FAQ

Q: Why on earth would I ever want to use TINY when I can just run my DOS programs directly on my Windows machine using Virtual PC or VMWARE?

I love Virtual PC and think it a great tool, but here are a few examples (taken from actual TINY users) when you might still need TINY...

* The DOS machine needs to be physically located in a remote location (like inside a nuclear power plant smokestack or on top of denim cutting machine) because it is directly connected to some sensors and controllers.

* You only have a license to run your DOS software on a single physical machine, but you want multiple people in multiple locations to be able to use the software.

* Your DOS software is incompatible with Virtual PC.

* You don't want to use a Windows machine because your DOS machine hasn't once crashed in over ten years and you like it that way.

* You can't afford to buy a new Windows machine(s), especially when you already have a perfectly good Turbo XT machine(s).

Q: I am able to connect to the remote TINY host and even type on keyboard, but I just see a blank screen on the client, what should I do?

While the TINY host does the best job it can of auto-detecting the current screen mode and active page, in DOS this is not always reliable (especially with non-VGA video cards). You can try manually selecting video modes from the pop-up menu on the TINY client until you find the mode/page your host is in. Once you find the correct screen mode for a given video card/application, you can look up the mode number in the TINY status window and specify this number on the client command line using the "/M" parameter. This way the TINY client will launch already in the correct mode next time.

Q: TINY works fine when I connect from a machine on my local network, but it doesn't work when I try over the internet.

It sounds like the UDP packets that TINY uses are getting blocked somewhere between the host and client when you try to communicate over this internet.

This could be due to a firewall or NAT router on either side of the connection. Make sure that you do not have UDP port 25443 (the default port Tiny used) blocked anywhere.

Also, if you have a NAT router, you might have to add a Port Forwarding entry for UDP port 25443 that points to the TINY host machine's IP address on the local network.

Q: Can I have multiple TINY clients viewing the same host simultaneously?

Absolutely, as long as you have enough bandwidth to carry all the packets. Try using higher "/D" settings on the client if bandwidth becomes an issue.

Having multiple clients can open up interesting opportunities. You could make a very simple billboard system by setting up a single DOS host and giving one person the control password and all the other people the view password. The person with the control password would be able to type messages on the screen, while the other people could only view the messages as they were entered.

You could even use TINY's ability to manually select between the various host video pages to set up a pseudo-multi user system. Imagine you had a PC that was controlling a set of conveyor belts via its serial ports. You could write a simple application that showed the speed and load for each belt on a different DOS text page. By having different people select different DOS pages to view on their clients (either manually via menus or with the /M command line parameter), you could have four different people watching the different screens at the same time on the same DOS host. By carefully picking non-overlapping control keys (belt one uses "Q" to speed up, "A" to slow down; belt two uses "W" and "S", and so on), these people could even simultaneously and independently adjust their respective belt speeds. Pretty cool, huh?

Q: When I restart the host, the client continues running without updating the screen. It would be nice to say something like "Lost connection to Tiny Host" and blank the client screen.

I think this is a matter of taste. I actually prefer the current behavior. After I reboot, I like to leave the client running so that I can see it start to update again the moment the host machine comes back up. Also, for the client to detect that the host was about to restart, the host would have to send some sort of special "I'm about to reboot" packet to the client. This makes things very complicated. What if that packet got lost? So you would need the client to send a acknowledgement "I heard that you are about to reboot, go ahead" packet back to the host. What should the host do if it never gets the acknowledgement? Then it could never reboot! Since you (or at least I) usually have a good reason to do a reboot, I usually want it to be as quick and direct as possible with no chance that it could not happen.

Q: I found that I can start the client and point it anywhere and the client will start but with a blank screen. It would be nice to return something like "No Tiny Host found at this address".

