

# OS/400 Pseudo-TTY driver and ASCII telnet server for PASE applications

(C) Copyright 2000 IBM Corporation.

This sample has not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of this program. All programs contained herein are provided to you "AS IS". THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED.

## Contents

[Overview](#)

[Installation](#)

[Starting ATELNETD Server](#)

[PTY Modes](#)

[PTY Special APIs](#)

## Overview

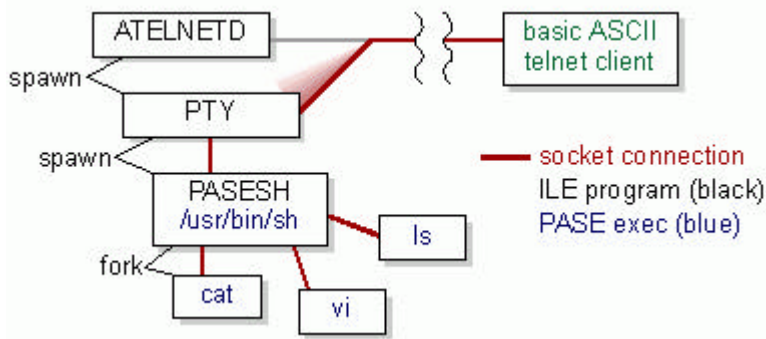
This page describes the tools required for running ASCII tty-style OS/400 programs, especially those running in PASE. It also describes how to set up the AIX/PASE development environment such that applications can control the Pseudo-TTY (PTY) driver.

The first, ATELNETD, is an ASCII telnet daemon. It accepts connections from a simple VT100/ANSI telnet client, such as the telnet.exe program included with Microsoft Windows. This program takes one parameter, the port number on which to listen. This number must be a text string, as shown in the SBMJOB command below. If not specified, the program will default to port 6042. One login attempt is permitted per connection. Authentication is through normal OS/400 facilities. Only enabled OS/400 user profiles will be allowed login access. Once logged in, the user's PTY and shell processes all run authorized as the user's user profile. Login access can be restricted by securing the PASESVE/QPTY program object with normal OS/400 access controls.

The second program, QPTY, is a pseudo-TTY driver program, which processes keystroke input and shell/program output. It has two modes, a RAW mode which does no processing of data between the shell and terminal, and a COOKED mode, which, among other things, adds carriage-returns to line-feeds from the shell and removes carriage-returns from the input stream, since the shell does not tolerate them. COOKED mode also handles backspace processing. These two general modes are set up using the special control flags as described below.

The third part is a simple ILE program which calls the PASE shell. Before it calls the shell, using Qp2RunPase(), it sets up signals to cascade from ILE into PASE, which are otherwise only delivered to ILE.

Below is a diagram of the server's architecture.



The last components in the list are related to the development environment and are discussed in detail later in this document.

## Installation

The package contains the following objects.

Object Type	Object Name	OS/400 Library Name
OS/400 *PGM	ATELNETD	PASESVR
OS/400 *PGM	QPTY	PASESVR
OS/400 *PGM	PASESH	PASESVR
C header file	sys/ioctl.h	
C header file	termios.h	
C header file	unistd.h	
PASE API library	libqpty.a	
PASE utility	spty	
this HTML file	readme.html	

The package is distributed as a compressed archive. Inside the archive are the stream file objects listed above, and a save file containing a SAVLIB of the OS/400 program objects.

The save file must be transferred into a save file object on OS/400. Use FTP from either a PC or UNIX box where the archive was expanded, and use the following FTP commands.

```

quote rcmd crtsavf qgpl/pasesvr
bin
put pasesvr.savf qgpl/pasesvr
quit
  
```

Then, issue the following restore command on an OS/400 command line.

```
RSTLIB SAVLIB(PASESVR) DEV(*SAVF) SAVF(QGPL/PASESVR)
```

## Starting the Server

There are two options that can be specified to start the server. The first option defines on which port number the server will listen. The second parameter defines the name of the PASE executable to run as a user login.

The options must be specified in the above order. The PASE executable cannot be specified if a port number is not specified.

