telnetd.device

A telnet daemon for serial applications

by Christopher A. Wichura

Legal Information

`telnetd.device` is not in the public domain. All source files, along with the resulting executable, are Copyright © 1993 by Christopher A. Wichura. You may not sell `telnetd.device`.

Only the demo version of `telnetd.device` may be freely redistributed. Registered versions may not be redistributed. If someone is found redistributing a registered version of `telnetd.device`, their registration will be revoked and legal action to collect damages will ensue.

The demo version of `telnetd.device` may be freely redistributed via BBSs, InterNet/Usenet, and disk libraries such as Fred Fish's, as long as the archive is not modified. Disk magazines and services that charge extra for file transfers may not distribute `telnetd.device`.

In using `telnetd.device`, you accept the responsibility for any damage or loss of productivity/money that may occur through or during its use. Christopher A. Wichura is not and cannot be held accountable.

Registering `telnetd.device`

`telnetd.device` is shareware. The demo version of the device is restricted in several ways (see [Creating the configuration file], page 5). Registering `telnetd.device` will provide you with a personalized version of the device which does not have any of these restrictions. Supporting the author will also help to insure that improvements on the device are made.

The most recent information on how to register will always be found in `telnetd.device`, itself. To get this information, open the device with a regular terminal program, open a capture file, and then type `ATI1 RET`[1]. You can now close the capture file and print out the registration form.

---

[1] In registered versions of the device, the `ATI1` command displays who the device has been registered to.

Supplied Files

Included you should find the following files:

‘`telnetd_device.guide`’

‘`telnetd_device.guide.info`’

`AmigaGuide` format documentation for `telnetd.device`, including Workbench icon.

‘`telnetd_device.dvi`’

`DVI` format documentation for `telnetd.device`. Using a suitable `DVI` driver, you can print a hardcopy version of this documentation.

‘`telnetd.AS225-68000.device`’

`telnetd.device` compiled for use with CBM’s AS225 TCP/IP software. This requres AS225 Release 2, which, at the time of this writing, is only available to registered developers. This binary has been compiled to work with 68000 or higher CPUs.

‘`telnetd.AS225-68020.device`’

`telnetd.device` compiled for use with CBM’s AS225 TCP/IP software. This binary has been compiled to take advantage of 68020+ CPUs.

‘`telnetd.AmiTCP-68000.device`’

`telnetd.device` compiled for use with the freely redistributable AmiTCP TCP/IP software. This requires version 2 or higher of AmiTCP to operate. This binary has been compiled to work with 68000 or higher CPUs.

‘`telnetd.AmiTCP-68020.device`’

`telnetd.device` compiled for use with the freely redistributable AmiTCP TCP/IP software. This binary has been compiled to take advantage of 68020+ CPUs.

‘`ReleaseNotes`’

‘`ReleaseNotes.info`’

This ASCII file (and it’s associated Workbench icon) contains release notes regarding changes from one version of `telnetd.device` to the next.

System Requirements

In order to use `telnetd.device` on your system, you must have:

1. Workbench 2.04 or higher
2. One of the following TCP/IP protocol stacks:
   1. AS225 Release 2
      This is Commodore's TCP/IP protocol stack. At the time of this writing, Release 2 is only available to developers registered with Commodore. AS225 is the prefered stack to use with `telnetd.device` at this time, however.
   2. AmiTCP/IP Version 2 (or higher)
      This is a freely redistributable TCP/IP protocol stack for the Amiga.
3. Some form of network
   It would be rather pointless to install `telnetd.device` if one does not have a network connection. No specific hardware is required, though. `telnetd.device` will work equally well over a slip connection or ethernet, for example.

Theory of Operation

Incentive for `telnetd.device`

Networking is becoming more and more popular on the Amiga. Thus, people are now looking for ways to make their Amigas more accessable to others on a network. One common desire expressed many times on Usenet is for a way to allow people on the net to connect into their bulletin board system (BBS), for example. Unfortunately, most BBS software is written to talk only to modems and generally lacks support for network protocols. `telnetd.device` was written to allow people to connect into an Amiga running a serial-aware application from the net.

The `telnet` protocol

In the world of TCP/IP, there are two main protocols widely used to log into other machines on a network. These are `rlogin` and `telnet`. Most platforms with a TCP/IP stack support these protocols. `telnetd.device` was originally written to accept `telnet` connections. It now knows how to accept `rlogin` connections as well.