TINY uses UDP, which is a connectionless protocol. This makes it efficient and simple, but also means that there really is no connection between the client and server, just a bunch of unreliable packets. There is no difference between pointing it to nowhere and pointing it to a valid host and just having the packets get dropped. Interestingly, this is also a problem with TCP/IP, it is just hidden so most people don't think about it.

There are three ways to know if you are actively getting updates or not...

* The cursor blinks every time a new update packet is received from the host.
* The program window icon blinks every second when updates are coming in.
* You can watch the status window and if you don't see packets coming in then you know that the screen is not updating.

Once you get used to it, you will intuitively know that your connection is dropped or at least lagging by watching the cursor.

Keep in mind that there *ultimately* is no difference between a non-connection and just a sequence of packets getting lost. Networks are not magic. With most connection-oriented protocols (i.e. TCP/IP), the protocol specifies a fixed amount of time to wait for packets before arbitrarily giving up and declaring a connection to be "lost".

With UDP (and thus TINY), it is up to you to decide if the screen is not updating enough for the connection to be considered down, and that really depends on your application. If not enough packets are getting through, then there really is nothing you can do but wait anyway, restarting the client will not change anything since the problem is some where between you and the host and as soon as the problem is solved and the packets can get through, the client will start updating again.

Q: How can I run TINY from a Windows shortcut without starting a Command Prompt window?

One option is to right-click "properties" on the shortcut option and select "run minimized". This will start the DOS window minimized, although it will still be there.

Set the shortcut target to point to "javaw.exe" rather than "java.exe" if you don't want to see the dos box at all. JAVAW doesn't start a command window, so you won't see any error messages that might come up. TINY doesn't really print any error messages after it starts, but Java itself might rarely (I.e. OutOfMemory exception, NetworkHardware exception, you get the idea).

Q: How much bandwidth does TINY use?

Here is how TINY works: Client sends request packet to host, then host responds with a screen snapshot packet. By default this happens 10 times a second (100ms delay between requests). Each request is tiny, less than 100 bytes. Each response is a full screen worth of data- 80x24x2 = about 4K bytes. So, by default TINY uses about 0kb/s up and 40kb/s down bandwidth for viewing DOS text screens.

Depending on your application, you can probably reduce the update rate to once or twice a second (500ms or 1000ms delay) and greatly reduce the bandwidth.

Note that graphics screens have a lot more data, so either they update less often or use more bandwidth depending on how you set the update rate.

Q: Is TINY secure?

No! TINY is extremely *not* secure. Anyone with a network sniffer attached to your wire will easily be able to see every key you press, everything on your screen, even your TINY password. Note that this is also true of VNC and many, many other protocols you probably use every day. In the end, good security is very hard to do correctly, so I'd rather make TINY "explicitly insecure" than "probably secure" like most programs. If you want a secure, encrypted version of TINY, let me know and I'll try to help out, but I bet I can find lots of other ways people will be able to get the same data from you regardless of how secure TINY is.

Q: Graphics screens are so darn big, how did you squeeze them though such a tiny (pardon the pun) wire?

TINY uses a clever technique of "slicing" the graphics screen into lots of little pieces (40 of them for 640x480x16 mode) and sending each slice in a UDP packet. This is why you see that psychedelic painting pattern when watching the TINY client paint a graphics screen. This is nice because it limits network bandwidth, reduces the maximum UDP packet size to something that DOS stacks can handle, and does not require a memory-wasting buffer on the memory limited DOS machine. The downside is that you never see a single snapshot of the remote graphics screen for that screen is constantly being updated.

In VGA 800x600x256 mode, TINY also uses a technique called "nibble compression" where it takes 256 colors on the host screen and compresses them down to only 16 colors. This uses half as much bandwidth, but means you don't see as many colors. You can manually select the full color mode on the "Screen Mode" menu.

Q: Can you make TINY support the Microsoft DOS Network Client 3.0 TCP/IP stack?

Thanks to Mark Phinney at cflsb.com, I now have the Microsoft TCPIP SDK files so in theory it should be possible to make TINY work with this stack.

