

DECnet
Digital Network Architecture
(Phase IV)

Network Virtual Terminal
Command Terminal Protocol

Order No. AA-DY88A-TK

software **digital**

DECnet
Digital Network Architecture
(Phase IV)

Network Virtual Terminal
Command Terminal Protocol

Order No. AA-DY88A-TK

December 1984

This document describes the Network Command Terminal services, which provide the function and protocol for terminal handling in distributed Digital systems. The Network Command Terminal services is part of the Terminal Software Architecture (TSA) — and TSA is part of the Digital Network Architecture.

SUPERSESSION/UPDATE INFORMATION: This is a new manual.

VERSION: 2.0

To order additional copies of this document, contact your local
Digital Equipment Corporation Sales Office.

digital equipment corporation • maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.


The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital or its affiliated companies.

Copyright © 1984 by Digital Equipment Corporation

The postage-prepaid Reader's Comments form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC
DECmate
DECnet
DECUS
DECwriter
DIBOL


MASSBUS
PDP
P/OS
Professional
Rainbow
RSTS
RSX

RT
UNIBUS
VAX
VAXcluster
VMS
VT
Work Processor

CONTENTS

1.0	INTRODUCTION	4
1.1	Relation to Digital's Network Architecture	4
1.2	Relation to Digital's Terminal Software Architecture	4
1.3	Definition of Character Set	4
1.4	Requirements, Goals, and Non-goals	5
2.0	NETWORK COMMAND TERMINAL OVERVIEW	7
2.1	Host Terminal Interface	8
2.1.1	Writing Characters	8
2.1.2	Reading Characters	9
2.1.3	Out-of-band Input	9
2.1.4	Writing Characteristics	9
2.1.5	Reading Characteristics	9
2.2	Server Terminal Interface	9
2.2.1	Echoing	10
2.2.2	Type-ahead	10
2.2.3	Input Editing	10
2.2.4	Output Control	10
2.2.5	Out-of-band Input	10
2.2.6	Quoting	10
2.2.7	Output/Echo Synchronization	11
2.3	Multi-character Input Tokens	11
2.4	Escape Sequences	11
2.5	A Data Flow Model	11
2.5.1	Pre-input Process	12
2.5.2	Input Process	14
2.5.3	Input Process/Host Terminal Interface Interaction	15
2.5.4	Output Procedure	15
2.5.5	Output Process	16
2.5.6	Synchronization	16
3.0	INTERFACES	18
3.1	Host Terminal Interface	18
3.1.1	Reading Normal Characters	18
3.1.2	Reading Out-of-band Characters	24
3.1.3	Writing Characters	24
3.1.4	Control and Status Functions	26
3.1.5	Reading and Writing Characteristics	27
3.2	Server Terminal Interface	29
3.2.1	Quoting	29
3.2.2	Output Control	29
3.2.3	Input Editing	29
3.2.3.1	Redisplay Input	30
3.2.3.2	Delete Character	30
3.2.3.3	Delete Word	30
3.2.3.4	Clear Input	30
3.2.3.5	Clear Type-ahead	30
3.3	Terminal Characteristics	31
3.4	Termination Set	34
4.0	OPERATION	35
4.1	Interfaces and Protocols	36
4.2	Host/Server Division of Labor	36
4.3	Data Channels	36

4.4	Protocol Message Overview	37
4.5	General Message Processing	37
4.6	Protocol Errors	38
4.7	Initialization	38
4.8	Characteristics Management	38
4.9	Read Request Processing	39
4.9.1	Issuing the Read	39
4.9.2	Unreading	39
4.9.3	Position Modeling	40
4.9.4	Read Completion	40
4.9.5	Input Editing	40
4.9.5.1	Distributed Input Editing	41
4.9.5.2	Selecting Distributed Input Editing	41
4.10	Other Input Processing	42
4.10.1	Input Escape Sequences	42
4.10.2	Raising Input	43
4.10.3	Out-of-band Processing	43
4.10.4	Control-V	43
4.10.5	Control-X	43
4.10.6	Control-O	43
4.10.7	Errors on Input	44
4.11	Write Request Processing	44
4.12	Other Output Processing	45
4.12.1	Output Discard State Handling	45
4.12.2	Locking	46
4.12.3	Output Escape Sequences	47
4.13	Additional Status and Control Operation	47
4.13.1	Reading and Writing Characteristics	47
4.13.2	Clearing Input	48
4.13.3	Input Character Count Handling	48
4.14	Foundation Services Interface Events	48
4.15	Protocol Evolution	49
4.16	Network Command Terminal Protocol Messages	49
4.16.1	General Message Format	51
4.16.2	Initiate (H <--> S)	52
4.16.3	Start Read (H ---> S)	54
4.16.4	Read Data (H <--- S)	57
4.16.5	Out-of-Band (H <--- S)	58
4.16.6	Unread (H ---> S)	59
4.16.7	Clear Input (H ---> S)	59
4.16.8	Write (H ---> S)	59
4.16.9	Write Completion (H <--- S)	61
4.16.10	Discard State (H <--- S)	62
4.16.11	Read Characteristics (H ---> S)	62
4.16.12	Characteristics (H <--> S)	63
4.16.13	Check Input (H ---> S)	64
4.16.14	Input Count (H <--- S)	64
4.16.15	Input State (H <--- S)	64
4.16.16	Reserved for VMS	64
4.16.17	Reserved for VMS	65
4.16.18	Reserved for VMS	65
4.17	Selector Values for Characteristics	65
4.17.1	Foundation-maintained Characteristics	66
4.17.2	Handler-Maintained Characteristics	68
4.17.2.1	CHARACTER-ATTRIBUTES Compound Characteristic	69

APPENDIX A Escape Sequence Recognition Algorithm

FIGURES

2-1	Distributed Common Terminal Services Model	7
2-2	Refined Common Terminal Services Model	12
4-1	Structure of Distributed Terminal Handler	35

1.0 INTRODUCTION

This specification describes a model for communication between terminal-handling subsystems and operating systems in a communications network. The integration of this communication with other aspects of terminal operation derives from the Digital Terminal Software Architecture.

Systems in which this model could be implemented include, but are not necessarily limited to, intelligent terminals, host systems, front end systems, and terminal concentrators.

This specification defines

- o The services (interface functions or primitive operations) and semantics (but not the syntax) of the services provided by this model for terminal handling by Digital's Terminal Software Architecture
- o The communication protocol (both syntax and semantics) used by this model to provide the defined services

This document consists of four major sections:

1. A description of requirements and goals
2. A "black-box" model that identifies important interfaces
3. An interface description
4. A protocol model that defines messages and the operation of the modules that (a) provide the interfaces and (b) send and receive the messages

1.1 Relation to Digital's Network Architecture

The remote command terminal architecture is a component of the network application layer of Digital's Network Architecture.