How `telnetd.device` works

`telnetd.device` is written to look like the standard Amiga `serial.device`. `serial.device` is the driver which serial applications use to actually talk to the serial hardware. It provides a high level abstraction from the hardware, simplifying the application's job of handling serial data. Because of this abstraction, it is possible to write `serial.device` look-a-likes that talk to something other than the Amiga's build in serial hardware. Expansion serial boards for the Amiga, such as Commodore's A2232 multi-port io board or GVP's ioExternder board, are examples of the use of `serial.device` clones to talk to expansion serial hardware.

By looking like `serial.device`, `telnetd.device` can be used by serial applications. However, looking like `serial.device` is not enough. Most BBS programs, for example, assume that they are talking to a modem, and look for certain messages generated by modems to indicate a call coming in, etc. Thus, `telnetd.device` also emulates a Hayes compatible modem to a limited degree.

In conclusion, by telling serial applications (such as BBSs) to use `telnetd.device` instead of `serial.device`, it is possible for the application to receive `telnet` connections.

Installation Procedures

Installing the device

There are four different versions of the `telnetd.device` load module. Two have been compiled for use with AS225r2, and are named '`telnetd.AS225-68000.device`', for use with any 68000 family CPU, and '`telnetd.AS225-68020.device`', which has been optimized for 68020 or higher CPUs. The other two, compiled for AmiTCP, are named '`telnetd.AmiTCP-68000.device`' and '`telnetd.AmiTCP-68020.device`'. When installing the proper device for your TCP/IP stack and CPU, you must rename the load module to '`telnetd.device`'.

Thus, to install the 68000 AS225 version, from a shell you would enter:

```
copy 'telnetd.AS225-68000.device' to 'devs:telnetd.device'
```

The equivalent command for the 68000 AmiTCP version would be:

```
copy 'telnetd.AmiTCP-68000.device' to 'devs:telnetd.device'
```

Creating the configuration file

Before using `telnetd.device`, you must create a configuration file that lets `telnetd.device` know what TCP/IP ports to service, etc. For the AS225 version of `telnetd.device`, this configuration file is called '`inet:db/telnetd_device.conf`'. For the AmiTCP version, the name is '`AmiTCP:db/telnetd_device.conf`'. In either case, the format of the configuration file is the same.

Each line of the configuration file describes one TCP/IP port that `telnetd.device` should monitor. Configuration entries have the template (a description of the fields in this template as well as sample configuration entries follows this section):

port units [`PACERATE` *bps*] *accesslist* [`LOGFILE` *filename*] [`ANSWERTIMEOUT` *seconds*] [`RLOGIND`] [`RLOGIND-8`] [`BLACKLIST` *list*]

Blank lines may be included in the configuration file. Everything that follows a '`;`' will be treated as a comment. The '`;`' can occur anywhere on the line.

> **Demo Version Note:** The demo version of `telnetd.device` does not look for or read the configuration file. It is hard coded to two units on port 5432 and a pace rate of 9600 baud. The demo version also only allows connections from your local network.

The TCP/IP port number

There are many different services available over a network. To make it possible for services to be used in a sane manner, the concept of ports was added to the TCP/IP protocol. Different services use different port numbers. The `telnet` protocol normally uses port 23. However, this is not a requirement. Thus, `telnetd.device` allows you to specify the port number that it should accept connections on. By watching multiple port numbers, it is possible to have different serial applications available on the same machine.

Specify a port number as an integer value. For example, use '`23`' if you want `telnetd.device` to accept connections on the normal `telnet` port. When using other ports, you should choose a port number greater than 5000 (the reasoning behind this is beyond the scope of this document, and has to do with the way TCP/IP port allocation was designed to be used).

> **Please note:** You may not specify more than one configuration entry with the same port number.

Units per TCP/IP port

Unlike a modem, which can only accept one connection at a time, TCP/IP ports can have many connections open on the same port number. This is what allows you to have multiple `telnet` connections to the same Unix machine, for example. BBS software[1], being written for modems, doesn't know how to handle this. To get around this, `telnetd.device` lets you specify the number of units to allocate for a particular port number. You can then run multiple copies of the BBS software, specifying different unit numbers of `telnetd.device` as the device to use.

---

[1] While this document refers to BBS software, the discussion applies to any serial application one wishes to run on top of `telnetd.device`

Like port numbers (see [TCP/IP Port Number], page 6), the number of units is specified as an integer value. If you wanted five units available, for example, '`5`' would be specified. These five units would then be available as units 0 through 4 when opening `telnetd.device`.