Unfortunately, it looks like the Microsoft stack limits the packet size to 1400 bytes so it would take some effort to make it work efficiently. If you REALLY need to get TINY working on this stack, let me know and maybe we can make it a project.

Luckily the Novell stack is free and good, so most people are happy with it.

Q: Can you make TINY support the WATTCP TCP/IP stack?

While certainly possible, it would take some effort to make it work efficiently. If you REALLY need to get TINY working on this stack, let me know and maybe we can make it a project.

Luckily the Novell stack is free and good, so most people are happy with it.

Q: Can you make TINY run with no extra TCPIP stack?

I'd actually love to start a project to have TINY talk directly to the network card, probably using a Crynwr packet driver. This would reduce the amount of memory needed on a TINY host and with one less piece of software things would generally be simpler to set up. Let me know if you have a good reason to need this.

Q: How do I use custom keyboard translation tables?

Use a text editor like NOTEPAD to take a look at the sample translation files to get a feel for the very simple format. To load one or more translation files, use the /X command line parameter when loading the client. For example, to load the TINY client with both the sample translation files, you'd type something like...

```
java -jar TinyClient.jar /Xmathkeys.txt /Xgermankeys.txt x.x.x.x password
```

...assuming that you already saved those two files into the current directory.

Q: When I connect to the TINY host, I get a "Host mode unsupported" message.

There are a huge number of graphics modes available on the various brands and models of graphics cards and there is no standardized way to access these modes in DOS, so TINY only includes support for some of the most popular modes.

First, make sure you are using the "extended" version of the TINY host that includes graphics support. If you try to view a graphics screen with the standard version of the host, you'll always get the "mode unsupported" message.

If you using the "extended" host versions and are still seeing this message, then your application is using a mode that TINY does not know about.

Try to configure your application to use one of the modes that TINY does support. You can see a list of these modes by running the TINY client with the /M command line parameter for a list of supported modes.

If you really need to use a specific application with TINY and that application can only use a graphics mode that TINY does not support and you'd be willing to pay for the work to add support for that mode to TINY, get in touch with me at the email address below.

Q: How can I override the default UDP socket that TINY uses?

You can use optional command line parameters to specify what UDP port the host should listen on and what port the TINY client should call out to. By default, both use port 6363 hex.

For example, the command line…

```
TINYH_N /P6364 josh josh
```

…will load the TINY host and have it listen on port 6364 (hex) with the view and control passwords “josh”.

You can then start the client like this…

```
Java -jar TinyClient.jar /P6364 192.168.1.2 josh
```

…and it all should work (assuming the IP address of the TINY host is 192.168.1.2 and the two machines can PING each other).

Note that it is even possible to load multiple copies of the host listening on different ports on the same machine, although this uses up lots of memory.

Q: How can I improve TINY performance?

The TINY client is not a particularly efficient Java application. My goals were to make it small (tiny?) and simple rather than fast. Since it runs on the non-DOS side of the connection, I assume this machine will be pretty modern and fast. The TINY host program, on the other hand, is as efficient as possible since I don't want to waste a single precious CPU cycle on your tired old 2.8Mhz PC-AT.

Make sure you are using the very latest version of the Java runtime - the most recent once is better than the previous one. I'm not sure why, but it is cleverly hidden on the Sun website here...

http://java.sun.com/javase/downloads/index.jsp

...you should pick the "Java Runtime Environment (JRE)".

You can also specify how often you would like the TINY client to request a screen refresh from the host on the command line with "/Dx" where "x" is the number of milliseconds to delay between requests.

For example, launching the TINY client with this command line...

```
java -jar TinyClient.jar /D500 x.x.x.x password
```

...will send a screen refresh request twice per second (500 milliseconds = 1/2 seconds).

Using longer delays (and thus lower refresh rates) will reduce the CPU load on both the host and the client and will also reduce the bandwidth requirements. The downside to lower refresh rates is that you will see the screen update less frequently. This can be annoying if you are using an interactive application, especially when you are trying to type quickly.

