TELNET/3000-V

External Reference Specifications

HEWLETT PACKARD

Information Networks Division

Location Code: 66-7850

Project Number: 6651-3035

September 28, 1987

Jeff Orum

Version: A.00.04

PRINTING HISTORY

Version A.00.00 - Original Version.

Version A.00.01 -The names of the procedures the Telnet Monitor calls were changed to eliminate the "_" character. Changed interface specification for PDRecvMsg and added a description of IOWAIT. Changed values for the return codes and special function status returns. Updated the configuration section. Added a maximum size and more details on QStat in the pseudo IOQ specification.

Version A.00.02 - A note was added regarding the cstation parameter in IOWait. Return codes %407 and %413 were added. Function 5, set read timeout, was eliminated.

Version A.00.03 - Added section on FTP logon and logoff. A note was added for the write pseudo IOQ when switching from pre to post spacing. Added the fact that break should be enabled on certain set logon pseudo IOQs.

Version A.00.04 - Added information on the logon parameter for FTPLOGON. New result codes were added to FTPLOGON and FTPLOGOFF.

Version A.00.05 - Updated configuration information. Removed restrictions on aborting the user sesion for FTPLOGOFF.

CONTENTS

CONTENTS (continued)

PRODUCT IDENTIFICATION

SECTION 1

1.1 PRODUCT NAME

Telnet/3000-V

1.2 PRODUCT ABSTRACT

Telnet/3000-V is a joint project between IND and The Wollongong Group to provide the Defense Data Network's Telnet virtual terminal service. HP's involvement in Telnet is limited to producing an interface into the I/O system for Wollongong. This document will describe that interface.

1.3 PROJECT PERSONNEL

Jeff Orum
Ollie Polk
Katy Jenkins (Project Manager)

PRODUCT OVERVIEW

SECTION 2

2.1 PRODUCT GOALS

Hewlett-Packard had a number of goals in mind when this product was designed:

- Allow Wollongong to write their code as a user program. Any privileged mode code should be supplied by HP.
- Isolate the 3000 I/O system from Wollongong. This will make their code portable. Also, this guarantees that the interface between HP code and Wollongong code will remain consistent. Should the I/O system interface change, only the HP code will need to be modified.
- Support running VPLUS applications on terminals that are connected to an HP 9000.
- By providing this interface, other code (as yet unknown) could use the HP code in a similar manner to Wollongong's. In effect, HP is providing a generic driver.

2.2 INTERFACE OVERVIEW

A block picture of this product is as follows:

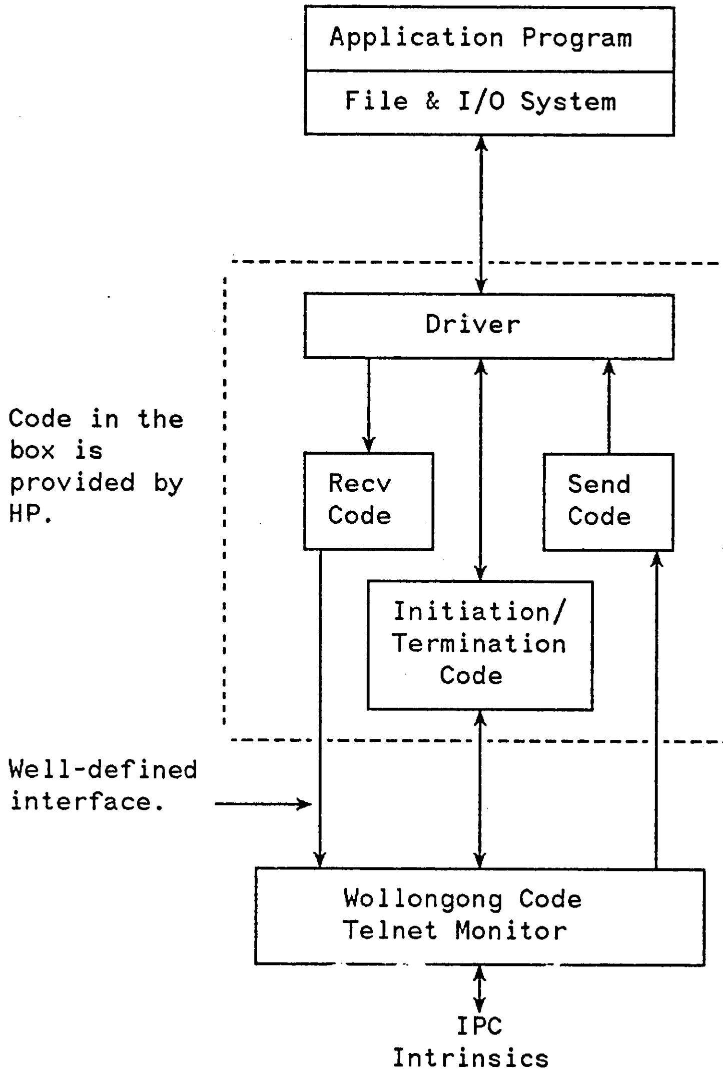

Figure 2.1 Telnet Code Structure

2.3 MODULE DESCRIPTION

2.3.1 Application Program and File System

The application program uses standard MPE Intrinsics to direct I/O to the terminal. Because it uses this interface, it does not know if the terminal is directly connected to the system or is coming in from a remote machine. Note that the application program can be the command interpreter.

The file and I/O system accepts requests from the application program. It will then build a data structure (called an IOQ) that specifies what type of operation to perform (e.g. read, write or control).

2.3.2 Driver

In MPE, a driver accepts IOQ's from the file system and then actually performs the I/O. Typically, drivers have real hardware associated with them (e.g. a terminal or printer). When a driver receives an IOQ, it will issue the necessary commands to the hardware to perform the operation.