1.2 Relation to Digital's Terminal Software Architecture

The remote command terminal architecture is a component of the mode layer of Digital's Terminal Software Architecture.

1.3 Definition of Character Set

This specification is intended for use with Digital's 8-bit coded character set. The term "character" means an 8-bit value from this character set. This character set defines the names of characters referred to in this specification (e.g., BEL).

1.4 Requirements, Goals, and Non-goals

Requirements are those attributes that the model described herein must have. The requirements of the remote command terminal architecture are:

- o LOGON. A person using one of a variety of terminals can log on to any Digital system (specifically, RSTS/E, RSX-11M, IAS, TOPS-10, TOPS-20, or VMS), give operating system level commands to the system, receive system responses, and communicate with all programs that are run under the operating system provided that a path exists between the terminal and host consisting of communication links and DECnet systems.
- o STANDARDS. This model adheres to any previously defined corporate terminal architecture standards.
- o MODULARITY. Terminal functions that can be classified together (e.g., input line editing) are either handled completely by the model or handled completely outside the model.
- o SIMPLICITY. The implementation of this model in hosts, communications processors, terminal concentrators, and terminals is sufficiently simple both to be understood by a wide variety of Digital developers and support personnel and to be contained in a reasonable amount of memory.
- o REASONABLE PERFORMANCE. The performance of products adhering to this model can be made sufficiently good that users (people and programs) find it acceptable.
- o EVOLUTION. This model allows future modifications.
- o FLEXIBILITY. This model allows implementors to trade performance for size.
- o TERMINAL SUPPORT. The model will support all specific and generic terminal types defined by the Digital Terminal Software Architecture.

"Goals" are those attributes that are desirable in the model described herein. The goals of the remote command terminal architecture, listed in descending order of preference, are:

- o HUMAN ENGINEERING. The model should be compatible with established concepts of ease of use at the terminal/user interface.
- o EXCELLENT PERFORMANCE. Terminal response time consistent with the service provided by the lower layer communication service used by the implementation.

- o EXTERNAL STANDARDS COMPATIBILITY. The model should not preclude host compatibility with existing external standards; for example, ANSI escape sequences.
- o TERMINAL CHARACTERISTICS. The model should support all terminal characteristics of all host systems as seen by both terminal users and programs (implying that all existing programs will communicate with terminals accessed via implementations of this model).
- o NEW STANDARDS. Where possible, similar functions seen by a terminal user (e.g., the echoing of "delete previous line") should be standardized across hosts and servers.

The non-goals of the remote command terminal architecture are:

- o INVENTION. The model should not include new terminal services, with specific reference to the following:
 - forms applications
 - graphics applications
 - editor applications

The model may provide a set of terminal handler "primitives", which are essentially atomic functions, which may be used by the applications listed above, but it will not provide any explicit, sophisticated support for these applications.

- o ROGUE DEVICES. The model should not provide support for "strange devices" (cassette drives, etc.) simply because such devices may be connected to a terminal interface. The model does not need to support any device that does not behave like a "normal" interactive terminal. In particular, support is not guaranteed for devices that do not support local flow control conventions (XON/XOFF for Digital terminals).

2.0 NETWORK COMMAND TERMINAL OVERVIEW

This section describes a model for distributed terminal handling. In this model, the terminal-handling functions historically provided by system terminal drivers are provided by a collection of terminal-handling modules distributed among the systems of a network.

There are two types of terminal handling modules: those residing in host systems and those residing in server systems. A host system runs an operating system that has resources such as application programs, compilers, and command language interpreters available to human terminal users. A server system is an intelligent system (in that it is capable of communicating with a host system via a protocol, and capable of communicating with a human terminal user via a command language) to which one or more terminal devices (keyboards, printers, screens, etc.) are attached. Examples of a server system are: a PDP-11/23 operating as a dedicated terminal concentrator; a personal computer attached to a network, and a VMS system that allows a user to issue the SET HOST command to connect to a remote host. Figure 2-1 illustrates these concepts.

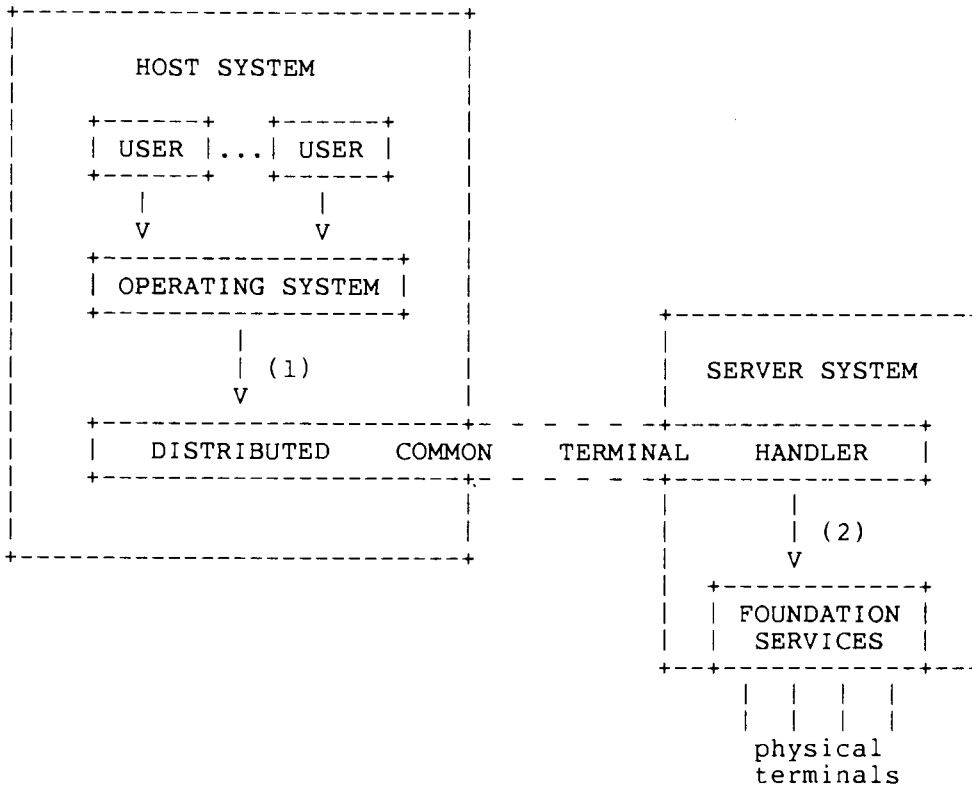


Figure 2-1 Distributed Common Terminal Services Model

In Figure 2-1, the module labeled Foundation Services is included for compatibility with the Foundation Services of the Digital Terminal Software Architecture. Briefly, this module is introduced to establish a clean position for future incorporation of terminal-model-independent control functions (for example, cursor movement) and for logical to physical terminal mapping functions (these functions would allow a CRT user, for example, to create two logical terminals, log each one on to a separate host, and see the output from each host appear in a separate window on the screen). The complete functions of this module are described in the Network Virtual Terminal Foundation Services. For the present, the Foundation Services module provides a one-to-one correspondence between a logical terminal and a physical terminal.