On the flip side, specifying too fast a refresh rate (too low of a delay) can cause annoying problems as well. If you set a rate that generates more data than your network connection can handle, packets will start to get dropped. Once this happens, performance degrades very rapidly and unpredictably. This once happened to me and it took a while to figure out what was wrong. I had specified a refresh rate that seemed to work fine and gave me snappy response when I was typing on the remote machine, but every once in a while everything would grind to a halt and my keystrokes would start taking several seconds to show up. It turned out that someone else was occasionally TINYing onto a different machine over the same DSL link and between the two of us we were saturating the connection. We both slightly reduced our refresh rates and everything worked great.

The default refresh rate is 100ms, which translates to 10 refreshes per second. This is fine for a local Ethernet network connection, but will probably not work well over the internet. For a 768KB up DSL line, I typically use a 500ms delay. This leaves enough bandwidth available that two people can TINY across the connection at the same time without saturating our internet connection.

Ultimately the correct setting for a given installation will depend on the application and the connection, so you'll probably need to try several different values to find what works best for you. Start with a high delay and work your way up to until you find a delay that is not annoying to use and doesn't saturate the connection. Turn on the TINY status screen on the client and watch the dropped packet counters. A few occasional dropped packets are normal, but if you start seeing the counters incrementing regularly, you'll probably need to throttle back and use a higher delay. Keep in mind that other things going on over your network connection or the one at the remote machine can use bandwidth too, so you are more like to get see dropped packets if someone in your house starts downloading a Justin Timberlake video on YouTube.

Q: What are all the TINY command-line parameters?

Client Syntax:

TinyClient [optional params] host_address password

Parameters :

/? = displays this help info

/Dn = sets the delay between screen requests in milliseconds (default=100)

/Kn = sets the delay between key requests in milliseconds (default=500)

/R = disable popup menus on left click (default=left or right click)

/X = key_file loads a keyboard translation table file (you can load multiple key files)

/C = show key scan codes to console for diagnosic purposes

/Px = sets target UDP port in hex (default=6363)

/S = show status window on startup (default is don't show)

/Ln = sets the startup screen scaling size (default=1, 0=full screen)

/Mn = sets the start-up screen mode (default=automatic)

/M = lists available screen modes

/F = specifies an optional screen font file to load

host_address = is the IP address or name of the host machine (required)

password = is the password for the host machine (required)

Host Syntax:

TINYH [/D] [/Px] view_password control_password

Where :

/D = enables debug mode

/Px = sets UDP port to listen on in hex (default=6363)

/I = enables sending a keyboard interrupt after each keystroke which is needed by a very few (one so far) applications

view_password = is the password that will let clients see what is on the host's screen, but not send keystrokes or reboot the machine

control_password = will allow the client full control over the host machine

Q: How does debug mode on the host work?

If you load the TINY host with the /D parameter, it will start in debug mode. In this mode, the TINY host will write two letters to the upper-left corner of the text screen whenever a packet is received. This can be useful in debugging problems where you are not getting any response on the client.

The letter in the first position simply cycles each time any request packet is received by the host. If you see this blinking/spinning when you run the client, at least you know that there is good connectivity between the host and the client and they are both set to use the same port.

The second letter lets you know how the received packet was processed. Here are the possible codes...

Code	Meaning
R	A packet was received. This should only show up for the split second between after the packet is received and before it is processed, so you probably won't see it.
9	The packet received was ignored because it was less than 9 bytes long. This maybe means someone is trying to spoof/hack you TINY host since the TINY client would never send a packet like this.
K	An old style key request was received with the wrong control password. Check that your password on the host and client match, and consider upgrading the client to the latest version too.
V	An new style key request was received with the view password. Use the control password on the client if you want to send keys.
L	An new style key request was received with a totally wrong password. Check that your password on the host and client match.
X	A view request was received, but the password did not match either the view or control passwords. Check password on the client.
B	A reboot request was received that did not match the control password. Check the password on the client.
F	An unknown packet was received. This again probably means that someone is trying to hack your TINY host, or another application is mistakenly sending UDP packets to the port that the TINY host is listening on.
S	Success. The most recently received packet was processed successfully and a response packet was sent, so you should be able to debug the problem on the client.

