WideFS: Techical Reference

by Pete Dowson, 6th November 2008

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

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).

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.

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.

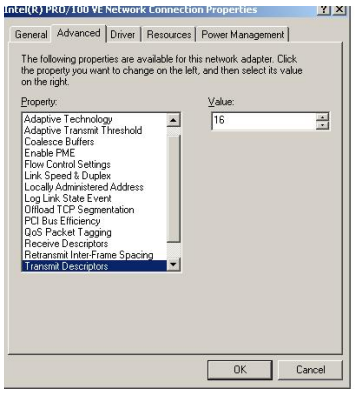
9. Network adapters can go wrong!

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 reties …”, 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.

WideServer INI file options

Note: these are in the [WideServer] section of the FSUIPC4.INI file on FSX and later.

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 (… but the [WideServer] section on FSX,)

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 witohut 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=0: 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 – try 10 or 20 seconds. You may also find it useful to use the 'Restart Hot Key' facility, described below.

This feature is disabled by setting the RestartTime to 0, and this is now the default setting as the restart has been found to cause some stutters with some Network drivers.

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 and 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 defalut 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 mpbs, USB, Serial or Parallel) then you may find 2048 better.

The [User] Section (… but still the [WideServer] section on FSX)

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 shown in an Appendix to this document.

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 Fligfht 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 WidevieW, 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
OnTop	same as 'Yes', but set to stay on top of other windows
OnTopMax	same as 'Max', but set to stay on top of other windows

The main use of having a visible WideClient window is for a ButtonScreen, as described fully in the User Guide. Apart from that, a Window may be needed for applications to "attach to", but there aren't many that do. The FS98 EFIS program 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 is defaluted 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 king 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 WidevieW), 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. To forcibly terminate obstinate programs, substitute "**Kill**" for the "**Yes**", but be aware that this method is the equivalent of using the Windows Task Manager (Ctrl_Alt_Del) to delete processes. Use it with care. It provides no opportunities at all for the murdered process to do any sort of tidy up or options saving.

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. You can use **Kill** instead of **Yes**—see the description and warning under "**Close1** ... " etc above.

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. You can use **Kill** instead of **Yes**—see the description and warning under "**Close1** ... " etc above.

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=TToolWindow97

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 in the Appendix. 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.

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 by 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 in the Appendix, below. 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.]*

Appendix: Windows Keycodes

Here are the keycodes which are usable in several of the WideFS parameters. This is a complete list, but they may not all be usable in each context.

8	Backspace	76	L	119	F8
9	Tab	77	M	120	F9
12	NumPad 5 (*NumLock OFF*)	78	N	121	F10
13	Enter	79	O	122	F11
19	Pause	80	P	123	F12
32	Space bar	81	Q	124	F13
33	Page Up	82	R	125	F14
34	Page Down	83	S	126	F15
35	End	84	T	127	F16
36	Home	85	U	128	F17
37	Left arrow	86	V	129	F18
38	Up arrow	87	W	130	F19
39	Right arrow	88	X	131	F20
40	Down arrow	89	Y	132	F21
45	Insert	90	Z	133	F22
46	Delete	96	NumPad 0 (*NumLock ON*)	134	F23
48	0 on main keyboard	97	NumPad 1 (*NumLock ON*)	135	NumPad Enter or F24?
49	1 on main keyboard	98	NumPad 2 (*NumLock ON*)	144	NumLock
50	2 on main keyboard	99	NumPad 3 (*NumLock ON*)	145	ScrollLock
51	3 on main keyboard	100	NumPad 4 (*NumLock ON*)	160	Left Shift **
52	4 on main keyboard	101	NumPad 5 (*NumLock ON*)	161	Right shift **
53	5 on main keyboard	102	NumPad 6 (*NumLock ON*)	162	Left Control **
54	6 on main keyboard	103	NumPad 7 (*NumLock ON*)	163	Right control **
55	7 on main keyboard	104	NumPad 8 (*NumLock ON*)	164	Left ‘Menu’ **
56	8 on main keyboard	105	NumPad 9 (*NumLock ON*)	165	Right ‘Menu’
57	9 on main keyboard	106	NumPad *	186	; : Key*
65	A	107	NumPad +	187	= + Key*
66	B	109	NumPad -	188	, < Key*
67	C	110	NumPad .	189	- _ Key*
68	D	111	NumPad /	190	. > Key*
69	E	112	F1	191	/ ? Key*
70	F	113	F2	192	# ~ Key*
71	G	114	F3	219	[{ Key*
72	H	115	F4	220	\ | Key*
73	I	116	F5	221	] } Key*
74	J	117	F6	222	' @ Key*
75	K	118	F7	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.

Published by Peter L. Dowson, 6th November 2008