As indicated above, `telnetd.device` maps the first port's units starting with the device's unit 0. If more than one configuration entry is specified in the configuration file, subsequent TCP/IP ports will have their units mapped such that they follow the preceeding port's units. I.e., if your first port has 5 units allocated to it and the second port has 3 units, then the first port's units would be accessed as `telnetd.device` units 0 through 4 and the second port's units would be accessed as `telnetd.device` units 5 through 7. If a third port was specified, it's first unit would be accessed as `telnetd.device` unit 8.

Pacing outgoing data flow

The TCP/IP protocol will try to deliver data from one point to another as quickly as it can. Usually this is the desired behavior, and does not cause any problems, even over slow network links, because the protocol also implements a form of flow control. However, there may be cases where it is desirable to limit the flow of data. For example, many people reading large text files at breakneck speed could bog the system down. Since people generally can't read text scrolling by at incredible speeds, pacing the output may 1) make it easier for the reader to read data and 2) reduce system load.

`telnetd.device` provides a way to pace data output to the client on a per-port number basis (incoming data from the client is not paced). Thus, one might provide a few units on port 23 (the standard `telnet` port) with no pace rate, while making another port available that paces data at slower rates for those who want it.

To specify a pace rate, in your configuration entry you must include the keyword `PACERATE` followed by the baud rate your want data paced at. If no `PACERATE` keyword is specified then no pacing will occur. If the baud rate specified is 0 then pace will occur at the baud rate reported by the `telnet` client. Since not all clients support informing the daemon of the user's connect speed[2], `telnetd.device` also lets you manually specify a pace rate. Thus, entering a pace rate of `19200` would pace at 19200 baud, regardless of the speed reported by the client.

---

[2] When a client does not know how to tell `telnetd.device` what the connection speed is, `telnetd.device` uses 19200 by default.

Please note: `telnetd.device` does not pace between characters. Instead, within each second, `telnetd.device` will allow no more than the appropriate number of characters to be transmitted. Thus, pacing will appear as quick blasts of data at the beginning of each second, followed by dead time.

Logging incoming connections

The `LOGFILE` keyword allows one to specify a file to log connection information to. Because the connection information includes traffic statistics, it is only written out once a connection is terminated. Each connection adds one line to the log file. The format of this line is:

(*Date Time*) *port-number port-unit userid*`@`*host* (*host-ip*) *connect-time connect-date bytes-received bytes-transmitted*

The *userid* information is only available for `rlogin` connections; the `telnet` protocol does not provide it.

More than one configuration entry can point at the same logfile. The device will internally lock around writes to the log to prevent the logfile from being clobbered. If you have `OwnDevUnit.library` installed, it will also lock the logfile using that so that utilities such as the AmigaUUCP `trimfile` command will not be able to interfere with log updates, either. For this to work properly, however, you need at least version 2.1 of `OwnDevUnit.library`.

Controlling the amount of time a port can take to answer.

The `ANSWERTIMEOUT` keyword allows one to specify the maximum number of seconds that `telnetd.device` will wait for a unit to answer. If the unit does not answer within this time, it will try the next available unit. This is a useful way to deal with hung BBS ports, for example. Under previous versions of `telnetd.device`, the port supervisor (which is responsible for accepting connections and then handing them off to the first available unit) would wait indefinately for a hung port to answer. This effectively made it very difficult for additional people to log on since the supervisor always searches for the first free port in numerical order. Since it would always find the hung port, it would be difficult for people to connect to BBS ports on subsequent units; someone would have to be attempting to connect to the hung port at the time a second connection request came in so that the supervisor would think the hung port was in use and try the next unit.

In setting the timeout value with this keyword, be sure to leave enough time for a BBS port to reasonably be able to accept an incomming connection. Making it too long, however, can give users the impression that the system has hung, however.

Making ports use the `rlogin` protocol

`telnetd.device` normally expects incoming connections to use the `telnet` protocol. However, by specifying the `RLOGIND` option as part of a configuration entry, it is possible to make `telnetd.device` accept `rlogin` connections. The `rlogin` protocol is only truely 8-bit clean in one direction. However, but specifying the `RLOGIND-8` keyword instead of `RLOGIND`, you can force `telnetd.device` to ignore parts of the `rlogin` specification and treat both directions as 8-bit clean.

Restricting access

Security issues are becoming increasingly more important as more and more information is made available via networking. Because of this, many institutions may wish to limit access to certain networks or even specific hosts. There are also those who abuse their network privledges, thus creating the need for an ability to blacklist specific machines or even entire networks. `telnetd.device` supports limiting access on a per-port basis.

`telnetd.device` first verifies that a site belongs to one of the networks or machines allowed. If the machine network the client is connecting from is ok, `telnetd.device` then check the blacklist (if specified) to see if the particular machine (or even the entire network it belongs to) should be rejected.