Q: How can I get TINY client to prompt for the password each time I connect?

In Windows, you can use a batch file to ask for a password and then launch TINY with the password as a parameter. Try something like...

TinyPass.bat:

```
 Set  /P pass=Please enter your TINY connection password:
 %windir%\System32\javaw.exe -jar TinyClientNew.jar /D400 192.168.1.113 %pass%
```

Q: How do custom fonts work?

To load a custom font on the Tiny client, you'd use the /F command line parameter like this...

```
java -jar TinyClient.jar /Fsmall.tf 192.168.15.1 foobar
```

...where small.tf is one of the sample Tiny fonts.

Here are sample fonts in a collection that I've made just to give you an idea of what is possible...

small.tf A	Tiny 5x7 font designed for LCD displays.
normal.tf	The standard VGA 9x16 font. Good base for making a custom font.
big.tf	A huge 14x23 font I converted from an ASCII TrueType font.
cyrillic.tf	A normal VGA font with some Cyrillic letters in the high ASCII range.
arabic.tf	A normal VGA font with Arabic letters in the high ASCII range.
ibm_arab.tf	An IBM supplied Arabic font.
script.tf	A silly font to give some ideas of what is possible.

Note that if you use a font on the client that is a different height in pixels than the font on the host, your cursor might look different on the client than on the host. For example, if the host machine is using the standard VGA font that is 16 pixels high, but you are loading the "small.tf" font on the host which is only 8 pixels high, you might not see the cursor bar on the host since it is located at pixel location number 14 which is lower than the bottom of the character on the host.

Q: Sometimes when I manually pick a screen mode, I see "Received Corrupted Frame" on the Java console and the client screen stops updating.

Despite the ".tf" extension (for Tiny Font), the font files are just plain ASCII text files, so you can edit them with notepad or any other handy editor to make your own fonts. If you make a particularly useful one, please email it to me so I can share it with everyone else.

I also have a tool that I wrote to dump the font table on a VGA card into a TF file, but it would take me longer to explain how to do this than it is worth. If you really need to dump a special font, let me know and I'll walk you though it. I also have a tool that can convert some TrueType fonts into TF files.

Q: Sometimes when I manually pick a screen mode, I see "Received Corrupted Frame" on the Java console and the client screen stops updating.

If you request an invalid mode (say, Hercules Page #1 on a machine that doesn't have a Hercules card installed), the host machine may try to send packets based on memory that doesn't even exist on the host machine. The contents of the packets are typically all 1 bits, so the TINY client can't find the appropriate data fields in the packet. This does not cause any harm or problems for either the host or client machines, and you can fix it by just selecting a mode that is supported on the host machine.

Q: I just upgraded to the new version of TINY and now I can not see the screen on my host.

The latest version of TINY introduced a new protocol that is much more efficient. If you can seamlessly connect to a host running the new version with an old client, but to connect to a host running the old version using the new client you must select "Auto-Legacy" under screen modes. You can easily do this from the TINY client command line or using the pop-up menus. Or better, upgrade your host to the new version too!

Q: Do you know where I can find the Novell FTP Client?

Here is a package that contains a bunch of handy utilities that work with the Novell stack including FTP client and server, TFTP client and server, TELNET, RCP, and even a TSR that lets (supposedly) you connect to you DOS machine from TELNET or X-Windows (but I've never gotten it to work).