Figure 2-1 also presents a logical view of distributed terminal handling. The emphasis here is in viewing the host-resident and server-resident terminal-handling functions as creating a single, distributed terminal handler. This view of terminal handling is described in more detail below.

Figure 2-1 shows how each terminal attached to a given server may communicate with a given host. In a network, each such terminal may communicate with a different host, and a given host communicate simultaneously with terminals attached to multiple servers. Figure 2-1 and the model presented below concentrate on the communication of a single terminal with a single host. Multiple communications may occur simultaneously in a network.

In Figure 2-1, two interfaces, labeled (1) and (2), are identified. These are key interfaces in that their definition captures the essence of the functions of the proposed distributed terminal handler. This specification does not require that these interfaces be implemented as described in any host or server system. However, their precise definition is viewed as the best way to develop the protocol description that is presented herein.

2.1 Host Terminal Interface

The host terminal interface, labeled (1) in Figure 2-1, is the interface that an operating system uses to communicate with a terminal. This interface is defined to allow the operating system to provide application programs with the same terminal access functions they have had historically via system-resident terminal handlers.

Examples of functions available at the host terminal interface follow.

2.1.1 Writing Characters - The operating system can send characters from an output buffer to the terminal.

2.1.2 Reading Characters - The operating system can provide a read buffer for receiving characters from the terminal. The handler will terminate the read on a "termination" condition. A termination condition can occur in a number of ways: the read buffer is filled; a termination character is entered (the operating system defines the active set of termination characters when it issues the read request); the interarrival time of characters from the terminal exceeds a threshold; or by request of the operating system (via an "unread" function). The handler provides input buffer editing, echoing, and other input-related functions. Synchronization of input character echoing with output characters is also done within the handler.

2.1.3 Out-of-band Input - The operating system can read out-of-band characters (e.g. control-C) that have been entered at the terminal. These characters are delivered to the operating system independently from the normal stream of input characters. The host specifies which characters are out-of-band characters and enables this function by writing characteristics.

2.1.4 Writing Characteristics - The operating system can set various terminal characteristics. Some of these characteristics have been mentioned above. Additional examples of characteristics are:

- o the "normal echo" flag - a Boolean variable that defines whether or not characters other than control characters are echoed
- o the "raise input" flag - a Boolean variable that defines whether or not an entered lower case character is raised to an upper case character before being processed in any other way

Further characteristics are described in detail later.

2.1.5 Reading Characteristics - The operating system can read any characteristic.

2.2 Server Terminal Interface

The functions of the server terminal interface, labeled (2) in Figure 2-1, can be described at two levels. At the lowest level, it is the interface over which characters to and from a human terminal user pass. This interface level is described in detail in the Foundation Services specification. At a higher level are the functions perceived by the human using the terminal. Examples of the higher level functions that are available at the server terminal interface follow.

2.2.1 Echoing - Echoing is the function of printing back out on the terminal any character that is input. A character is echoed when it satisfies the current Read pending from the operating system. For purposes of echoing, input characters are either control characters (and DEL) or non-control characters. Non-control characters either echo as themselves or do not echo as specified by the NORMAL-ECHO characteristic. Control characters (and DEL) either do not echo, echo as themselves, or echo in a "standard way" as specified by the CHARACTER-ATTRIBUTES characteristic. Echoing in a "standard way" allows the operating system, for example, to have a <carriage return, line feed> echoed when a carriage return is entered.

2.2.2 Type-ahead - Type-ahead is the function of internally queuing entered input characters for which there is no current pending Read request from the operating system.

2.2.3 Input Editing - Input editing includes the functions associated with the DELETE, control-U, and control-R input characters (viz., delete previous character, delete current input, and display current input). It also includes the functions associated with the VMS control-X character (delete type-ahead as well as current input) and the TOPS-20 control-W character (delete previous word).

2.2.4 Output Control - Output control is the function associated with the control-O input character (alternate keystrokes discard some amount of queued output and reenable output). This function also sets a characteristic that can be read by the operating system indicating if output discard operation is in effect.

2.2.5 Out-of-band Input - Out-of-band is the function generally associated with the control-C, control-Y, or control-T input character (depending on the operating system). This function has two forms. In either form, an entered out-of-band character is placed in a separate out-of-band buffer that can be read by the operating system independently from normal input. In one form (normally used for control-C and control-Y), the entry of this character also clears any type-ahead and terminates the current Read, if any; however, in the other form it doesn't.

2.2.6 Quoting - Quoting is a function that allows a terminal user to pass through, as data, a character that is otherwise defined as a control character (e.g., an input editing character, an output control character, or an out-of-band character). This function is associated with the control-V character. Its effect is to shield the following character in the input stream from recognition as a control character.

2.2.7 Output/Echo Synchronization - Output/Echo synchronization is the function of synchronizing output from the operating system with echoed output resulting from operating system Read requests. The form that this synchronization takes is controlled by the host. In general, if an operating system requests a Read, the characters from any previous output function will appear on the terminal before the first character echoed as a result of the Read. However, Write output can "break through" a Read in progress under certain conditions. The resolution of the "collision" of input echoing and these types of output can be controlled by the operating system through the use of a LOCKING flag on WRITES from the operating system.

2.3 Multi-character Input Tokens

Certain sequences of input characters, called "tokens", are considered as groups. These are: (1) a quote character followed by any character and (2) an escape sequence. The characters which constitute a token will not be divided across two or more Reads unless the first character of the token is the first character placed in an empty Read buffer. This can not happen for quoted characters except for the incongruous case of one-character Reads. The details of escape sequence handling are discussed in the next section and in Sections 4.10 through 4.12.

2.4 Escape Sequences

Escape sequence recognition may be enabled or disabled. When enabled, escape sequences are recognized as a syntactic unit (token) and treated, wherever possible, as a unit. When enabled, on input they terminate Reads, do not echo, and have precedence greater than that of enabled editing characters. On output, they are written to the Foundation Services transparently and cause the horizontal and vertical position modeling to be set back to the origin. The network command terminal module performs only escape sequence recognition; it does not parse escape sequences. For the convenience of implementors, Appendix A provides an algorithm for escape sequence recognition.

