPSout]
Port=COMn
Speed=n
```

... with, of course, the correct Port and Speed for the local connection.

Note that you can use the freeware Virtual Serial Port program (provided in the IGPSout package). This way you do not actually need any real COM ports to support a moving map program on the Client PC. In fact you can now use them on a Network with no actual serial cables involved.

You can have the same GPSout data being used on any of your client PCs. It is not restricted to one recipient.

For more details please see recent GPSout package releases.

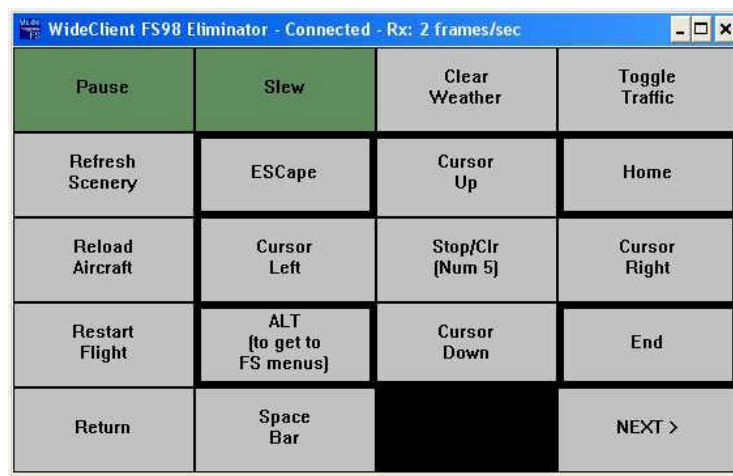
Button Screen Facilities

You can have the WideClient window itself turn into an array of "buttons". The window becomes a regular matrix of rectangular buttons, by default as many as will fit in the particular size of window set.

These are actually "virtual buttons" and operate the 288 virtual buttons provided in FSUIPC, numbered as Joystick 64, Button 0 up to Joystick 72, Button 31 (with 32 buttons on each of 9 imaginary "joysticks"). These buttons can, in turn, be programmed to do just about anything in FSUIPC's Buttons page.

The main use of this will be for touch-sensitive screens in cockpits, for all those FS-related (and perhaps Instructor Station-related) things you need access to but which certainly aren't part of a real cockpit. It beats having a mouse and keyboard around by miles!

As an example, look at the picture below, which is what appears on my Lilliput 7" touch sensitive screen (which, incidentally, is just right for mounting in an enclosed cockpit):



To use this new facility you need to place the parameter

ButtonScreen=Yes

into the [User] section of the WideClient.INI file, then add a completely new **[ButtonScreen]** section to the WideClient.ini file. Here's an example (this is for the screen above). Explanations are given below.

```
[ButtonScreen]
Size=4,5

0=T"Pause"
1=T"Slew"
2=B"Clear\rWeather"
3=B"Toggle\rTraffic"

4=B"Refresh\rScenery"
5=Ba"ESCape"
6=Bt"Cursor\rUp"
7=Ba"Home"

8=B"Reload\rAircraft"
9=Bl"Cursor\rLeft"
10=B"Stop/Clr\r(Num 5)"
11=Br"Cursor\rRight"

12=B"Restart\rFlight"
13=Ba"ALT\r(to get to\rFS menus)"
14=Bb"Cursor\rDown"
15=Ba"End"

16=B"Return"
17=B"Space\rBar"
```

Explanation:

The Size parameter is optional. It gives the desired button matrix size in terms of buttons wide by lines of buttons high. Here I've fixed it to 4 x 5 as you can see, as this suits my 7" display and fixing it like this allows me to develop it and test it, with that correct size, on another PC with a larger screen.

If you omit the size parameter then WideClient will fit in as many buttons of a finger-usable size as it can inside its Window. Just maximise the Window for the most on your screen.

The numbers 0= etc run from 0 to 287 max and equate to the 288 virtual buttons in FSUIPC. 0 = Joystick 64, Button 0 ... 287 is Joystick 72, Button 31. (These are normally written as 64,0 and 72,31 in keeping with FSUIPC's display).

In the part after the = the first character should be one of

B is for Button: this declares an ordinary button, i.e. push on, release off. These go Blue when you touch them.

T is for Toggle: push once, it's on, push again, it's off. Dull green when off, bright Red when on.