The Telnet driver has no real hardware associated with it and so it is referred to as a *pseudo* driver. Since no hardware exists for this driver, the device it controls is called a *pseudo* terminal. The purpose of this driver is to accept IOQ's and translate them into a data structure, a *pseudo IOQ*, that will be sent to the Wollongong code. This data structure will be documented in Section 4. Note that if the structure of the IOQ (or any other system table) changes, only the driver needs to be modified and it will still output the same data structure to the Wollongong code. Thus the interface between the HP code and Wollongong code will remain consistent.

2.3.3 Initiation, Termination, Send and Receive

These modules are provided to allow the Wollongong code to communicate with the driver. They are necessary because privileged mode is required to do the communication. The interface for these modules is described in Section 3.

2.3.4 Wollongong Code

This code is also called the Telnet monitor. Its purpose is to build the appropriate Telnet messages and send them out over the network. I assumed in the diagram that IPC would be used, but this could change and it would not affect the HP supplied code. The monitor code would be responsible for implementing any changes in the Telnet protocol.

The monitor will need to have a receive posted on the network and the driver simultaneously. Due to the asynchronous nature of terminals, it is not known where data to the monitor will come from next. For example, the monitor could be receiving write requests from the driver and then a break request would come in over the network.

In order to do this, the monitor will operate in *nowait* mode. The monitor will post a receive on the driver which will complete immediately with no data and then post a receive on the network which will also complete immediately with no data. The monitor will then call the intrinsic IOWAIT which will impede it. When either of the receives completes, the monitor will be awoken. It can then process the data that has arrived.

2.4 FUNCTIONAL DESCRIPTION

2.4.1 Session Startup

The following description of how the monitor gets started is only a suggestion. It will be up to Wollongong to decide exactly how to start a monitor.

There will need to be a process that listens to a well-known address for a connection request. Perhaps this process could be in a job that is streamed whenever the system comes up. When a request is received from a remote machine, this process will create a Telnet monitor process.

The Telnet monitor will send an initialization request to the pseudo driver. The driver will obtain a pseudo terminal and inform MPE to start the logon procedure. MPE will, through the driver, issue a write and then a read. At this point, we are operating in steady state mode.

2.4.2 Steady State

In steady state mode, the Telnet monitor will receive I/O requests from the pseudo terminal driver. It will either process them locally or send some sort of Telnet message to the remote machine. Once the monitor has finished processing a request it will send the results to the driver.

An exception to this driver initiated form of operation is if an interrupt process request comes from the remote side. In this case, the monitor will need to notify the driver that this condition exists. The driver will notify MPE and then process the requests that MPE sends down to it.

2.4.3 Termination

A termination request can come from either the driver or the network. In either case, the monitor should call the termination code that is supplied by HP. This will terminate the session, free the pseudo terminal, and break the connection from the driver to the monitor. The monitor should not do any more sends or receives from the driver at this point. Once the monitor has finished its termination code, it can also terminate.

2.5 CONFIGURATION

Telnet pseudo terminals can be configured with SYSDUMP or INITIAL. The following dialogue is used in both:

```
ANY CHANGES? y
.
.
I/O CONFIGURATION CHANGES? y
.
.
<< Note, for each pseudo terminal you want, do the following: >>
LOGICAL DEVICE #? N, where N is the LDEV of the new pseudo terminal
DEVICE NAME? <<carriage return>>
DRT #? Input the logical device number of the console preceded by
       a #.  For example, if the console is ldev 20, input #20.
UNIT #? 0
SOFTWARE CHANNEL #? 0
TYPE? 16
SUB TYPE? 0
ENTER [TERM TYPE #],[DESCRIPTOR FILENAME]  ? <<carriage return>>
SPEED IN CHARACTERS PER SECOND? <<carriage return>>
RECORD WIDTH? 40
OUTPUT DEVICE? N, same as for the logical device question.
ACCEPT JOB/SESSIONS? y
ACCEPT DATA? n
INTERACTIVE? y
DUPLICATIVE? y
INITIALLY SPOOLED? <<carriage return>>
AUTO REPLY? <<carriage return>>
DRIVER NAME? iopterm0
DEVICE CLASSES?  input any name here you want
```

INTERFACE SPECIFICATIONS

This section will describe the procedures the Telnet monitor uses to communicate with the pseudo driver. The interface specifications will be given in Pascal.

3.1 INITIALIZE PSEUDO DRIVER

```
Procedure PDInit        ( var ldev        : small_int,
                          var return_code : small_int    );

Function:
  Initializes the connection to the pseudo driver.  The pseudo
  driver will allocate a pseudo device and start the logon
  process.  This procedure must be called by the monitor
  before any attempt is made to communicate with the driver.

Input Parameters:
  None.

Output Parameters:
  ldev          - the logical device number of the pseudo
                  terminal.  Required for further communication
                  with the driver.
  return_code   - See codes in section 3.5.
```

3.2 SEND MESSAGE TO DRIVER

```
Procedure PDSendMsg       (     ldev           : small_int,
                                msg            : pIOQ_ptr,
                            var return_code    : small_int );

Function:
  Sends the pseudo driver a message.  If this procedure returns
  successful, the driver has received the message.

Input Parameters:
  ldev       - the logical device number of the pseudo terminal.
  pIOQ_ptr   - a pointer to the pseudo IOQ message.  See section
               4 for a description of this data structure.

Output Parameters:
  return_code - See codes in section 3.5.
```

3.3 RECEIVE MESSAGE FROM DRIVER

```
Procedure PDRecvMsg        (    ldev          : small_int,
                            var number        : small_int,
                            var return_code   : small_int  );

Function:
  Initiates a receive from the driver.  The receive will be done
  nowait, so control will be immediately returned with no data.
  An IOWAIT must be done in order to receive the data.

Input Parameters:
  ldev        - the logical device number of the pseudo terminal.

Output Parameters:
  number      - contains the completion number IOWAIT will return
                when a pseudo IOQ is received.
  return_code - See codes in section 3.5.  Note that a
                successful completion only indicates that the
                receive has been posted.  When IOWAIT completes,
                if the pseudo IOQ is a special function reply,
                QStat should be examined to get the status of
                the request.
```

The IOWAIT that is done by the Telnet Monitor should be done as follows:

```
  completion_num := IOWAIT ( , buffer, count )
```

Because the file number is not specified, this IOWAIT will complete when any I/O completes. If a pseudo IOQ caused the completion, the **completion_num** parameter will equal the **number** parameter that is returned by PDRecvMsg. The **buffer** parameter should point to the Telnet Monitor's receive buffer for pseudo IOQs. The buffer must be large enough to accommodate the pseudo IOQ or an error will be returned. The **count** parameter will be set to the pseudo IOQ length in words. The **cstation** parameter should not be specified.