The server will default to port number 6042 and /QOpenSys/usr/bin/ksh, respectively, if no options are specified.

```
SBMJOB CMD(CALL PASESVR/ATELNETD '99') JOB(ATELNETD) JOBQ(QUSRNOMAX)
```

In the above example, the server is started on port 99. The default shell executable is selected. Use a simple ASCII telnet client to connect to the server on port 99.

```
SBMJOB CMD(CALL PASESVR/ATELNETD ('200' '/usr/bin/csh') +  
JOB(ATELNETD) JOBQ(QUSRNOMAX)
```

In the second example, the server is started on port 200, and a different shell is specified for the user to run at login. Use a simple ASCII telnet client to connect to the server on port 200.

## PTY Modes

The PTY driver operates in two modes. In **RAW** mode, every keystroke is sent to the stdin of the child shell process. This is needed for programs like vi, which expect to deal with each keystroke. The other mode is **COOKED** or **CANONICAL** mode. In COOKED mode, backspace key(s) are processed, and input is not sent to child until a CR is sent (enter key pressed). A canonical mode is preferable for programs like sh.

Below are the POSIX standard mode flags currently recognized by the PTY driver. If your application requires responding to other control flags, please contact us at the email address at the end of this document.

Input Control	Output Control	Local Control
BRKINT	OPOST	ECHO ECHOE ECHOK ECHONL ICANON ISIG

A utility, spty, is included, to switch the mode flags of the PTY driver. This has been compiled on AIX for PASE and may be used as any other PASE shell command. The spty utility can be used to switch the above flags on or off individually, as below.

```
spty ECHO
spty -ECHONL -ECHOE
spty ICANON -ECHO
```

For applications which are not able to be rebuilt using the [PTY special APIs](#), this command can be used in sequence to enable applications which require raw mode. In most cases, the application will perform the mode switch using the APIs described below. Without recompilation these APIs will not communicate with the PTY driver. For example, we may desire to run vi in raw mode and then switch back to cooked mode.

```
spty -ECHO -ICANON; vi whatever.c; spty ICANON ECHO ;
```

The spty utility uses the APIs mentioned below. These APIs rely on the QPTYPID environment variable to be set to the process ID of the PTY driver. The environment variable QPTYPID is described further in the [PTY special APIs](#) section.

## PTY Special APIs

OS/400 SLIC does not provide the terminal control attributes for file descriptors. So, the PTY driver allocates some storage for the terminal control flags for descriptors 0,1,2. These flags are for controlling operational modes, signalling rules, and echo behavior.

The APIs in the following table need to be remapped to special versions of the same APIs which work with the PTY control data. These APIs rely on an environment variable, QPTYPID, which is set to the process ID number of the PTY process. This environment variable is automatically set when the initial shell process is spawned.

tcsetattr	tcgetattr	ioctl	isatty
-----------	-----------	-------	--------

Of the above APIs, only isatty() operates differently than expected. Because the operating system does not maintain whether a file descriptor is actually a TTY device, this API responds to the environment variable QPTY\_ISATTY. For descriptors 0,1, and 2, if this variable is set to "Y", "YES", "Yada Yada Yada" or anything else that starts with a capital "Y", isatty() will return 1. Otherwise, it will return 0.

There are essentially two changes needed to the development environment to use the special versions of these APIs. No change to any C source code is required. First, an include directory must be created for the OS/400 PTY-specific header files. These are the three header files included in this package. Ensure that the new include directory is the first directory searched for include files. This can be accomplished by using the -I option on the compile of your PASE application.

```
/qpty/include/termios.h
```

```
/qpty/include/unistd.h
```

```
/qpty/include/sys/ioctl.h
```

Second, the program must be linked to the library which contains the special versions of these APIs. This library is named **libqpty.a**.

Using this example, you would specify the following as a compile command. Of course, this would normally be incorporated into a make environment.

```
cc -I/pty/include whatever.c libqpty.a
```

## **Notice to users of OS/400 earlier than V4R5**

This was developed and tested on OS/400 V4R5. Options available in OS/400 V4R5 make using the server much simpler. No testing or support will be provided for use on any release of OS/400 prior to V4R5. In exception, if there is a justifiable need for this to function on OS/400 V4R4, we might make such a version available. Please contact us, explaining your situation, if you have such a need.

Enjoy.

IBM PartnerWorld

Questions can be directed to [rchgo400@us.ibm.com](mailto:rchgo400@us.ibm.com).