Then there are optional letters for border highlighting (thickening), so you can divide up the matrix on screen or highlight special areas. See the picture above where you will see its usefulness. The letters are:

| | |
|---|---------------|
| a | all 4 borders |
| t | top border |
| l | left border |
| r | right border |
| b | bottom border |

You can combine them, for instance ltr (in any order).

Then there's an optional Label for the button, in quotes (""). You can have up to 3 lines (with \r between them). The program doesn't automatically wrap them, you have to fit the text in yourself.

If there's no label the default label is "Button N\rJ,B" where N is the virtual button/line number (0–287) and J,B are the Joy,Btn numbers as seen by FSUIPC.

As not all 288 buttons will fit on one screen there's a PREV and NEXT button provided—these always take up the last two button spaces on each screen, but will be blacked out when not applicable.

If you resize the window the buttons are resized and rearranged accordingly. The definitions in the INI file are re-read as well when it is resized, so you can take advantage of that when making changes to avoid having to close and reload WideClient. For actual operation, though, it is best either to use the Size parameter, or to keep it maximised (set Visible=Max in the [Config] section, for example).

When WideClient is not connected, the buttons will all be blank and inoperative. They may actually flash if you are losing and regaining connections to WideServer.

To complete the above example, here's the FSUIPC [Buttons] section entries corresponding to the 18 virtual buttons programmed:

```
609=P64,0,C65794,0
610=U64,0,C65795,0
611=P64,1,C65731,0
612=U64,1,C65730,0
613=P64,2,K87,11
614=P64,3,C1009,0
615=P64,4,C65562,0
616=P64,5,K27,8
617=P64,6,K38,8
618=P64,7,K36,8
619=P64,8,C66512,0
620=P64,9,K37,8
621=P64,10,K12,8
622=P64,11,K39,8
623=P64,12,C65591,0
624=P64,13,K0,24
625=P64,14,K40,8
626=P64,15,K35,8
627=P64,16,K13,8
628=P64,17,K32,8
```

Note that, in order to get all the desired results some diting in the FSUIPC.INI file was necessary. Some of the keys enabled actually allow FS's Menus to be used. Pressing the ALT key on its own starts the process. To get this programmed in FSUIPC you need to tell it to produce a normal combination which it will accept, eg. Shift+Alt+A. This will give you an entry K65,25 in the file. Change that to K0,24 and you will get just the ALT. The "0" is a "null" character not featured on the keyboard. You will see this result in line 624 above.

The other one you may need to edit is the ESCape key. This is K27 as seen in line 616.

Incidentally, the Clear Weather key encoding is Shift+Ctrl+W, which is how it is programmed in FSUIPC's "Hot Keys" page. Buttons programmed to produce keystrokes will be detected as hot keys by FSUIPC or any application using FSUIPC's programmable hot keys facilities.

History of Recent Changes

Version 6.70 (July 2006)

- Fixed a bug in WideClient which prevented it connecting if the INI file specified both **Protocol** and **ServerIPAddr** values.
- Assorted internal improvements in preparation for future versions of FS.

Version 6.65 (June 2006)

- Automatic protocol setting added, with “ProtocolPreferred” facilities in the Server.
- WideClient supports new GoFlight device buttons and switches, provided the latest GFDev.dll is accessible (the latter is available from my Support Forum). This includes support for the Go Flight MCP Pro.
- WideFS now offers a facility to copy files. Any FSUIPC-interfacing application can ask for a file to be copied (renamed too if required) from any path to any other path anywhere on a currently connected WideFS network, whether Client to Server, Server to Client or Client to Client.

Normally the file to be copied would be resident on the application's own PC, but as UNC paths can be used, it doesn't have to be. However it does rather negate the point of the facility if a UNC path is used for the source file and refers to another PC—unless of course that PC is a Windows Server and has no connection limits.

The destination will be a file created on a connected WideClient or WideServer PC. A UNC path must always be used for the destination -- the computer name from the UNC path is used to create the TCP/IP connection.

Files are created or replaced. Any errors such as read-only status will be reflected in a "send" failure at the source, because the connection will be severed by the receiver.

Details of this facility will be published in the next Release documentation for WideFS. Meanwhile interested developers can obtain full programming details on application to me at petedowson@btconnect.com.