3.4 TERMINATE PSEUDO DRIVER

```
Procedure PDTerm       (    ldev        : small_int,
                        var return_code : small_int );

Function:
  Terminates the connection to the pseudo driver.  The pseudo
  driver will release the pseudo terminal and terminate the
  session (if there is one) logged onto the terminal.  Once
  this procedure is called, no further communication with the
  driver is possible.

Input Parameters:
  ldev - the logical device number of the pseudo terminal.

Output Parameters:
  return_code - See codes in section 3.5.
```

3.5 RETURN CODES

The following are the return codes these routines will return.

Code %	Meaning
1	Successful
402	Invalid ldev passed
403	Unsupported function, QType not 1 or 2
405	Can't receive message from driver
406	Can't send message to driver
407	Can't find request with specified QNum
411	No pseudo terminals available
412	Can't set up communication with driver
413	Incompatible version of the driver

DATA STRUCTURE SPECIFICATIONS

SECTION 4

This section will describe the data structures the Telnet monitor and driver use to communicate with each other. The main data structure is a pseudo IOQ, and is shown below:

Word		
0	Function Type	QType
1	Function	QFunc
2	Request Number	QNum
3	Data Offset	QAddr
4	Parameter 1	QParm1
5	Parameter 2	QParm2
6	Flags	QFlags
7	Count and Control Returns	QCount
8	Status	QStat
9 on up	Data	

Figure 4.1 Pseudo IOQ

QType This field specifies the type of function. The values allowed are:

```
0 - Normal I/O Function Request.
1 - Normal I/O Function Reply.
2 - Special Function Request.
3 - Special Function Reply.
```

QFunc	This field specifies the function. Values for this field will be described in the next sections.
QNum	This field is used to synchronize the driver and monitor. Each pseudo IOQ will have a request number and when a completed pseudo IOQ is returned to the sender it will contain the same number. The request number is generated by the sender; the receiver should not expect the number to be of any relevance or in any particular sequence.
QAddr	This field points to the start of the data in the pseudo IOQ. It currently will have the value 9. By using this field, if the pseudo IOQ expands, the code that looks at the data will not have to be changed.
QParm1 QParm2	These contain additional information for a given function. See the individual functions for a description of what data these fields contain.
QFlags	This field contains flags which may modify the function.

Bit Position	Meaning
(0:1)	Abort
(12:2)	Preemptive: 1-soft, 2-hard

The Abort bit is used to abort a previous request. QNum is set to the same value as the QNum in the request to be aborted. The receiver of this request should return the request with an aborted status.

The Preemptive bits are used to indicate a request is of a high priority. A soft preempt means to finish doing the current I/O and then process this request next. A hard preempt means to stop the current I/O and process this request.

QCount	On initiation, this field contains the positive byte count. At completion of the request, this field contains the actual transmission count. Certain control requests return data through this location.
QStat	This field contains the result status of the request. See section 4.3 for a description of the codes. It is set only in replies. In requests it is 0.

The maximum size of a pseudo IOQ is 15009 words. This allows for a 9 word header plus 15000 words of data. This should be sufficient for most applications. VIEW, for example, can do a maximum read or write of 12000 words.

Note that in general, only one pseudo IOQ will be sent at a time from the driver to the Telnet Monitor. No new pseudo IOQs will be sent until a reply is received on the outstanding pseudo IOQ. An exception to this will be if an existing pseudo IOQ is to be aborted. Also, a preemptive request may be sent while there is still an outstanding pseudo IOQ. The QFlags word is used to indicate these situations.

4.1 NORMAL I/O FUNCTIONS

This section will describe what the functions are when QType = 0 or 1. With these functions, the request is always from the pseudo driver to the monitor and the reply from the monitor to the pseudo driver.

4.1.0 Function 0 - Read

This request is to read data from the terminal. In MPE, there are three types of reads:

Edited Mode - In this mode, the read is terminated by a carriage return or the byte count being reached. If an alternate end of record (AEOR) character is passed [QParm2(0:8)], this character will also terminate the read.

Unedited Mode - This is also called Transparent Editing Mode. In this mode, which is enabled via Function 37, the read is terminated by the end of record character (EOR) passed in the function, the AEOR (if one is specified), or the byte count being reached. If the input terminates with the AEOR or EOR character, carriage return/line feed is not sent to the terminal. If the byte count terminated the read, the carriage return/line feed will be sent depending on QParm1(0:1). Note that unedited mode remains in effect until it is disabled.

Binary Mode - In this mode, which is enabled by QParm2(11:2), the read is terminated only by the byte count being reached. Binary mode is enabled by FControl 27 and disabled by FControl 26. (This will also affect writes.) This mode will disable unedited mode if it is active. Carriage return/line feed is not written to the terminal when the read completes. Note that binary mode is enabled in each read.

QParm1

(0:1)	If 0, issue a carriage return/line feed when the read completes. If 1, do not. This is enabled/disabled by FSetMode.

QParm2

(0:8)	Contains the AEOR character that will terminate the read. If 0, reset the AEOR. If a read completes with the AEOR character, the AEOR character should be returned in the user data, and no carriage return/line feed is issued. Also, on return, QStat should be %11. The AEOR is set by FControl 25.