Here is an example of an FTP script file called script.ftp that logs into an ftp server at ip address 11.22.33.44 using user username root and password fubar and downloads all the files in the default directory:

```
open 11.22.33.44
user root fubar
mget *.*
quit
```

Here is an example of a batch file to execute the above script...

```
ftp -X <script.ftp
```

You might also want to check out JoshFTP. It is like the Novell FTP client, except that it doesn't crash all the time. :)

Q: Do you know where I can download the latest version of the Novell DOS client?

Yes. You can download it directly from Novell.

The TINYDISK.ZIP file also includes all the Novell files you'll need besides drivers for your network card.

Q: How can I send files to/from the TINY host over the network?

I think the best way is to use an FTP client on the DOS machine. This requires an FTP server somewhere, but there are lots of free and simple FTP servers that you can install on your non-DOS machine.

If you are using the Novell stack on the DOS machine and have a Win2000 or Win2003 machine on your network, you could load Microsoft's free File and Print Services for Netware service on the Windows server and then map a drive on the DOS machine using the Novell MAP command.

If you have a spare Linux or BSD box on your network, you could install mars_nwe so you can use that machine as a file server for your DOS box.

Another option is to make the DOS machine an FTP server by running the Novell FTP daemon on the DOS machine and connecting to that with a normal FTP client from your Windows/Linux box. Make sure to add your username:password in the \net\hstacc\ftpdusr.log file on the DOS machine. After transferring files end your FTP program and TINY will then regain full control of the DOS machine. (thanks Joe Murphy!)

You may also be able to transfer files to a Windows SMB server using the MS Client running concurrently with the Novell Client32.

Q: Is there any documentation about the /Ln parameter (i.e. what the values of n can be and what they mean)?

The specifies the screen scaling factor that the TINY client should use at startup. Higher values make for a larger (and blockier) client window.

"/L1" specifies that one pixel on the host should correspond to one pixel on the client (the default),
"/L2" specifies that the client window should be twice as big as the HOST screen in pixels (each host pixel corresponds to a 2x2 block on the client), etc.

Q: Is there any documentation about the /Mn parameter (i.e. what the values of n can be and what they mean)?

This specifies which screen mode TINY should use on start-up. You typically would want to let TINY auto-detect this, but you might want to specify it explicitly in cases where TINY can't accurately auto-detect which mode to use.

One example is if you have a two monitor set up on the host, TINY can't know which monitor you want to see so), or if the host is in a Hercules Graphics mode (there is no reliable way to detect Hercules modes).

If you run the TINY client and do not specify a value for the /M parameter, it will list all available modes.

Q: Can the size of the client window be changed dynamically by the user when connected.

If you click inside the TINY client window when connected, you can use the "Scale Screen" submenu to change the size of the window.

Q: When I run TINY on my DOS machine using the PCTCP stack, it crashes/stops responding.

Make sure your config.sys has a STACKS=16,512 line in it.

Q: I can get the Novell TCPIP stack to work on a virtual DOS machine, but how the heck do I get it to work on my actual DOS machine?

You'll need to load a suitable Novell Client32 driver for your card. These drivers always end with a ".LAN" extension. You can find drivers for most common network cards on the Novell website. You can also usually get the driver file from the card's manufacturer. If you can't find the right driver anywhere, consider just buying a new (well supported) card since they are so cheap now.

Look in the STARTNET.BAT for the line that says something like...

LOAD E100B.LAN

This is the line that loads the network card driver for an Intel Pro Ethernet 100B network card. Copy your driver into this directory and then change that line to reflect your card and everything should work. Note that some cards need extra parameters at the end of the line. These are different for every card, so you'll have to figure it out yourself. Try looking at the card's documentation, or just experimenting. Sometimes the driver will give a useful error message when you use the wrong parameters and that message can help you guess what the right ones might be.

Q: My card doesn't have a 32-bit Novell driver (ends in .LAN), just a 16-bit ODI Novell driver (ends in .COM). Am I screwed?