Version 6.60 (April 2006)

- This version includes many small improvements—minor adjustments for more efficient operation in some areas.
- It includes a new UDP protocol option. UDP is simpler and faster than TCP, and is installed on your PC as part of the TCP/IP package. It differs from previous protocols supported by WideFS in that it is “connectionless”—data is sent to specific addresses (by IP address), but there are no checks on whether they get there and no error recovery to ensure they do. This is why it is faster, and also why I've not supported it till recently. With the block sequence and checking now in WideFS you should at least be able to detect if it is going wrong.
- The method of selecting the protocol to be used is changed. The WideClient “UseTCPIP” parameter is gone. If that was previously set to “Yes” (the default) then nothing replaces it, and the protocol is chosen automatically by the Client at run time, trying TCP, SPX (the part of IPX/SPX actually used) and UDP in turn.

With a WinXP server and a WinXP client, you can add “ProtocolPreferred=XXX” (XXX being TCP, SPX or UDP) to the [Config] section of the WideServer.INI file, and those clients not otherwise committed will use that protocol if possible. This gives you a quick way of changing the protocol on all computers in the Network, for performance comparisons.

If you wish to select a specific protocol on any client, just add “Protocol=XXX” (XXX being TCP, SPX or UDP) to the [Config] section of the WideClient.INI file

Version 6.51 (December 2005)

- WideClient now correctly sets the program's local path when one is loaded with one of the “Run” parameters.
- WideServer now recovers better from any accesses attempted by Clients to illegal FSUIPC offsets (which actually vary with aircraft type rather faster than FSUIPC can tell WideServer). Instead of logging an unrecovered error then effectively starting again (thus disconnecting and reconnecting all clients), only that one transaction is voided. The logging looks the same and will still be useful to track down illegal accesses.
- Facilities to divert the Log files to other folders, on other PCs in the Network, have been added. However, it has been determined that these are almost useless in a standard Windows network setup because of the severe limitations Windows imposes on the number of simultaneous connections to, say, an administrative computer. It seems to only be truly viable on a setup built around Windows Server, instead of a peer-to-peer network and I have not been able to test it on such a setup. If any WideFS user wants to pursue this he should contact me via the Support Forum.

Version 6.50 (August 2005—there were no versions 6.48 or 6.49) includes these changes:

- Program closing by “CloseReady” parameters (and similar) is more consistently successful. Timing problems at the Server end sometimes either resulted in programs not closing when they should, or more usually closing but then being reloaded again before FS had finished terminating.
- When both Server and Clients are running on Windows 2000 or XP the Clients can now locate the Server without being told the ServerName, IP address or, for IPX/SPX even the Server Node (except in some multiple network cases, where the Node might be ambiguous).

This means that WideFS can simply be installed and, with no parameter changes, it should work.

This is accomplished by having WideServer broadcast its name and node using the “Mailslot” facility (a variation on NetPipes). The broadcast is a very short block and, by default, it is only transmitted once per second. But it will do this continuously, and, of course, being a broadcast it affects all connected PCs. In the unlikely event that this presents too much of a load on your network you can adjust this behaviour by changing the following parameter in the [Config] section of the WideServer.INI file:

AdvertiseService=Yes

Set this to "No" or "0" for no broadcasting, or set a desired time between broadcasts (1–10 seconds. Default is 1 in any case).

Note that if there are more than one PCs running FS with WideServer’s broadcasts enabled, it is pot luck which ones the Clients will attach to. In such a case you would still need to specify the Server details in the WideClient INI files.

If a client has attached to a broadcasting server (by using its broadcast) and that Server goes down (eg FS is terminated), then the Client keeps trying it for 10 seconds. After that it will listen for broadcasts again, so could attach to an FS on another PC.

- The parameter “ActionKeys” is now abolished. All WideClients listen for KeySends automatically.
- The Roger Wilco/Squawkbox/AVC PTT controls (RWon/Rwoff) are complemented by new ones for Squawkbox 3’s private voice PTT: PVTon and PVToff. These will work with SB3 when the SB3 update supporting the controls is available. [See next item also].
- FSUIPC’s controls for PTT and PVT now operate locally on the FS PC and on all WideClient PCs automatically. There is no need now to use KeySend controls nor to add the RWon/off or PVTon/off controls to the list in the WideClient INI file.
- The program load pathnames given in the Run, RunReady, RunKey parameters in WideClient.INI can now include command line parameters.