QParm2
(cont'd)

(9:1) If 1, indicates a VIEW read. The VIEW read is enabled by FControl 31 and disabled by FControl 30. Neither of these are documented, they are only for use by VPLUS. The terminal must be in block mode, the D strap must be turned on (PAGE), the H strap must be turn off, the G strap must be turned on (meaning the terminal uses a DC1/DC2/DC1 handshake), echo must be off, the driver must be in unedited mode, and subsystem break must be disabled. VPLUS will send the necessary escape sequences and FControls for this. The handshake is as follows:

```
Computer                       Terminal
 Application program
   issues a read.
 Sends DC1 --->
                                User inputs data,
                                     presses ENTER.
                           <--- Sends DC2

 The driver issues
 the escape sequences
 to lock the keyboard
 and home the cursor. --->

 Sends DC1 --->
                           <--- Sends data then record
                                  separator character.
```

If the terminal uses XON/XOFF flow control, a different handshake is used for VIEW reads. See section 6.1.1 for a description of this.

QParm2
(cont'd)

(10:1) If 1, indicates an OWN read (also known as User Block Mode Handshake). This is enabled by an FControl 29 and disabled by an FControl 28. If the OWN read is disabled, the following is done (The terminal should already be in block mode, and use a DC1/DC2/DC1 handshake.):

```
Computer                              Terminal
  Application program
    issues a read.
  Sends DC1 --->
                                        User inputs data,
                                             presses [ENTER].
                                  <--- Sends DC2
  Sends DC1 --->
                                  <--- Sends data then record
                                       separator character.
```

If the OWN read is enabled, the following is done (The terminal should already be in block mode, and use a DC1/DC2/DC1 handshake.):

```
Computer                              Terminal
  Application program
    issues a small read.
  Sends DC1 --->
                                        User inputs data,
                                             presses [ENTER].
                                  <--- Sends DC2
  Application program gets
    the DC2.  The program will
    take some action, such as
    moving the cursor.  It
    will then issue a read.
  Sends DC1 --->
                                  <--- Sends data then record
                                         separator character.
```

(11:2) If non-zero, indicates a binary read.

QCount The request will contain the maximum number of bytes to read. The reply will have the actual number of bytes read.

Data The read data will be returned here.

4.1.1 Function 1 - Write

This request is to write data to the terminal. A zero length write means to just output the carriage control.

QParm1

(8:8)	Contains the carriage control character. If this is a 1, the carriage control character is actually the first byte of the data. For a list of carriage control characters, see the *MPE V Intrinsics Reference Manual (32033-90007).*

QParm2

(11:2)	If set to 0, write data is 7 bit ASCII. Otherwise, data is 8 bit ASCII, sent in Binary mode. Binary mode is enabled by FControl 27 and disabled by FControl 26. (This will also affect reads.) In Binary mode, carriage control is disabled.
(15:1)	If set to 1, carriage control is prespacing. This means the carriage control is written before the data. Otherwise, it is postspacing. Pre or post spacing is set by doing an FControl 1 with the appropriate carriage control character. If postspacing is used, and the previous write request was prespacing, a carriage return and line feed should be output.
QCount	Contains the number of bytes to write.
Data	The write data will be here.

4.1.2 Function 2 - File Open

This request is issued when the user does an FOpen. Once the file system issues this function, it does not wait for a response from the driver. As nothing is ever done by the driver for this function, the pseudo driver will never send it to the monitor.

4.1.3 Function 3 - File Close

This request is issued when the user does an FClose. When the monitor receives this request, it should do the following:

EOR and AEOR are reset.
Line delete echo should be reenabled.

4.1.4 Function 4 - Device Close

This request is issued from the File System when the last FClose is done. When the monitor receives this request, it should do all the things described in Function 3 - File Close. This is also a termination request. The monitor should terminate the connection, and among other things, call PDTerm.

4.1.5 Function 5 - Set Read Timeout

This request will set a timeout for a read. It is set by FControl 4. The read timeout affects only the next read. Timers are not user accessible, so this feature will be handled in the pseudo driver. Therefore, this function will never be sent from the pseudo driver to the monitor.

4.1.6 Function 6 - Set Input Speed

This request is to set the input speed of the terminal. It is set by FControl 10. Some terminal drivers make no distinction between FControl 10 and 11 because it is not possible to run with "split" line speeds.

QParm1	Contains the input speed in characters per second.
QCount	On return, contains the old value of the input speed in characters per second.

4.1.7 Function 7 - Set Output Speed

This request is to set the output speed of the terminal. It is set by FControl 11. Some terminal drivers make no distinction between FControl 10 and 11 because it is not possible to run with "split" line speeds.

QParm1	Contains the output speed in characters per second.
QCount	On return, contains the old value of the output speed in characters per second.

4.1.8 Function 8 - Enable Echo

This request is to enable echoing on the terminal. This is set by FControl 12.

QCount	On return, contains the old echo status - 0 is on, 1 is off.

4.1.9 Function 9 - Disable Echo

This request is to disable echoing on the terminal. It is set by FControl 13. Note that this will not suppress the carriage return/line feed that is sent when a read completes. To suppress this, use FSetMode. See Function 0 - Read for details.

QCount	On return, contains the old echo status - 0 is on, 1 is off.

4.1.10 Function 10 - Disable System Break

This request is to disable the System Break. It is set by Fcontrol 14.

4.1.11 Function 11 - Enable System Break

This request is to enable the System Break. It is set by FControl 15. If System Break is enabled, the BREAK key will cause an interrupt. For a discussion of system break, see section 5.1. Note that system break is available during sessions only.

4.1.12 Function 12 - Disable Subsystem Break

This request is to disable the Subsystem Break. It is set by FControl 16.

4.1.13 Function 13 - Enable Subsytem Break

This request is to enable the Subsystem Break. It is set by FControl 17. If Subsystem Break is enabled, the user can invoke it by typing CONTROL Y. For a discussion of subsystem break, see section 5.2.

4.1.14 Function 14 - Disable Tape Mode

This request is used to disable Tape Mode. It is set by FControl 18. As Tape Mode, which was used to read paper tapes and tape cartridges, is not supported on the newer terminal controllers, the pseudo driver will not send this function to the monitor.