2.5 A Data Flow Model

Figure 2-2 depicts the distributed common terminal handler of Figure 2-1 as if it were a single module in a single system, internally composed of three processes, a procedure, and four buffers. This version of the model emphasizes the flow of data between the operating system and the terminal.

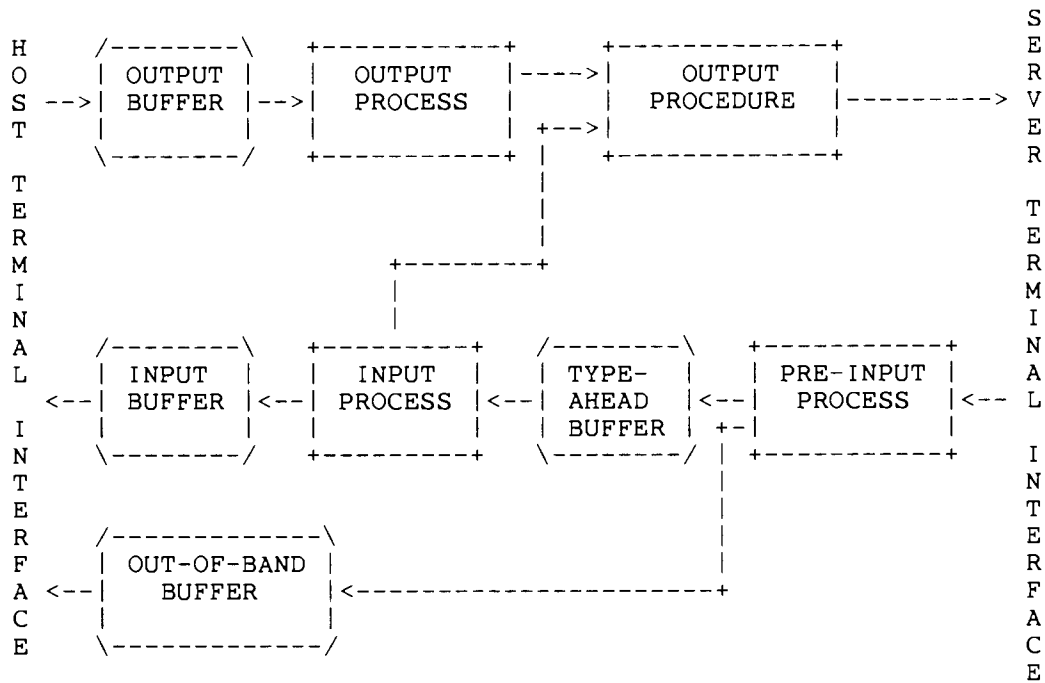


Figure 2-2 Refined Common Terminal Services Model

The model depicted in Figure 2-2 and discussed below describes data movement between the operating system and the terminal. In this model, distributed operation is shown as the cooperation of multiple processes. Each process is assumed to run at its own speed, independently from every other process.

2.5.1 Pre-input Process - The pre-input process reads characters from the server terminal interface. Each character is examined to determine how it should be processed. The examination proceeds as follows.

First, if the input character is defined as an out-of-band character (the operating system may define multiple such characters), the character is placed in the out-of-band buffer. The out-of-band buffer, as well as the remaining buffers, is really a queue in which the characters placed in it retain their order. There are three types of out-of-band characters:

1. Hello -- a "hello" out-of-band character normally has no effect on the normal input data stream (as an option, however, a copy of a "hello" out-of-band character can be included in the type-ahead buffer as well as in the out-of-band buffer).

2. Immediate clear -- an "immediate clear" out-of-band character clears the type-ahead buffer and terminates any pending Read. Only control characters can be "clear" out-of-band characters.
3. Deferred clear -- a "deferred clear" out-of-band character, unlike the other out-of-band characters, consists of two consecutive, identical control characters treated as a unit. It clears the type-ahead buffer and terminates any pending Read. Only one character is put into the out-of-band buffer.

Out-of-band characters are read one at a time at the host-terminal interface.

Normally, non-control out-of-band characters are not echoed -- if the option to include a copy of a "hello" out-of-band character as normal input is exercised, the copy (treated as a normal input character) may be echoed depending on whether that character would normally be echoed. Control out-of-band characters are echoed according to the CHARACTER-ATTRIBUTES characteristic. They are echoed immediately without regard to whether the server is locked.

Second, if control-X is enabled (control characters are enabled by the operating system via writing characteristics), and a control-X character is read, the type-ahead buffer is cleared, and, if a read is pending, the control-X is changed to a control-U and placed in the type-ahead buffer.

Third, if the control-O function is enabled, and a control-O character is read, an "output-discard" state variable used by the output process is toggled. This state variable takes on "discard output" and "don't discard output" values. The value of this state variable affects the operation of the output process, as described below. The operating system can read the value of the state variable by reading its state. It may set the value to "don't discard output" whenever it issues a Write request.

Finally, if the input character is not one of the above, it is placed in the type-ahead buffer. Characters flagged with an error (i.e., line break, framing error, parity error, and receiver overrun) are either discarded or placed in the type-ahead buffer (together with the error) according to the value of the ERROR-PROCESSING characteristic.

If the pre-input process cannot completely process a character because the out-of-band buffer or the type-ahead buffer is full or because the output procedure rejects the call to output (because it was rejected by the Foundation Services), the pre-input process is modeled as looping until the character can be completely processed. During this time, the pre-input process is not reading characters from the server terminal interface. This causes data entered by the terminal user to be queued within the Foundation Services module and, possibly, within hardware. Eventually, input data is lost if the pre-input process cannot make progress. (Notification of data loss to the terminal user is a function performed by the Foundation Services module and/or terminal hardware.)

2.5.2 Input Process - Where an input buffer is present (i.e., the host has a Read request outstanding), the Input Process reads characters from the type-ahead buffer and writes them to the input buffer. Each character is examined to determine how it should be processed. The examination proceeds as follows.

First, if escape sequence recognition is enabled and the character is part of an escape sequence, it is processed by the escape sequence state machine. If the character is the last character of an escape sequence, the Read is terminated. The details of escape sequence processing are in Section 4, OPERATION.

Second, if the character is an enabled input editing character, the input editing function is performed. The defined input editing functions are:

- o Delete character (DEL) - this function causes the character at the end of the input buffer to be removed and "unechoed".
- o Delete word (control-W) - this function causes the word at the end of the input buffer to be removed and "unechoed".
- o Delete input (control-U) - this function causes the input buffer to be emptied and the prompt redisplayed on the next line.
- o Redisplay input (control-R) - this function causes the entire input buffer plus the prompt (if any) to be re-echoed starting on the next line of the presentation device.