The Access List

At least one access entry must be specified in a configuration entry. Access entries are specified in IP form only. If you wish to allow any site to connect, specify an access of '`255.255.255.255`'. To limit access to a specific network, specify the network's IP. For example, if your network is 128.135, you would specify '`128.135`'. At this time, subnets are not supported, and attempting to specify one will cause access attempts to fail (except from the machine with address 0 within the subnet you tried to specify). If you wish to allow access from specific machines, then specify the full IP address for the machine(s) desired.

Blacklisting specific machines or networks

Blacklist entries are not required. If you wish to specify a blacklist entry, however, you must include the `BLACKLIST` keyword in the configuration entry.

> **Caution:** The *only* thing that may follow the `BLACKLIST` keyword in the configuration entry is actual blacklist entries. Anything else will cause a parse error.

Blacklist entries are specified in the same format that access entries (see [Access List], page 9) are specified.

Sample configuration entries

The most simple configuration entry might look something like '`23 5 255.255.255.255`'. This would allocate five units for use on port number 23. Since 255.255.255.255 is specified as the access list, connections would be accepted from any machine on any network.

Working with the above example, suppose someone was harassing you, connecting from a machine with an IP address of 200.199.198.1. To prevent this person from harassing you, you could add a blacklist entry for this machine, making the configuration entry become '`23 5 255.255.255.255 blacklist 200.199.198.1`'. If the person harassing you has access on other machines on the net his machine belongs to, and starts connecting in from a different machine, you could blacklist the entire net by changing the configuration entry to be '`23 5 255.255.255.255 blacklist 200.199.198`'.

Now suppose you want to make a second serial application available on the net. For whatever reason, you don't want the general public to be able to connect to this application. You could add a second configuration entry that would open units on a second port number, with access restricted to your local net (198.147.221 will be used as the local net in this example). Your configuration file might be changed to look something like this:

```
23 5 255.255.255.255 ; allow anyone to connect on the normal telnet port
6543 2 198.147.221
```

A configuration file such as this would allow anyone to connect on the normal `telnet` port, for which there are five units available. These units are accessed as units 0 through 4 of

`telnetd.device`. People whose machines are on the net 198.147.221 would also be able to connect to port 6543, which only has two units available. These two units would be accessed as units 5 and 6 of `telnetd.device`.

Testing the Installation

Once you have installed `telnetd.device` and created a configuration file, you will probably want to make sure that the device is working properly. This is most easily done by opening the device with a terminal program. If all goes well, you should be able to enter Hayes style AT commands and have the device respond back as if it was a modem. For example, `AT$ RET` can be used to get a list of the AT commands that `telnetd.device` understands. If you are able to enter AT commands properly, you should now try `telnet`ing into the port number you specified in the configuration file. You should see `RING` messages in your terminal window. Entering `ATA RET` should cause `telnetd.device` to 'answer' the connection. You can then hang the connection up by dropping DTR.

If there is a problem with the configuration file, an error requester will appear telling you which line has a syntax error. Other error requesters may occur, but are much less likely. For example, if `telnetd.device` complains that it "Couldn't bind name to initial socket", then there is probably already something else (i.e., some other daemon) listening on the port number you specified.

Once you have determined that `telnetd.device` is properly installed, you can set up your BBS (or whatever) software. Since `telnetd.device` does not emulate the NVRAM most modern modems have, you must specify all needed parameters in an init string that your software will send out each time it opens the unit. For example, `telnetd.device` defaults to not 'auto-answering' a connection. If your software expects the modem to do 'auto-answer', you will have to specify an init string such as `ATS0=1`.

Implementation Details

> **Please note:** This section contains technical information about how various aspects of `telnetd.device` were implemented. Information in this section is not needed to install or use `telnetd.device`. This section is provided mainly for those who may be writing a serial application and wish to know how `telnetd.device` differs from `serial.device` at the device level.

The Hayes AT Commands supported

`telnetd.device` supports a relatively small number of the standard AT commands defined by Hayes. A list of the AT commands `telnetd.device` knows is available by entering `AT$ RET` from a terminal program. Most commands supported are directly related to the needs of an application that wishes to act in answer mode only. For example, S Register 0 is used to determine if the device should 'auto-answer' the connection or not. Commands not recognized by the device, or which are syntactically incorrect, will generate an `ERROR` message.