4.1.15 Function 15 - Enable Tape Mode

This request is used to enable Tape Mode. It is set by FControl 19. Like the previous function, this request will never be sent from the pseudo driver to the monitor.

4.1.16 Function 16 - Disable Read Timer

This request is to disable the read duration timer. It is set by FControl 20.

4.1.17 Function 17 - Enable Read Timer

This request is to enable the read duration timer. It is set by FControl 21. Note that this not the same timer used by Function 5 - Set Read Timeout. This timer is used to get the length of time the read takes, not to set a timeout on the read. This request only affects the next read.

4.1.18 Function 18 - Return Read Time

This request is used to return the read time that was enabled by Function 17. It is set by FControl 22.

QCount	On return, contains the duration of the read in hundredths of a second. If the duration was greater than 655.35 seconds, QStat should be set to %163.

4.1.19 Function 19 - Disable Parity

This request is to disable parity checking and generation. It is set by FControl 23. Parity must be disabled in order to transmit and receive binary data.

4.1.20 Function 20 - Enable Parity

This request is to enable parity checking and generation. It is set by FControl 24. The type of parity used is that which was in use before it was disabled. To specify the type of parity, Function 32 - Set Parity, is used.

4.1.21 Function 21 - Set Logon Type

This request is sent by MPE to inform the driver of the logon type.

QParm1	Contains the logon type. 0 is for data, 1 is for sessions, and 2 is for jobs. If the logon type is a session, break should be enabled.

4.1.22 Function 22 - Unused

This function will never be sent from the pseudo driver to the monitor.

4.1.23 Function 23 - Set Terminal Type

This request is used to change the terminal type. It is the same as Function 34.

QParm1	Contains the terminal type.

4.1.24 Function 24 - Allocate Terminal

This request is used to allocate a terminal, setting terminal type and speed. It is the same as Function 33.

QParm1	Contains the terminal type.
QParm2	Contains the terminal speed.

4.1.25 Function 25 - Clear Flush and Write

This request is a special write request. When a terminal goes into break mode, it will throw away all write requests until it gets a Clear Flush and Write request. For a detailed description of break, see section 5. 1. See the description for Function 1 - Write for the applicable parameters.

4.1.26 Function 26 - Enable Line Delete Echo

This request is to enable the printing of "!!!" upon execution of a CONTROLX. It is set by FControl 34.

4.1.27 Function 27 - Disable Line Delete Echo

This request is to disable the printing of "!!!" upon execution of a CONTROLX. It is set by FControl 35. Note that the data is still deleted from the input buffer after a CONTROLX has been typed and a carriage return/line feed is still output.

4.1.28 Function 28 - Quiese IOQ

This request should cause any buffered write data to be output.

4.1.29 Function 29 - Paper Tape Read

This request is used to read paper tapes. It is set by the PTape intrinsic. This function is not supported by the newer terminal controllers and the pseudo driver will not send this function to the monitor.

4.1.30 Function 30 - Set/Clear Break

This function is used to set and clear break mode. For a discussion of system break, see section 5.1.

QParm1

(15:1)	0 - Clear break mode.
	1 - Set break mode.

4.1.31 Function 31 - Set/Clear Console

This request is to set and clear console mode. Console mode is set when the operator types CONTROLA at the console. Because this is a console only function, the pseudo driver will not send this function to the monitor.

QParm1

(15:1)	0 - Clear console mode.
	1 - Set console mode.

4.1.32 Function 32 - Set Parity

This function will set the parity checking and generation characteristics. It is set by FControl 36.

QParm1	0 - No parity 0's.
	1 - No parity 1's.
	2 - Even parity.
	3 - Odd parity.

4.1.33 Function 33 - Allocate Terminal

This request is used to allocate a terminal, setting terminal type and speed. It is the same as Function 24. It is set by FControl 37.

QParm1	Contains the terminal type. If 0, use the configured terminal type.
QParm2	Contains the terminal speed. If 0, use the configured terminal speed.

4.1.34 Function 34 - Set Terminal Type

This request is used to change the terminal type. It is the same as Function 23. It is set by FControl 38.

QParm1	Contains the terminal type.

4.1.35 Function 35 - Get Terminal Type

This request is to return the terminal type. It is set by FControl 39.

QCount	On return, contains the terminal type.

4.1.36 Function 36 - Get Output Speed

This request is to get the output speed. It is set by FControl 40.

QCount	On return, contains the output speed.

4.1.37 Function 37 - Set Unedited Mode

The request puts the terminal in Unedited Mode, or resets the terminal into Edited mode. It is set by FControl 41. See the description under Function 0 - Read on Unedited Mode.

QParm1

(0:8)	Contains the new subsystem break character. This character functions like the CONTROL Y character in Standard Editing Mode and is ignored if subsystem break is not accepted. It is always stripped from the input stream.
(8:8)	Contains the new EOR character. If non-zero, puts the driver into unedited mode. If zero, resets the EOR and puts the driver into edited mode.

4.1.38 Function 38 - Vary Console Mode

This request is to enable the detection of CONTROL A. This detection is only needed at the console. Because this is a console only function, the pseudo driver will not send this function to the monitor.

QParm1	0 - Enable CONTROL A.
	1 - Disable CONTROL A.

4.1.39 Function 39 - Speedsense

Speedsense mode is used when a terminal is not logged on to detect when the user presses return, no matter what speed the terminal is configured at. Because this is a function that only makes sense for locally attached terminals, the pseudo driver will not send this function to the monitor.

4.1.40 Function 192 - Device Control

This request is used to control various aspects of the device. It is set by doing an FDeviceControl with a control code of 192. The individual functions are sent in QParm1. QParm2 specifies the access of the function as follows:

QParm2

1	Read - return requested item in the data area.
2	Write - use the value in the data area to change the item.
3	Read and Write - use the value in the data area to change the item and return the old value in the data area.

Only the listed functions will be sent from the pseudo driver to the monitor. If any others are sent they should be returned with a QStat of 5.