Additionally, for users of Project Magenta’s CDU program, the command line parameters can include one or more %P inserts. These are replaced by the name of the last “Company Route” loaded into the PM CDU. This allows ATC or route following programs to be loaded on a button press (by RunKey) with the name of the plan they need also to load. (A good application for this will emerge with the forthcoming Radar Contact Version 4).

WideClient will now receive GPS data from GPSout and relay it to a local COM port. To do this you need to use GPSout 2.58 or later in the FS PC and set its port to “WideFS”. Details are then added to the Wideclient.INI file where reception is needed. See the last section in the main part of the document for details.

- WideClient now features a Virtual Button-Screen option—see the section on this above for details.

Version 6.47 includes:

- Error trapping has been re-enabled in WideClient, just in case there are any more occurrences. Please check WideClient.log files if anything odd happens during a session. Don’t forget that you can restart WideClient and still access its previous log as “WideClient0.log”.
- The **ApplicationDelay** parameter now merely delays by “sleeping” if the value specified is less than 6, but loops processing messages and allowing other application calls when a larger delay is requested. In general, however, the default of 0 is still recommended.
- Neither Server nor Client now relies on Windows timers for timeouts, regulation or other time-dependent actions. Instead they use their own timer arrangements based on separate timing threads.

This seems to give better response times and altogether a tighter operation, especially on multi-client systems.

- The program “Run” options have been extended to allow programs to be started and stopped from the Server via the KeySend mechanism. An extra category of Run program is added (**RunKeyN**, where N runs from 1 to 9) which is never started automatically by Wideclient, but only via the specified KeySend arriving. The extra parameters are

RunKey1 to RunKey9, and CloseKey1 to CloseKey9

The KeySend facility can accept any of these keywords after the **KeySendN=**

RunKeyN, CloseKeyN, RunReadyN, CloseReadyN, RunN, CloseN

Thus enabling you to program KeySends to allow *any* of the programs known to WideClient to be started or stopped by key or button from anywhere in the system.

Version 6.45 changes are:

- A facility has been added to allow Hot Keys to be defined in Wideclient.INI that can be seen in FSUIPC on the Server as “virtual buttons” (one of 288 buttons provided on “virtual joystick” numbers 64–72). The buttons are defined in a new section [**ButtonKeys**], and, being implemented via the Windows’ Hot Keys facility, do not need WideClient to have the keyboard focus.
- The client send queue parameters and checks are revised in order to try to eliminate the occurrence of multiple reconnections due to excessive send queue build-ups on some folks systems. The reason for the build-ups has not been determined, because I cannot make them happen at all on any of my set-ups. The changes are:

1. The check is now controlled by the parameter **OnMaxSendQ**, which automatically replaces the previous one, and this defaults to “**Log**”, which simply logs the excessive queue and does nothing about it. Other values possible here are “**Flush**” which flushes the queue, and “**Recon**” which reconnects as well as flushing the queue. The last is the action that was occurring in 6.44. The danger is reconnecting is that the same problem immediately recurs, as on a reconnection all the request read data has to be re-requested, as if the client is a new one.
2. The **SendScanTime** parameter, which defaulted to 10, is replaced by **NewSendScanTime**, which defaults to 50. This limits the number of send queue scans to 20 per second, but it now does *not* limit the number of frames sent. Each single scan will send as many queued frames as it can before the Windows network indicates that one will block.
3. The **ApplicationDelay** implementation is changed to make the timing more accurate. Previously the value of 6, for instance, could have actually given rise to a delay of anything from 6 to 55, because of the granularity of the timer being used. The greater accuracy should make things run smoother still, but, now, to guarantee an average which approximates to the delays in WideClient 6.41 and before, the value would have to be set to something like 25.

I recommend strongly, however, that this parameter is left to its default of 0. If you have increased this in an attempt to get over problems, please restore it to 0 and see if the problems are better dealt with by the **NewSendScanTime** change above.

- WideClient now logs interim performance data as well as final performance data when closed. The interim values are logged whenever the Server sends out any type of Close message that doesn’t actually result in the Client closing. For instance, if the **AutoShutdown** parameter in the Server is set to “Apps” then, even if application closure is not allowed in the client, it will still log the performance so far when FS is closed.