The device supports a form of "caller id". The information returned includes the machine the connection was initiated from and the time of the connection. The IP address of the machine is always supplied. If the hostname can be determined, it will also be returned. Otherwise, the hostname will be reported as "???". Two forms of the "caller id" output are available. The verbose form generates output suitable for humans to read. The terse form is intended to be machine readable, and is output as:

seconds minute hour day month year day-of-week ip-address host-name

> **Caution:** The "caller id" information does not supply you with the login account of the client. Since it is possible for many different accounts on the same machine to telnet in, using the "caller id" information to do any form of automatic login would be a huge security hole. Be careful when deciding what you wish to use the "caller id" information for.

`telnetd.device` does not emulate the NVRAM most moderm modems have. Thus, any options you require must be specified in an init string whenever a unit is opened.

`telnetd.device` does *not* support the "`+++`" escape sequence for returning to command mode. Thus, in order to drop a connection, one must do a DTR drop. This can be accomplished by closing and re-opening the device, or by using the ASDG extension IO Request `SIOCMD_SETCTRLLINES`.

Processes Used

When `telnetd.device` is opened for the first time, it reads in the configuration file and starts a supervisor process for each TCP/IP port that is to be monitored. Each unit that is opened also has a process associated with it. When a connection comes in on a port, the supervisor for that port first checks for any access violations. If the site is note allowed for some reason, an access restriction message will be sent to the client and the connection will be closed. Otherwise, it will look for a unit which is 1) opened and 2) not servicing a connection. If it doesn't find a free unit, it sends a message to the client indicating that there are no free units and closes the connection. If there is a free unit, the supervisor hands the connection off to the unit's process. At this time, the supervisor no longer touches the connection, except in the case that the unit experiences a fatal problem trying to initialize the connection. If this happens, the supervisor will send a message to the client that a fatal error has occurred and the connection will be closed.

When the unit process receives a connection from its supervisor, it first trys to negotiate several `telnet` options with the client. These include asking the client to operate in character-at-a-time mode (rather than line-by-line mode), asking the client what the actual connection speed is, etc. Once these `telnet` options have been negotiated, the unit will start generating `RING` messages. At this time, the client will receive a message that the device is waiting for the unit to answer. Depending on the setting of S Register 0, `telnetd.device` will either wait for the application to issue an `ATA` command or will 'auto-answer' the connection. Once the unit has answered the ring (either by `ATA` or by 'auto-answer'), the device will respond with a `CONNECT` *baud* message. The client will also receive a message indicating which unit the connection has been answered by. The baud rate reported to the application in the `CONNECT` message is the speed reported by the client. If the client does not support the `TELOPT_TSPEED` negotation (and many don't), the device reports the speed as 19200 baud.

IO Requests Supported

`telnetd.device` understands all the standard `serial.device` IO Requests. It also knows about ASDG's `SIOCMD_SETCTRLLINES` extension that allows the DTR line to be controlled without having to close the device and re-open it.

SDCMD_SETPARAMS is treated mainly as a no-op. All settings are ignored. This includes baud rate, SERF_XDISABLED, etc. SERF_EOFMODE *is* honored, however. The buffer size can not be changed; it is fixed at 16k.

SDCMD_SENDBREAK sends a telnet BREAK sequence over the wire. Some clients may panic if they receive this signal. telnetd.device, when it sees a telnet BREAK, AO, or IP will flag that a break has been received. If there is a read request pending, it will be returned with *io_Error* set to SerErr_DetectedBreak. Otherwise, the IO_STATF_READBREAK bit in *io_Status* will be set TRUE when a SDCMD_QUERY is performed.

CMD_RESET will not cause the current connection to be dropped.

Credits

Brian Vargyas for getting me started on writing `telnetd.device` and for providing the extra machine used in testing, as well as beta testing the device with the C-Net BBS software.

William Coldwell for beta testing the device with the BBX BBS software and for providing a few comments on the operation of the device itself.

Contacting the Author

If you happen to find a bug, want to register, or have a suggestion for `telnetd.device`, or even just want to say "hey, cool program", please contact me using one of the ways listed below (registrations are limited to US Mail, obviously, since you can't e-mail money). Even if you want to say "`telnetd.device` sucks", let me know and be sure to say why you feel this way so that I might be able to fix what you think is wrong with the program.

These electronic forms are the most prefered means of contacting me. They will get you a response pretty quick.

```
e-mail: caw@miroc.chi.il.us
BIX: caw
```

My US Mail address is:

Christopher A. Wichura
5450 East View Park
Chicago, Il. 60615
USA

You can also reach me by phone. However, please try to limit your calling to evening hours (I'm in the central time zone). If I'm not home and you leave a message, call back again anyway. Around here, one tends to get maybe 5% of the messages left for them, if lucky...

(312)/684-2941

Concept Index

Table of Contents