4.1.40.1 Item 28 - Block Mode Support

This function is used to determine the block mode handshake the terminal uses.

QParm1 Contains 28.

QParm2 Contains the access type (read or write).

Data Contains the block mode handshake as an integer:

```
    0 - No block mode              1 - Line Block
    2 - DC1/DC2/DC1 Page Block     3 - Both 1 and 2
    6 - XON/XOFF Page Block        7 - Both 1 and 6
```

If the data is not one of the above values, QStat should be returned as %65 if the data is < 0, %75 if the data is > 7, and %105 if the data is a 4 or 5.

QCount Contains the count. If the access type is 1, if QCount is smaller than 1, QStat should be returned with an %115. If the access type is 2, if QCount is greater than 1, QStat should be returned with an %125. For access type 3, QCount must be 1. If it is not, QStat should be returned with either %115 or %125 as appropriate.

If it is not possible to tell a remote system to set a particular block mode type, only those requests with a QParm2 read (1) access type should be allowed. Others should be returned with a QStat of %15 and QCount should contain a 1, indicating read is the only type allowed.

Also, it is recommended that if the block mode type cannot be directly obtained from the other side, that a 7 be returned in the data. This will allow VPLUS to work on some remote systems. For a detailed description of this, see section 6.1.

4.2 SPECIAL FUNCTIONS

This section will describe what the functions are when QType is 2 or 3. These functions always have the request sent from the monitor to the pseudo driver, and the reply from the pseudo driver to the monitor.

4.2.1 Function 1 – Initialize Driver

This request is sent from the monitor to the pseudo driver. The monitor does not directly send this message, it calls PDInit in order to do this. The pseudo driver will allocate a pseudo terminal and start the logon process on that terminal when this request is received.

QParm1	
(15:1)	If set to 1, indicates a request for a null pseudo terminal. See section 7 for a discussion of null terminals.
QCount	On return, contains the logical device number (ldev) of the pseudo terminal.

4.2.2 Function 2 – Terminate Driver

This request is sent from the monitor to the pseudo driver. The monitor does not directly send this message, it calls PDTerm in order to do this. The pseudo driver will release the pseudo terminal and terminate the session (if there is one) logged onto the terminal.

4.2.3 Function 3 – Invoke Break

This request is sent from the monitor to the pseudo driver. The monitor sends this request when a break is received and system break has been enabled (via QType = 0 and QFunc = 11). For a discussion of system break, see section 5.1.

4.2.4 Function 4 – Invoke Subsystem Break

This request is sent from the monitor to the pseudo driver. The monitor sends this request when a subsystem break is received and subsystem break has been enabled (via QType = 0 and QFunc = 13). For a discussion of subsystem break, see section 5.2.

4.3 STATUS RETURNS

The following status returns are put in QStat before the reply is sent. Other status returns may be added to this list as implementation proceeds.

4.3.1 Status Returns For Normal I/O

For normal I/O, bits 12:3 are the general status and 8:5 are the qualifier code which further defines the general status. Unless indicated, a particular status can be returned by all functions.

4.3.1.1 General Status 1 - Successful

Status %	Meaning
1	Normal Completion
11	Read terminated with special character. Returned only with QFunc = 0 (read).

4.3.1.2 General Status 3 - Unusual Condition

Status %	Meaning
33	I/O Aborted Fxternally
163	Read Time Returned Overflow. Returned only with QFunc = 18 (return read time).
173	Break Stopped Read. Returned only with QFunc = 0 (read).

4.3.1.3 General Status 4 - Irrecoverable Error

Status %	Meaning
4	Invalid Request
14	Transmission Error - This should be used if no other category fits.

4.3.1.4 General Status 5 - Error In Control Information

The following statuses are used for QFunc 192 (Device Control).

Status %	Meaning
5	Invalid item number.
15	Invalid Access for Item. QCount should contain the valid access.
65	Passed value is less than the minimum allowed. QCount contains the minimum value.
75	Passed value is more than the maximum allowed. QCount contains the maximum value.
105	Passed value is unsupported.
115	Count less than needed to return information. QCount contains the minimum space needed.
125	Count greater than available to store information. QCount contains the maximum space available.

4.3.2 Special Functions Status Returns

Status %	Meaning
1	Successful
404	Bad function. QFunc must be 1, 2, 3, or 4.
411	No psuedo terminals available.
421	Break request rejected, break not enabled.
422	Break request rejected, system refused request.
431	Subsystem break request rejected, subsystem break not enabled.
432	Subsystem break request rejected, system refused request.

BREAK AND SUBSYSTEM BREAK

SECTION 5

This section will describe the actions the pseudo driver and monitor perform when a break or subsystem break is received.

5.1 BREAK HANDLING

On an HP terminal, break is invoked when the user presses the BREAK key. Break is used to return to the CI. If a subsystem was running when break was invoked, the user can return to the subsystem by entering the RESUME command.

5.1.1 Enabling and Disabling

In order to enable break, an FControl 15 is done. This will cause a QType 0 pseudo IOQ with a QFunc of 11 to be sent from the pseudo driver to the monitor.

To disable break, an FControl 14 is done This will cause a QType 0 pseudo IOQ with a QFunc of 10 to be sent from the pseudo driver to the monitor.