- WideClient performance data now includes the maximum and average send data (frames and bytes). You will find that the averages tend to be surprisingly low, peaking really soon after connection (and reconnection) or when new applications are started. Upon closure, WideServer also logs an overall average “received” data line for every separate client which has been connected, but this is averaged over the whole session so the figures will tend to be less than those shown at the Client.
- Data for individual client applications is also now logged, referenced by its Process Id number (which is logged against its name when it connects). This data shows to number of FSUIPC_Process calls it is making and the actual total data size exchanged between it and WideClient. Hopefully this will be a useful aid to optimising applications.
- WideClient now sends its version number to the Server along with the Client PC name. This is logged by WideServer in the connection message and is a useful way to check that you do have all client PCs up to date. So that programs like the PM file checker can check this easily too, the *lowest* WideClient version number is provided in the 16-bit word at offset 3520, as the version number x 1000 in binary-coded decimal (BCD), e.g. hex 6450. If WideServer is out of date (before 6.442), or any one WideClient is earlier than 6.442, then offset 3520 will be zero.
- WideServer now provides the IPXROUTE data (with decoded ServerNodes) if the main details returned to it by Windows has a non-zero Network Number and a suspicious-looking MAC address for the adapter.

- Both WideClient and WideServer now rename the existing LOG file as "...0.Log" before creating a new one when started. This allows both clients and FS to be restarted without losing valuable diagnostic data, which can now be retrieved afterwards.
- More errors that previously would result in a message box report on screen are now merely logged. This is especially useful on Win98 and WinMe which seem to vary somewhat in how they report errors when one of the supported protocols is not installed.
- Additional data, items which do not change too much during a session, are now requested by WideClient on loading, so that they are available to applications immediately they connect. These include, for example, the FS install path, the aircraft name, and the Project Magenta NetDir field.
- A problem discovered in the way the "WaitForNewData" option operated is fixed in this release. When there are a number of different client applications all clamouring for different data on initial start-up, the "WaitForNewData" timeout was not always applied correctly—data requested by one application could, on arrival, sometimes effectively cancel the timeout pending for different data for another application.

This is a very old bug, but it has only really had any noticeable affect on things recently with the other performance improvements that have been made, and the ever increasing load placed on WideFS these days with so many demanding applications.

Version 6.44 includes:

- The WideClient "TimeOut" parameter is now removed, and replaced by "ApplicationDelay", which more clearly signifies its true action. The default has been reduced to 0, which actually suits most current programs better than the Timeout default. To get the original value, use 6 for this parameter, not 12—the original 12 milliseconds has been halved internally for some time. A non-zero value is more likely to be needed on Win98 or WinMe clients, because the timeslicing isn't so good with those operating systems.
- The "Priority" parameter has been added to the WideClient INI "Config" section. This allows the relative priority of the main program/application, and the now separate receive and send threads, to be changed should performance warrant this.
- Parameters are also added to controls the maximum length of the Send queue, and what action is to be taken in the event of this being exceeded. However, with the other changes made this should never need any changes from default values.
- The AutoShutDown action from the Server is followed by a one second delay before FS is allowed to terminate. This may improve the shutdown actions on Client PCs.
- The WideClient logging for the "monitor" option now provided separate timestamps for each separate application read/write call ("FSUIPC_Process") rather than lumping similar calls together.
- WideServer now records checksum errors in blocks correctly, even when no additional logging is requested, and if there are any it provides a total count in its performance summary at the end of the Log.
- WideServer now handles the reception of partial blocks correctly, re-forming them after arrival. The occurrence of such split blocks wasn't actually thought possible until recently—the Winsock protocols and buffering are supposed to deal with this.
- Wideclient now provides an additional method for sending Keystrokes as a result of KeySend parameters programmed in the FSUIPC Keys or Buttons pages (or indeed in the WideServer.INI file, where that original version of the feature is still used).

This method is by "SendInput" and operates in the same way as FSUIPC's keystroke programming for Buttons. It is enabled in Wideclient.ini by the line **UseSendInput=Yes**. The SendInput method is not directed—whatever program has the focus will receive the keystrokes for any undirected KeySends listed. All directed KeySend actions will still use the previous method—either record-playback, or, if **PostKeys=Yes** is set, message posting.

Note that this method should work well with TeamSpeak, for PTT actions from an FSUIPC programmed button.