You can also use 16-bit Novell ODI drivers, you just need to load some "shim" drivers that make it possible for the 32-bit stack to talk to them. The only downsides are slightly lower performance and slight higher conventional memory use compared to a real 32-bit driver. And it can be a pain to get it to actually work sometimes. It is usually easier to get a new card that does have Novell 32 bit drivers.

Q: What does the "Send keyboard scancode" function do?

Some rare DOS programs (most notably EDIT.COM) use tricks to access the keyboard in non-standard ways. Because of this, these programs can not see keys that are stuffed into the keyboard buffer by programs like TINY.

Thanks to a an amazingly useful text on PC keyboards, I've added the ability for TINY to put keyboard scancodes directly into the keyboard controller chip. This technique is good enough to fool every program I've ever tried into thinking the keys were really pressed on the local keyboard.

So why not use this technique by default? Well, scancodes are messy. There are lots of different types of keyboards with different keys in different places making different scan codes. Normally the PC BIOS takes care of all this so you just push the "A" key and see an "A" pop up on the screen, but when faking keys into the controller you need to do a lot of this work yourself.

The basic idea is that every actual key on a keyboard is a little switch. When you push the key, the keyboard sends a scancode to the keyboard controller in the computer. When you let go of a key, the keyboard sends a different code. You can use the "Send keyboard scancode" to directly put these codes into the control as if they came from the keyboard itself and trick the computer into thinking you pushed (or let go of) any key on the keyboard in an order (Think Alt, scroll lock, even the power button).

To send a key, you need to know the "make" code (the code sent with the key is pressed) and the "break" code (the code sent when the key is released.

 Here is a table of all the make and break codes for a standard keyboard taken from the above website...
PC Keyboard Scan Codes (in hex)

Key	Down	Up	Key	Down	Up	Key	Down	Up	Key	Down	Up
Esc	1	81	[{	1A	9A	<	33	B3	center	4C	CC
1 !	2	82	] }	1B	9B	. >	34	B4	right	4D	CD
2 @	3	83	Enter	1C	9C	/ ?	35	B5	+	4E	CE
3 #	4	84	Ctrl	1D	9D	R shift	36	B6	end	4F	CF
4 $	5	85	A	1E	9E	* PrtSc	37	B7	down	50	D0
5 %	6	86	S	1F	9F	alt	38	B8	pgdn	51	D1
6 ^	7	87	D	20	A0	space	39	B9	ins	52	D2
7 &	8	88	F	21	A1	CAPS	3A	BA	del	53	D3
8 *	9	89	G	22	A2	F1	3B	BB	/	E0 35	B5
9 (	0A	8A	H	23	A3	F2	3C	BC	enter	E0 1C	9C
0)	0B	8B	J	24	A4	F3	3D	BD	F11	57	D7
- _	0C	8C	K	25	A5	F4	3E	BE	F12	58	D8
= +	0D	8D	L	26	A6	F5	3F	BF	ins	E0 52	D2
Bksp	0E	8E	; :	27	A7	F6	40	C0	del	E0 53	D3
Tab	0F	8F	' "	28	A8	F7	41	C1	home	E0 47	C7
Q	10	90	` ~	29	A9	F8	42	C2	end	E0 4F	CF
W	11	91	L shift	2A	AA	F9	43	C3	pgup	E0 49	C9
E	12	92	\ |	2B	AB	F10	44	C4	pgdn	E0 51	D1
R	13	93	Z	2C	AC	NUM	45	C5	left	E0 4B	CB
T	14	94	X	2D	AD	SCRL	46	C6	right	E0 4D	CD
Y	15	95	C	2E	AE	home	47	C7	up	E0 48	C8
U	16	96	V	2F	AF	up	48	C8	down	E0 50	D0
I	17	97	B	30	B0	pgup	49	C9	R alt	E0 38	B8
O	18	98	N	31	B1	-	4A	CA	R ctrl	E0 1D	9D
P	19	99	M	32	B2	left	4B	CB	Pause	E1 1D 45 E1 9D C5	-

For example, if you wanted to type the letter "J" on the host, you'd first look up the codes from the above table and then type "24 A4" into the "Send scancode" window on TINY. The "24" pushes the "J" key on the keyboard and the "A4" lets go.