5.1.2 Invoking

When a break request is received at the monitor, the monitor first needs to determine if break has been enabled. (The details of how a break request is sent to the monitor are dependent on what the protocol of the network is. For Telnet, this would be the IP command.) If break has not been enabled, the pseudo driver is not invoked. If break has been enabled, the following sequence should happen:

```
      Pseudo Driver                           Monitor
                                                Builds a special function,
                                                 type 3 (invoke break)
                                                 pseudo IOQ.  Calls
                                  <-----         PD_Send_Msg.
        Determines if break
          can be invoked.  If
          it can, returns the
          pseudo IOQ with a
          successful status.  ----->
                                                The monitor should abort
                                                  any I/O requests it has
                                                  (QStat is %33).  If a
                                                  read is pending, return
                                                  it with a QStat of %173.

                                                Go into "flush mode".
        Sends a normal function,
          type 30 pseudo IOQ to
          set break mode.     ----->
                                                Break mode will cause the
                                                  monitor to abort all I/O
                                                  requests it has (QStat
                                                  is %33).  If a read is
                                                  pending, return it with
                                                  a QStat of %173.
        Sends normal function,
          type 25 pseudo IOQ to
          clear flush mode and
          output a ":".        ----->
                                                Resets "flush mode".
```

In flush mode, all new read (type 0) and write (type 1) pseudo IOQs should immediately be returned as successful, with no actual I/O taking place. Reads should be returned with QCount equal to 0. Writes that are preemptive [QFlags (12:2)] should be output in flush mode. Flush mode is reset by a clear flush and write (type 25) pseudo IOQ.

In break mode, if there is a subsystem break enabled, it is disabled during the time break mode is set. Break mode will be cleared when the user types ABORT or RESUME. It will be cleared by sending a QType 0, QFunc 30 pseudo IOQ down to the monitor with QParm1(15:1) set to 0. When break mode is cleared, the subsystem break (if there is one) should be reenabled. This type of pseudo IOQ will only be sent if the user was in a subsystem (e.g. EDIT/3000) when break was invoked. If the user was doing a CI command (e.g. LISTF), no break mode pseudo IOQ will be sent.

5.1.3 Errors

It is possible the the invoke break request from the monitor to the driver will fail. This could happen for a couple of reasons. First, at the time when break is being invoked, the operating system may have disabled break, but has not informed the driver yet. Secondly, when the user is in the CI, break is always enabled, but may not always be accepted. It will only be accepted if the user is doing a CI command (e.g. LISTF). If the user is just sitting at the CI prompt with a read and invokes break, it will fail. In both these cases, the request will fail with a status of %405.

The request will also fail if the monitor attempts to invoke break, but it has not been enabled. This could happen if the pseudo driver has been told that break has been disabled, but has not yet sent the monitor the disable break pseudo IOQ. In this case, the request will fail with a status of %404.

5.2 SUBSYSTEM BREAK HANDLING

Subsystem break is usually invoked by typing $\boxed{\text{CONTROL}}$Y. The subsystem break character can be changed by doing an FControl 41 to put the terminal in unedited mode (this will be passed to the monitor with a normal pseudo IOQ of QFunc 37). Subsystem break is used to trap to a user supplied procedure.

5.2.1 Enabling and Disabling

The user indicates the procedure to trap to by calling XCONTRAP. This will also enable subsystem break. The user can disable subsystem break by doing an FControl 16 and enabling it by doing an FControl 17. It makes no sense to do an FControl 17 if there is no trap procedure to call.

The monitor will be sent a pseudo IOQ with a QFunc of 13 to enable subsystem break and a pseudo IOQ with a QFunc of 12 to disable subsystem break.

5.2.2 Invoking

When the subsystem break character is received at the monitor, the monitor first needs to determine if subsystem break has been enabled. (Note that in normal editing mode, $\boxed{\text{CONTROL}}$Y should be stripped from the input data regardless of the state of subsystem break.) If subsystem break has not been enabled, the pseudo driver is not invoked. If subsystem break has been enabled, the following sequence should happen:

```
Pseudo Driver                    Monitor
                                   Builds a special function,
                                    type 4 (invoke subsystem
                                    break) pseudo IOQ.  Calls
                      <-----        PD_Send_Msg.
  Determines if subsystem
    break can be invoked.
    If it can, returns the
    pseudo IOQ with a
    successful status.  ----->
                                   The monitor should complete
                                     all existing I/O as
                                     successful (QStat is 1).
                                     Any read pseudo IOQ
                                     should be returned with no
                                     data.
```

Note that unlike break, there is no such thing as a "subsystem break mode". So, once the special function type 4 pseudo IOQ is returned, the monitor is operating as normal.

5.2.3 Errors

It is possible the the invoke subsystem break request from the monitor to the driver will fail. This could happen for a couple of reasons. First, at the time when subsystem break is being invoked, the user may have disabled subsystem break, but the driver has not yet been informed. Secondly, the user may have enabled subsystem break (via an FControl 17), but not have give a trap procedure to call. In both these cases, the request will fail with a status of %407.

The request will also fail if the monitor attempts to invoke subsystem break, but it has not been enabled. This could happen if the pseudo driver has been told that subsystem break has been disabled, but has not yet sent the monitor the disable subsystem break pseudo IOQ. In this case, the request will fail with a status of %406.

5.3 FAST BREAK AND SUBSYSTEM BREAK

When running over a network, the delay between when a user enters BREAK (or CONTROL Y) and when output finally stops can be quite lengthy. This is because there could be a lot of output data buffered in the network. It is recommended (but not required) that the local side where the real terminal exists implements what is called fast break (or subsystem break).

With this implementation, the local side needs to be told if the monitor side has enabled (subsystem) break. If it has, when a (subsystem) break happens, data will not be written to the terminal, but will be thrown away. Data should start being output to the terminal when the special function pseudo IOQ that says to invoke (subsystem) break is returned by the pseudo driver to the monitor. (This is true regardless of whether or not the status was successful.) So, there needs to be some mechanism for the monitor to tell the local side to stop throwing away data. This mechanism is dependent on the protocol being used, and will not be described in this document. It may not be possible to implement such a mechanism, and so fast break and subsystem break could not be implemented.

NON-HP3000 SYSTEMS

SECTION 6

This section will describe how certain HP3000 specific functions may be done if the local machine is not an HP3000.

6.1 BLOCK MODE SUPPORT

The main goal (as far as block mode support is concerned) of Telnet/3000-V is to be able run VPLUS applications on an HP 9000. It should be possible to support VPLUS applications if the following requirements are met:

- The terminal is one of the newer HP terminals that support XON/XOFF flow control.
- The local system (where the terminal is) should use XON/XOFF flow control to talk to the terminal. It should also support *typeahead*, meaning data coming in from the terminal is buffered, regardless of whether a read is pending or not.