Third, if the character is a termination character, the input process terminates the current Read and returns the input buffer to the operating system (see below). The termination character is echoed if the operating system has enabled echoing for this termination character.

Finally, if the character is a normal data character, it is placed in the input buffer and echoed. If the character fills the buffer, the current Read request is terminated.

In addition to the processing described above, the input process treats the following as termination conditions:

1. no character appears in the type-ahead buffer for a continuous period of time (defined by the operating system)
2. the input process receives an "Unread" request from the operating system
3. the user enters an enabled DEL, control-W, control-U, or control-X character while the input buffer is empty (this termination condition is selected by the operating system independently for each Read request)

4. an enabled vertical change on the presentation device occurs as the result of echoing a user entered character (this termination condition is selected by the operating system independently for each Read request)
5. an error (i.e., line break, framing error, parity error, or receiver overrun) is flagged on an input character from the user

See Section 4, OPERATIONS, for details of the handling implied by these conditions.

As in the case of the pre-input process, if the input process cannot completely process a character because the operating system has no active Read request or because the output procedure rejects the call to output the echo, the input process is modeled as looping until the character can be completely processed. During this time, the input process is not reading characters from the type-ahead buffer. This eventually causes the pre-input process to stop processing data as described above.

2.5.3 Input Process/Host Terminal Interface Interaction - The preceding description of the input process refers to "returning the input buffer to the operating system". This can be visualized as follows.

When the terminal is first connected to the operating system, the input buffer shown in Figure 2-2 is not really present. The operating system makes the input buffer "present" by issuing a Read request at the host terminal interface, specifying the address of an input buffer. "Returning the input buffer to the operating system" is equivalent to marking the input buffer as "Read complete." Such an input buffer is not really "present" in the handler thereafter.

The Read function invoked by the operating system is powerful. It allows the operating system to preload the input buffer (for prompts), to define a portion of the input buffer at the beginning to be not deletable by input editing functions (also for prompts), and to cause the buffer to be "echoed" from any position to the end (for TOPS-20 input recognition operation). The Read function is defined in more detail below.

2.5.4 Output Procedure - The output procedure coordinates the output of characters to the Foundation Service from both the output process and the input process (for echoing).

The output procedure is called by the output process to write output characters and by the input process to write echoed characters. The output process defines the points in its stream of output characters at which it is willing to let echo characters be output. This is done via "lock" and "unlock" functions. "Lock" is a request to block the input process. "Unlock" is a request to unblock the input process. When output isn't locked, echo takes precedence.

2.5.5 Output Process - The output process handles the output-discard state and sends characters from the output buffer to the output procedure. The operation of this process proceeds as follows.

A state variable controls output data discarding. If the value of the state variable is "discard output", the output process discards data in the output buffer; otherwise, it calls the output procedure to write the data to the terminal.

Data in the output buffer consist logically of characters and flags. There are "start-of-message" and "end-of-message" flags. These flags are set as a result of the host terminal interface Write function, described in more detail below.

A "start-of-message" flag may "lock" the output procedure so it accepts no characters for echoing from the input process. There are two "end-of-message" flags. One "unlocks" the output procedure after the last character from the Write has been processed. The other optionally causes a "redisplay" of the input buffer if the output procedure was just "unlocked".

If the output process cannot completely process a character because the server terminal interface is not accepting characters (this could happen if the terminal user enters an XOFF character), the output process is modeled as looping until the character can be completely processed. During this time, the output buffer is not being emptied. This will eventually be observed at the host terminal interface as an inability to successfully issue the Write function.

2.5.6 Synchronization - The description of two points of synchronization concludes this section.

When the operating system has no outstanding Read request, it can redefine the echo characteristics without a race condition for normal input characters. That is, this operation ensures that there is no ambiguity about how a given normal input character will be echoed. If the operating system has a Read request outstanding while it changes the echo characteristics, the echo translation for an input character is indeterminate.

This description of conflict resolution by the output procedure pertains to a conflict of echo data with output data. In that case, the echoing is a result of a Read request given by the operating system BEFORE the Write request that produced the output. In the case of the conflict of echo data with output data when the corresponding Read request was made AFTER the Write request, the distributed terminal handler always completes the output before echoing the input. This means that if an operating system issues a Write, then a Read, then a Write, the output data from the first Write request is guaranteed to be given to the terminal before the first echoed character from the Read request. The output from the second Write request "breaks through" the Read request if the Read is still active when the Write arrives. If this happens, the optional locking of the output procedure determines whether or not echoes from the Read can be intermingled with output from the Write.

The definition of output/echo synchronization given above means that a Read request may be delayed arbitrarily long behind an output request (if, for example, the terminal user enters a XOFF with no succeeding XON for a long period of time) even in the case where the operating system has set the echo characteristics to "don't echo" for the Read. This means that input from and output to "full duplex" terminals does not occur simultaneously with complete independence.

3.0 INTERFACES

As stated previously, the two major interfaces to the handler are the host terminal interface, which exists in a host, and the server terminal interface, which exists in a server. Both are described below.

3.1 Host Terminal Interface

The host terminal interface functions are modeled as being provided by subroutines. Each subroutine description consists of a name followed by (in parentheses) a list of arguments and return values. The arguments appear first and are separated from the return values by a semicolon. Optional arguments are enclosed in brackets ([...]).

The following conventions pertain to the interface description:

1. Each function in this interface includes a portal identifier that specifies the associated logical terminal. A portal identifier is defined in the Network Virtual Terminal Foundation Services.
2. The term "buffer" refers to the combination of address and length information that identifies data and its location.
3. There are two types of interface functions: those that complete immediately ("atomic functions"), and those that queue a request to perform a function at some point in the future ("queued requests"). Queued requests require resources to queue the request therefore, have "insufficient resources" failure returns. If a queued request succeeds (the request is successfully queued), then the requested function will be performed sequentially with respect to all other queued requests. For example, if a queued request to WRITE-CHARACTERISTICS is followed by a queued request to READ-CHARACTERISTICS, the values returned from the Read request will reflect the previous Write request.

The interface functions described below contain "polling" functions. A polling function obtains status about a previously queued request. An implementation would probably convey this kind of status through an event-queuing mechanism rather than polling. The polling form was chosen here so that all information flowing between the requester of a function and the provider of the function would be modeled in a single, consistent way.