- WideServer now detects the bad **ServerNode** which is obtained sometimes on Windows XP. It seems this is due to some "loopback adapter" XP installs. To find the correct ServerNode, WideServer now runs the WinXP utility IPXROUTE and extracts the Nodes it lists, and logs them in the correct form.

Version 6.41 includes these changes:

- Minor adjustments to improve performance a little. Now, providing the main FS PC is powerful enough to run FS with relative ease, *and* you have no particularly slow or overloaded Client PCs, you might be okay relaxing the FS frame rate limiter, or even setting it to "unlimited". Experiment first.

- WideClient can now be run before its Server PC is even switched on. It will simply keep retrying the connection. It will make a log entry every minute or so to tell you what's going on, but the advantage of this is that you can have WideClient starting up automatically (and loading your applications), and you don't need to worry about the order in which your PCs come on.
- WideServer can now automatically tell clients to either shutdown completely (**AutoShutdown=Yes**) or close their "CloseReady" applications (**AutoShutdown=Apps**) when FS itself is closed normally.
- There are no extra parameters for the WideFS INI files, but you might like to note that you can now set Monitoring for up to 8 separate regions—previously only one. This is by listing the offsets and sizes in the **Monitor** parameter.
- In the client PCs, monitored offset areas are now also logged when read/written by applications, not just when changed via the LAN. This has helped clear up some timing problems.
- WideClient now ensures that data associated with a timestamp is on WideServer's list when the data request is made, not leaving it till the data itself is first requested. This fixes some cases of incorrect data being read because the timestamp change is seen first.
- Finally, quite a big change: WideServer now automatically provides a service for both TCP/IP and IPX/SPX protocols if they are both installed. The "UseTCPIP" parameter is no longer used in the WideServer INI file (it will be deleted).

You must still specify, for each separate Client, whether TCP/IP or IPX/SPX is to be used for that Client. If IPX/SPX is installed on the Server, the WideServer log will always provide the "ServerNode" parameter needed for the INI files of those WideClients using IPX/SPX.

The main benefit of all this is that you can now mix IPX/SPX and TCP/IP clients. This solves the problems of IPX/SPX on mixed WinXP + Win98 setups (like mine, in fact). You can run IPX/SPX on those clients best suited to that, and TCP/IP where IPX/SPX is problematic (as with Win98 connecting to a WinXP server).

Version 6.401 fixes problems with the **WriteLocalDirect=No** option of 6.40, which was defaulted. The idea didn't work for several important programs. The parameter is now scrapped, and the original problem (the reason for the change) has been solved a different way—by having WideClient save up the results of writes to application offsets and send the results to the server when eventually connected.

Version 6.40 includes a number of changes:

- A new "shutdown" facility has been added, allowing application programs to be shut down (with WideClient still running) via a different WideServer hot key or application program action. The latter is by the pattern 0xDCBA being written to offset 0x3320 instead of the more severe 0xABCD. For more details refer to the **CloseAppsHotKey** details above.
- The number of programs that can be started and closed by WideClient has been extended. The **RunN**, **RunReadyN**, **DelayN**, **DelayReadyN**, **CloseN** and **CloseReadyN** parameters now operate with **N** running from 1 to 9, giving 9 programs startable immediately and 9 when FS is ready.
- Retries on frames which are blocked by Windows (Winsock, actually) are now made no more frequently than every 250 mSecs. Previously the normal FS frame controlled the retry timing. It is hoped that this reduction will reduce the occurrence of multiple retries, in the hundreds, sometimes reported, those which it is said seem to cause Project Magenta (for example) to act many seconds late. Whether it will help or not remains to be seen—the types of problem reported have not been reproduced here.
- Writes to application-owned offsets in FSUIPC are no longer simultaneously written to the local WideClient memory. If this causes any sort of problem, the former behaviour can be re-instated by setting **WriteLocalDirect=Yes** (see above) in the WideClient.ini file, but in general it is considered safer not to.
- WideServer now performs at intervals of 250 mSecs (¼ sec) when FS is engaged in menus and such, rather than 5 seconds as it was previously. The very slow rate was intended merely to stop Clients timing out and attempting reconnection, but it seems there are applications for some add-in modules to retain client interactions whilst in their own menus. The 4fps rate is still much slower than a normal running rate but should be sufficient for this.