So, let's say you started dos EDIT by mistake and now you can not exit because it is not recognizing your keypresses. You could use the codes "38 21 A1 B8 2D AD". This would push "Alt", then push and release "F", then let go of the "Alt", then push and release "X", thus quitting the program.

If you want to know the scancode for some fancy keyboard or a foreign keyboard, refer to this guide (http://www.win.tue.nl/~aeb/linux/kbd/scancodes-1.html) to every keyboard scancode imaginable.

Q: What is the difference between the normal and "extended" versions of TINY?

The only difference is that the "extended" version of TINY supports graphics modes on the host. This takes significantly more memory, so if you only need text modes you should probably use the standard version.

If you load the standard version of TINY and then switch into a graphics mode, you'll see a "mode not supported" message in the client.

The standard version of TINY uses about 30K, whereas the "extended" version needs about 38K.

The TINY program will print which version it is on startup.

Note that I have not included a graphics version that runs on the FTP stack inside the distribution. Let me know if you need this and I'll compile it for you.

Q: Why are the colors all messed up when I run 800x600x256 graphics mode programs?

By default, TINY will automatically send 800x600x256 graphics screens in compressed nibble format. This mode uses only half as much bandwidth as sending the full color data, but the costs is that you only get half as much color information. DOS graphics applications tend to only use the first 16 colors anyway, so typically this is ok.

If you are using an 800x600 application that really uses all the colors and you want to see them all, you can manually select this mode on the client by clicking on "Host Screen Mode" and selecting the non-compacted 800x600 mode. Note that this will be MUCH slower than compacted mode. Also note that when you quit out of the graphics application, you'll have to go back into the "Host Screen Mode" menu and re-select "Auto" to see the text screen again.

Q: What is are the "sliced text" modes for?

These modes divide up text screens into multipule packets so that no packet is bigger than 1400 bytes. So, for example, if you pick "Sliced Text 80x25" mode, it will take 5 packets (each with about 800 data bytes) to see a full screen update. In the normal text modes, TINY sends a full snapshot of the whole screen in one giant UDP packet.

Some reasons why you might want to send lots of little packets rather than one big one...

* There is a bug in your network card driver that doesn't let it send giant UDP packets (I'm talking to you, Realtek)

* Each little packet causes a shorter pause for the running forground program than one big one would

* There is some firewall or router between your TINY client and host that can't deal with giant packets

* You want to have smoother traffic patterns on your network

Becuase the packets are smaller, you can probably send them more quickly without causing more network problems. If you were using a delay of 250ms with large packets (/D250), you can probably drop that to 100ms if you are using a sliced text mode and get better update performance.

TINY will never automatically select a "sliced text" mode. If you want to use one, you will have to manually select it from the "host mode" menu or use the /M option on the command line.

Q: What exactly does the /I option on the TINY host do?

The /I option will cause the TINY host to generate an INT 09 (the keyboard hardware interrupt) after each key is pushed into the keyboard buffer.

There is an application from TeamPhone that seems to need this interrupt to tell it to wake up and read any pending keys.

If you have an application that doesn't seem to want to read the keys from TINY, you can try this option, but in general you shouldn't need it.

Q: Can I run TINY over a PPP connection?

Maybe, but it will be slow. Maybe try using one of the sliced text modes. If you do not have an internet connection to the host, maybe consider trying my WHOST program instead. WHOST runs great over slow serial links like dial-up modems.

Q: What is the license for TINY?

TINY is free for non-commercial use. If you really love TINY, you can express your gratitude though a tax-deductible donation to The Aasha Foundation.

If you would like to include TINY in a product you sell, please email me at the address below and we can work something very reasonable out.