3.1.1 Reading Normal Characters - Some of the arguments for the READ function are index values into the input buffer. These indexes are 16-bit values. The maximum size of an input buffer is 2**16-1 bytes.

```
READ (PORTAL-ID, BUFFER, END-OF-DATA, UNDERFLOW-HANDLING, NO-ECHO,
NON-DEFAULT-TERMINATION-SET, CLEAR-TYPE-AHEAD-FLAG,
FORMATTING-FLAG, TERMINATOR-ECHO-FLAG, [RAISE-INPUT],
[RECOGNIZE-INPUT-ESCAPE-SEQUENCES] [DISABLE-CONTROL],
[TIMEOUT,] [END-OF-PROMPT,] [START-OF-DISPLAY,] [LOW-WATER,]
[TERMINATION-SET]; RETURN)
```

queues a read request.

where the parameters of the read function are:

- BUFFER is a buffer to receive input; the maximum size is 2**16-1 bytes.
- END-OF-DATA is the character position after the end of existing data. The first character of new input data will go into this position. This pointer also marks the end of the display.
- UNDERFLOW-HANDLING defines the action when an enabled DEL, control-W, or control-U character is entered for an empty input buffer. It is one of the following:
 - IGNORE - ignore such a character
 - NOTIFY - send a BEL character to the terminal
 - TERMINATE - treat it as a READ termination
- NO-ECHO is a Boolean flag. If TRUE, this overrides any echoing specified by the characteristics.
- NON-DEFAULT-TERMINATION-SET is a Boolean flag. If FALSE, it causes the server to use a universal default termination set for this READ; the default is all control characters except control-R, control-U, control-W, BS and HT. If TRUE, the server uses either the termination set specified by this READ or, if no termination set is specified with this READ, the last termination set it received with a previous READ.
- CLEAR-TYPE-AHEAD-FLAG is a Boolean flag. When it is TRUE, the type-ahead buffer is cleared before the READ is processed.
- FORMATTING-FLAG is a Boolean flag. If TRUE and the last character output was a CR, then output a LF (avoid overprinting); in addition, if the first character of the preloaded input would echo as a LF, ignore it (avoids accidental double spacing). If FALSE, no special action.
- TERMINATOR-ECHO-FLAG is a Boolean flag. If TRUE, termination character echoed; if FALSE, it is not echoed.

- RAISE-INPUT is a Boolean flag. If TRUE, lowercase alphabetic characters are converted to uppercase for this READ (overriding the value of the corresponding characteristic). If FALSE, no conversion is made (overriding the value of the characteristic). If this optional parameter is not present, the characteristic prevails.
- RECOGNIZE-INPUT-ESCAPE-SEQUENCES is a Boolean flag. When it is TRUE, the distributed terminal handler recognizes input escape sequences and processes them as described in the Input Escape Sequence subsection of Section 4.10. When the flag is FALSE, input escape sequences are not recognized and the characters of the escape sequence are treated as individual input characters. If this parameter is not present, input escape sequence processing is handled according to the value of the INPUT-ESCAPE-SEQUENCE-RECOGNITION characteristic.
- DISABLE-CONTROL defines which control characters are disabled as control characters and processed as normal data. This specification overrides the value of characteristics (e.g., control-U) for the duration of the read. It is one of the following:

NONE	all control characters perform their normal function (subject to the current characteristics).
CLEAR and REDISPLAY	the input editing control characters control-U and control-R are treated as normal data characters (i.e., they are disabled as editing characters for the duration of this read).
EDIT	input editing control characters (DEL, control-W, control-U, and control-R) are treated as normal data characters.
ALL	all control characters are treated as normal data characters except XON and XOFF; this includes all input editing characters, control-O, control-X, and all out-of-band characters.

NOTE

As XON and XOFF recognition is in the Foundation Layer, turning it on and off must be done by setting the Foundation INPUT-FLOW-CONTROL characteristic appropriately. Thus, for Read-Pass-All, the host must alter the Foundation INPUT-FLOW-CONTROL characteristic as well as the relevant CTERM characteristics. It is also recommended that Read-Pass-All be implemented by altering the appropriate CTERM characteristics (for the control characters) rather than using the ALL flag on the READ request. The reason for this is that there are race conditions between READs (control-X isn't disabled between READs for starters); use ALL with great care.

- TIMEOUT is the intercharacter arrival timeout value in seconds. If a character does not arrive at the beginning of the type-ahead buffer during this amount of time after the previous character has been removed, the READ terminates. A zero (0) value means to take the characters currently in the type-ahead buffer and terminate the READ immediately without waiting for further input.
- END-OF-PROMPT is the character position after a non-deletable prompt. This prompt may be displayed along with the rest of the input buffer, but it cannot be deleted by the terminal user. If this argument is not present, the end of prompt position is set to the beginning of the buffer.
- START-OF-DISPLAY is the first character position of data to be displayed immediately. The data is displayed by "echoing" it. That is, the characters to be displayed are translated into their echo representation just as if they had been entered at the terminal. If this argument is not present, the start of display position is set to the end-of-data value.
- LOW-WATER is the last character position in the buffer that is to be considered not to have been modified by input editing since the READ was issued. This parameter may be decremented during the READ, but is not incremented. The modified value of this argument is returned with a read completion indication via the READ-POLL function. If this argument is not present, the low water position is set to the beginning of the buffer.
- TERMINATION-SET is the set of characters that can terminate the READ. This set takes the form of a 256-bit mask. If this argument is not present, the termination set most recently specified by the READ function is used.

- RETURN is one of the following:
 - success - Read request queued
 - failure - Read request currently queued
 - failure - inconsistent arguments

Certain combinations of pointer values are considered as errors. An example is when END-OF-DATA is less than END-OF-PROMPT.

UNREAD (PORTAL-ID, CONDITION; RETURN) -- queues a request to (conditionally) terminate a previously queued Read request.

- CONDITION is one of the following:
 - ALWAYS -- terminate a Read unconditionally
 - EMPTY -- terminate a Read only if the type-ahead and input buffers are empty (except for a prompt).
- RETURN is one of the following:
 - success
 - failure - Unread request currently queued
 - failure - no unterminated Read requests outstanding

If CONDITION has the value ALWAYS and input escape sequence recognition is enabled, this resets the input escape sequence state machine.

It is indeterminate if the current Read request will be terminated by this request. This is due to the race condition that can occur within the handler when a Read request is being terminated by internal algorithmic activity (after a termination character is entered by the terminal user) at approximately the same time that the operating system issues the Unread request.

READ-POLL (PORTAL-ID; RETURN, BUFFER, LOW-WATER, MORE-DATA-FLAG, VERTICAL-POSITION, HORIZONTAL-POSITION TERMINATOR-POSITION)

checks if a previously queued READ request is complete.