- Originally included in version **6.233** (a limited release) are improvements in the handling of timestamp reads (see next item). When timestamp reads are combined in one process with other IPC operations, especially writes, the timestamp read is separated out and dealt with first. This ensures that they are seen to change later, should any of the writes prove to be one of the commands instigating a timestamp change.
- Similarly, an earlier limited release (**6.232**) included changes to assist with those programs which use facilities in FSUIPC that rely on timestamps changing to know that specific requests have been fulfilled. To be specific, these are the IPC based

requests for path information, airport runway-in-use, and weather setting and reading, especially for specific weather stations on FS2004. (These changes were originally in a version **6.231** but did not work as desired in that version).

Version 6.23 contains a fix to the RestartTime facility for a bug that could, in certain circumstances, cause WideServer to continually restart so causing every Client to reconnect – at 10–15 second intervals! Before this fix the problem could be averted by setting the **RestartTime** parameter to zero or something larger than 20.

Additionally, in this version, the settings of all of the [Config] parameters are added to the INI file so that, in future, it will be more obvious when they are different from previous versions. This may have allowed the RestartTime problem to be diagnosed earlier.

A fix to the way Wideclient updates the local memory copy of FSUIPC offset data is included. Before this version, when an application wrote to an FSUIPC offset with no connection to the Server yet established, the local memory, in WideClient, was still updated. Now this doesn't occur until the connection is made.

Version 6.222 contains a change to WideClient only, this being the addition of the “AllowShutdown=AppOnly” variation described in the text. WideServer remains at version 6.221.

Version 6.221 is a minor release that only affects users with GoFlight GF-TQ6 throttle quadrants when used on the client. The changes merely fix the problem of fast/slow dial turning on GoFlight units like the GF-RP48 when a TQ6 is also present.

Version 6.22 includes new facilities to recognise buttons on Joysticks, EPIC and GoFlight equipment connected to client PCs, and transmitting button events to the Server for processing in FSUIPC's Buttons programming page. This needs FSUIPC version 3.20 or later.

The only other change has been in WideServer, to speed up the connection on initial load using the FS2004 “ready to fly” indicator provided by FSUIPC. Timing on FS2002 and earlier is not affected.

Version 6.101 was originally only a small change in WideClient.exe only (the Server DLL was still version 6.10). WideServer was updated to 6.101 later. The change in the Server is minor: merely to prevent an error message complaining about the failure to send a block for more than 5 seconds being repeated for every further retry of the same block thereafter. This is a cosmetic, not a functional change.

The earlier change in WideClient only affected Project Magenta users. It helps eliminate some apparent pausing when the MCP or FMS modules are run on the same PC as one of the PFD glass cockpits. The change is to treat the lower memory allocated to PM (04E0–0537 inclusive) in the same way as the upper offsets (4000 and higher). These are updated locally as well as transmitted to the Server, whereas the normal FS variable areas 0000–3FFF are only transmitted to the Server, as the read-back is not always the same. I cannot understand why this potential problem has never been identified before, as it would have been so for years. It was only reported recently!

Version 6.10 includes two changes in the Server part only:

- It provides a short period of grace (3 seconds) during which WideServer will tolerate lack of access to FS variables without assuming an extended “stop” situation, and
- It avoids treating any longer stoppage as a necessary reason for restarting all the services to Clients. This latter change is optional (via the **NoStoppedRestarts** parameter) but is enabled by default.

Version 6.02 fixes a timing problem which can occur if WideFS gets loaded by FS before FSUIPC, or even very soon after. It seems that whilst FSUIPC is verifying the Registration of WideFS, the WideServer DLL may already have checked access and been rebuffed. This has been fixed by making WideServer retry for the access permission 10 times over a period of 20 seconds, which should cover all eventualities at start-up.

Also, the start-up has also been made a little faster, and the message “stopped” (which worried some folks) is replaced by “getting ready” or similar. This is actually more accurate at start-up, even though internally it is the same—it is the access to FS innards which is stopped until FS's aircraft and panels and so on are fully initialised.

Version 6.00 works on FS2004 as well as earlier versions of FS, and needs at least version 3 of FSUIPC. In addition to the changes explicitly for FS2004, WideFS is improved so that you don't get continuous re-connections from Clients when FS is in menu dialogues for long periods, and it also doesn't reset any data to zero when a long break occurs. However, data is not updated during such times. It does now maintain contact *and* data updates when FS is minimised, albeit at a much lower frame rate (something like 2–4 fps).