These requirements are met on the HP9000, and so VPLUS applications should work over Telnet. Note however that they are not met on the HP3000-V machines because the drivers do not do XON/XOFF flow control nor do they do typeahead. This means a user will not be able to run a VPLUS application if the Telnet connection is from one 3000-V to another 3000-V. In this case, the user would need NS/VT. This is the recommended software to use for 3000 to 3000 communication.

Strictly speaking, if the local system can accept data without overruns at the line speed, it is not necessary to support XON/XOFF for the transmission of data from the terminal to the host. However, the driver must be able to stop sending data to the terminal if the terminal sends an XOFF.

The older HP terminals only use the ENQ/ACK protocol. So, unless the driver uses ENQ/ACK, when it sends large amounts of data to the terminal (which many block mode applications do), there will be overruns. This is because there is no way for the terminal to tell the system to stop sending data.

6.1.1 XON/XOFF VIEW Read

Section 4.1.0 talks about a VIEW read, and gives an example of the DC1/DC2/DC1 handshake. If the block mode type is 7 (XON/XOFF page block mode) as described in section 4.1.0, a different type of handshake is used. For this handshake, the terminal must be in block mode, the D strap must be turned on (PAGE), the G and H straps are both on, meaning the terminal will send data immediately when the user presses ENTER, echo is off, unedited mode is set, and subsystem break is disabled. VPLUS will send the necessary escape sequences and FControls for this. The handshake is as follows:

```
     Computer                 Terminal
                                User inputs data, presses
                                  ENTER.  Terminal sends
                                  data then record separator
                                  character.
```

Note that there is nothing from the computer indicating a read is pending so data can be accepted. This is why the driver on the computer must support typeahead. The driver will XOFF the terminal if its buffers fill up.

One reason that the DC1/DC2/DC1 handshake cannot be used is that the block mode type will always be returned as XON/XOFF page block mode. This is because it is not possible to determine what block mode the local side supports as there is no way in Telnet to do this.

In order to support older HP terminals, which do the DC1/DC2/DC1 handshake, the driver must be able to accept all the data the terminal sends it at line speed. (There is no way to stop the terminals once they start sending data. An XOFF will not work.) Most drivers cannot do this, so this is another reason this handshake will not work.

6.1.2 OWN Read

The OWN read is also talked about in section 4.1.0. Unlike the VIEW read, there is no such thing as an XON/XOFF OWN read. So in order to do an OWN read, the DC2 the terminal sends must get passed back to the application program. As this is dependent on the driver on the local system, it may work for some systems and not for others.

In order for OWN reads to work, a DC1 needs to be sent to the terminal. Because it will not be known if the local side sent a DC1, when an OWN read is received by the monitor, it should send a write of a DC1 to the local side.

FTP LOGON AND LOGOFF

SECTION 7

This section will describe the routines needed by the File Transfer Protocol (FTP) to access an MPE/V HP 3000 system. The routines will allow FTP to create a user session on a null pseudo terminal, and to terminate a session.

A null pseudo terminal is a bit bucket; it allows a logon on the 3000, but no I/O is sent to the Telnet Monitor. The null terminal completes all I/O except a CI read, the CI read is queued and the session becomes inactive.

7.1 FTPLOGON

Creates a user session using a pseudo terminal and adopts the calling process into the user session.

Syntax:

```
FTPLOGON ( logon, sessioninfo, result )
```

Parameters:

```
logon          character array, by reference.  The logon string to be
(input)        used to create the user session.  Format:

                  user[/userpass].account[/acctpass][,group[/grouppass]]

               This parameter must be delimited by a carriage return
               (octal 15).  The maximum acceptable length for this
               parameter is 80 characters.

sessioninfo    array of 10 16-bit integers, by reference. Information about
(output)       the user session, returned by FTPLOGON and used by FTPLOGOFF

result         16-bit integer, by reference.  Result of call:
(output)             0 - successful
                     1 - could not adopt process into user session
                     2 - could not get info on user session
                     3 - could not create user session; logon may be
                         incorrect or resources may not be available
                     4 - no pseudo terminals are available for session
                     7 - parameter out of bounds
```

Description:

The FTP server calls FTPLOGON to create a user session and to adopt itself into the user session environment. It performs the following functions:

1. Saves identify of the job or session containing the server at entry to FTPLOGON. This should be the job or session running the FTP daemon. The server will return to this session after it has logged off the created session.

2. Allocates a null telnet pseudo terminal for the $STDIN/$STDLIST device for the user session. All writes to this device will be discarded and all reads will be held by the pseudo terminal driver. Consequently the FTP server should refrain from reading or writing to $STDIN/$STDLIST while it is in the user session.

3. Creates the user session for the specified logon using the null pseudo terminal.

4. Adopts the server into the user session. The server will assume the identity, capabilities, and access rights for the user, account, and group of the logon.

7.2 FTPLOGOFF

Adopts the FTP server back into the FTP daemon's session, logs off the user session, and deallocates the telnet pseudo terminal.

Syntax:

```
FTPLOGOFF ( sessioninfo, result );
```

Parameters:

```
sessioninfo    array of 10 16-bit integers, by reference. Information about
(input)        the user session, returned by FTPLOGON.

result         16-bit integer, by reference.  Result of call:
(output)             0 - successful
                     1 - could not adopt process into daemon session
                     5 - could not abort user session
                     6 - could not release pseudo terminal
                     7 - parameter out of bounds
```

Description:

The FTP server calls FTPLOGOFF when it is finished with its business in the user session. FTPLOGOFF performs the following functions:

1. Adopts server back into the original session it was in before the call to FTPLOGON. This should be the FTP daemon's session. If that session no longer exists, or if the original father process no longer exists, a result of 1 will be returned. The server then should take appropriate action, probably aborting itself and the user session.

2. Aborts the user session.

3. Deallocates the null pseudo terminal, so it can be reused for other FTP user sessions.