- RETURN is one of the following:
 - success - Read terminated by a termination character
 - success - Read terminated by a valid escape sequence
 - success - Read terminated by an invalid escape sequence
 - success - Read terminated by an out-of-band character

- success - Read terminated because the buffer filled
- success - Read terminated by intercharacter arrival timeout
- success - Read terminated by an Unread request
- success - Read terminated by underflow (e.g., a DEL character was entered when the input buffer was empty)
- success - Read terminated by an absentee token (complete token would not fit in current Read -- still in server)
- success - Read terminated by line break (final character in buffer is NUL received with break)
- success - Read terminated by framing error (final character in buffer is character on which error occurred)
- success - Read terminated by parity error (final character in buffer is character on which error occurred)
- success - Read terminated by receiver overrun (final character in buffer is character on which error occurred)
- failure - no Read request outstanding
- failure - Read not complete
- BUFFER is a returned buffer (returned only on success)
- LOW-WATER is the updated value of the corresponding parameter from the READ function (returned only on success).
- MORE-DATA-FLAG is a Boolean flag indicating if there was more data in the type-ahead buffer when the Read terminated.
- VERTICAL-POSITION is the relative vertical position change (number of lines) on the presentation device since the beginning of this Read.
- HORIZONTAL-POSITION is the relative horizontal position change on the presentation device since the beginning of this Read.
- TERMINATOR-POSITION is the number of data characters (excluding terminators, escape sequences, etc.) in the buffer. Thus, if there is a terminator or escape sequence in the buffer, it points to the terminator or the first character of the escape sequence.

NOTE

The prompt from the READ function is not returned by the READ-POLL function.

3.1.2 Reading Out-of-band Characters -

READ-OUT-OF-BAND (PORTAL-ID; RETURN, OUT-OF-BAND-CHARACTER)

requests the return of a pending out-of-band character.

- RETURN is one of the following:
 - success - out-of-band character returned
 - success - "line break" occurred; no out-of-band character returned
 - failure - no out-of-band character pending

3.1.3 Writing Characters - The WRITE function is used to send characters to the logical terminal. It may also be used to control the terminal without sending characters to it. In particular, the DON'T-DISCARD-FLAG has meaning in the WRITE function even if no data is written.

```
WRITE (PORTAL-ID, DON'T-DISCARD-FLAG, NEWLINE-FLAG [,LOCKING]
      [,TRANSPARENT-FLAG] [,COMPLETION-STATUS-FLAG] [,BUFFER]
      [,PREFIX-CODE, PREFIX-VALUE] [,POSTFIX-CODE, POSTFIX-VALUE];
      RETURN [,REQUEST-ID])
```

requests several functions including writing data to the terminal.

- DON'T-DISCARD-FLAG is a Boolean value that, if true, sets the output discard state of the handler to "not discarding" before the output data is processed (see READ-DISCARD-STATE function in Section 3.1.4).
- NEWLINE-FLAG is a Boolean value that, if true, directs the logical terminal to output a linefeed at the end of a Write and set an internal "skip linefeed" flag so that if the first character in the next Write is a linefeed, double spacing is avoided.
- LOCKING is a value that defines how locking is to be handled for the duration of this Write. The values are as follows:
 - 0 - unlock
 - 1 - lock before writing data; do not unlock after writing data
 - 2 - lock before writing data and unlock at end of data; do not redisplay
 - 3 - lock before writing data and unlock at end of data; redisplay input buffer after unlocking.

- TRANSPARENT-FLAG is a Boolean value that, if true, causes the data in this Write request to be written to the Foundation Services using the "transparent" Write-character function. "Transparent" Write does not expand tabs or wrap, and resets HPOS and VPOS (position modeling) to zero.
- COMPLETION-STATUS-FLAG is a Boolean value that, if true, causes the handler to return a Write completion status via the WRITE-POLL function.
- BUFFER is a buffer containing the data to be sent to the terminal. This argument is optional because the functions requested by setting the other arguments true may be requested without requesting data output. The WRITE function is modeled as copying the data from this buffer immediately to a buffer internal to the handler (an implementation may, of course, not operate this way).
- PREFIX-CODE defines whether or not data should be prefixed to the output and, if so, how PREFIX-VALUE should be interpreted. It is one of the following:
 - NONE - don't prefix the output data
 - NEW-LINES - prefix the output data with <carriage return> followed by the number of <line feed>'s specified in PREFIX-VALUE
 - CHARACTER - prefix the output data with the character specified in PREFIX-VALUE
- PREFIX-VALUE is either a number or a character, as defined by PREFIX-CODE.
- POSTFIX-CODE is interpreted exactly as PREFIX-CODE, except that it applies to postfixed data.
- POSTFIX-VALUE is interpreted exactly as PREFIX-VALUE.
- RETURN is one of the following:
 - success - request queued
 - failure - insufficient resources
- REQUEST-ID is a value returned only if COMPLETION-STATUS-FLAG is true. It is a handle for executing the WRITE-POLL function.

The following function is required only if the WRITE function requests completion status.

WRITE-POLL (PORTAL-ID, REQUEST-ID; RETURN, HORIZONTAL-POSITION,
VERTICAL-POSITION)

polls to get back status of a previously queued Write request that requested completion status.

- REQUEST-ID is a value returned from a previous Write request
- RETURN is one of the following:
 - success - Write complete, no data discarded
 - success - Write complete, data discarded due to control-O action by user
 - failure - no outstanding Write request for REQUEST-ID
 - failure - Write request still queued
- HORIZONTAL-POSITION is the relative horizontal position change of the cursor on the presentation device at the completion of the Write.
- VERTICAL-POSITION is the relative vertical position change of the cursor on the presentation device at the completion of the Write.

3.1.4 Control and Status Functions -

CLEAR-INPUT (PORTAL-ID; RETURN)

queues a request to clear the type-ahead and input buffers.

- RETURN is one of the following:
 - success - request queued
 - failure - insufficient resources

If input escape sequence recognition is enabled, this resets the input escape sequence state machine.

CHECK-INPUT (PORTAL-ID; RETURN)

queues a request to return the number of characters in the type-ahead and input buffers (combined). The requested value is returned via the CHECK-INPUT-POLL function. No more than one CHECK-INPUT request may be queued at a time.

- RETURN is one of the following:
 - success - request queued
 - failure - CHECK-INPUT request currently queued

CHECK-INPUT-POLL (PORTAL-ID; RETURN, COUNT)

returns the character count requested via a previous CHECK-INPUT function. The value returned is the value at some point in the past between execution of the CHECK-INPUT request and the completion of this function.

- RETURN is one of the following:
 - success - COUNT returned
 - failure - CHECK-INPUT request queued but not complete
 - failure - CHECK-INPUT request not queued
- COUNT -- the requested count (returned only on success)

READ-INPUT-STATE (PORTAL-ID; INPUT-STATE)

returns the input state of the terminal at some point in the past.

- INPUT-STATE indicates whether or not the number of characters in the input and type-ahead buffers (combined) is zero. It is one of the following:
 - ZERO
 - NON-ZERO

READ-DISCARD-STATE (PORTAL-ID; DISCARD-STATE)

reads the discard state of the terminal at some point in the past.

- DISCARD-STATE is a value indicating if output to the terminal is being discarded by the handler (due to the entry of a control-O). It is one of the following:
 - DISCARDING - output is being discarded
 - NOT-DISCARDING - output is not being discarded

The state may be set to NOT-DISCARDING via the WRITE function.

3.1.5 Reading and Writing Characteristics -

WRITE-CHARACTERISTIC (PORTAL-ID, (SELECTOR,VALUE) [,...]; RETURN)

queues a request to write a collection of selected characteristics.

- SELECTOR is a characteristic selector (e.g. "RAISE-INPUT").
- VALUE is the new value for the specified characteristic.

- RETURN is one of the following:

success

failure - insufficient resources

The WRITE-CHARACTERISTIC function allows the operating system to write both Foundation-maintained characteristics and handler-maintained characteristics. While the latter can always be written by the operating system, the former can not always be. If the operating system attempts to write a Foundation-maintained characteristic while such writing is disabled (see the Network Virtual Foundation Services specification), the Write will fail, but the operating system will be given no explicit indication that the Write failed. The operating system may ascertain that the Write failed by reading the characteristic(s) back via the READ-CHARACTERISTIC function to see if they changed.

READ-CHARACTERISTIC (PORTAL-ID, (SELECTOR,VALUE) [,...]; RETURN)

requests the return of the values of the selected characteristics. The VALUE (the information being requested) parameter will in general be "null" except in the case of characteristics with compound values where part of the value may be used to further define (i.e. help select) the characteristic whose value is being requested. See section "Selector Values for Characteristics" in the OPERATION section for further details of compound characteristic values. The values are returned by the READ-CHARACTERISTIC-POLL function.

- SELECTOR is a characteristic selector.

- RETURN is one of the following:

success - request queued

failure - previous request outstanding

failure - insufficient resources

READ-CHARACTERISTIC-POLL (PORTAL-ID; RETURN, (SELECTOR, VALUE) [,...])

returns the values of characteristics previously requested by the READ-CHARACTERISTIC function.

- RETURN is one of the following:

success - VALUEs returned

failure - no request queued

failure - READ-CHARACTERISTIC request still queued

- SELECTOR is a characteristic selector.

- VALUE is the value of the corresponding characteristic (returned on success).

3.2 Server Terminal Interface

The server terminal interface contains functions at two levels. At one level, this interface allows the network command terminal module in the server system to Read and Write characters, to detect mode changes, and to Read and Write characteristics. These interface functions are described in the Foundation Services specification.

At the second level, this interface contains functions perceived by the human terminal user, as described generally in Section 2, Network Command Terminal Overview. The quoting, output control, and input editing control functions are described more fully below.

3.2.1 Quoting - If quoting is enabled (via a characteristic) and a control-V character is entered at the terminal, the character following the control-V is not recognized as a control character. It is not acted upon as an output control character, an input editing character, an out-of-band character, or a termination character.

The control-V and following character are treated as a single character (a token). Therefore, entering a DEL character after such a pair causes the pair to be deleted.

The control-V is passed to the operating system on read completion. If the complete token will not fit in the current input buffer, the READ is terminated (status -- success, terminated by absentee token) and the host will have to post another READ to get the token plus any subsequent input (a quoted character token does not terminate a READ).

3.2.2 Output Control - If output discarding is enabled (via a characteristic) and a control-O character is entered at the terminal, the discard state of the terminal (read by the operating system via a READ-DISCARD-STATE function) is toggled. That is, if its current value is "discarding", it is set to "not discarding" and vice versa.

3.2.3 Input Editing - There are five input editing functions:

- o redisplay input
- o delete character
- o delete word
- o clear input
- o clear type-ahead

The invoking of input editing and the effect of the server's input editing algorithms on the presentation device are summarized below. The details of input editing are described later in Section 4, OPERATION.

3.2.3.1 **Redisplay Input** - The redisplay input function is invoked by the entry of a control-R character. It redisplay the prompt from the READ function (if any) concatenated with the contents of the input buffer on the line after the current one.

3.2.3.2 **Delete Character** - The delete character function is invoked by the entry of a DEL character. If underflow occurs and is a termination condition, the editing is handled by the host. Otherwise, the last character of the input buffer is deleted. If the character has been echoed, its echo representation is deleted from the presentation device as follows:

- o Hard-copy terminals -- if the character is the first character in a row to be deleted, a backslash (\) is output. The character is then echoed as it originally was. When the first non-delete character is entered after multiple DEL characters, an ending backslash is output.
- o Softcopy terminals -- each character in the echo representation is deleted separately, from the last to the first. In general, these echoed characters are removed directly from the screen.

3.2.3.3 **Delete Word** - The delete word function is invoked by the entry of a control-W character. It deletes the "word" at the end of the input buffer. A word is defined as consisting of a string of consecutive alphanumeric characters followed by a string of non-alphanumeric characters or the end of the input buffer. A word is preceded by either a non-alphanumeric character or the beginning of the input buffer. The effect of the delete word function when applied to other than a word, as defined above, is to delete all characters from the input buffer.

"Unechoing" for delete word is handled as though a DEL had been issued for each character being deleted, i.e. for soft-copy terminals, each character is blanked out -- for hard-copy terminals, each character deleted is re-echoed using the \xxx\ figure.

3.2.3.4 **Clear Input** - The clear input function is invoked by the entry of a control-U character. It clears the input buffer, the control character is echoed and the prompt redisplayed on the next line.

3.2.3.5 **Clear Type-ahead** - The clear type-ahead function is invoked by the entry of a control-X character. It clears the type-ahead buffer and then places a control-U in the type-ahead buffer if there is an active READ.

3.3 Terminal Characteristics

An operating system has access to several characteristics via the WRITE-CHARACTERISTICS and READ-CHARACTERISTICS functions. Some characteristics that may be read and written via these functions are maintained by the Foundation Services and are described in the Foundation Services specification. Foundation-maintained characteristics are maintained across bindings.

Other characteristics are maintained by the handler itself, and are not maintained across bindings. They are summarized below.

Each description of the characteristics maintained by the handler includes the initial value of the characteristic when a binding is first formed. Each characteristic is one of the following types:

- o Boolean -- takes TRUE and FALSE values.
- o Integer -- signed integer; bit width is 16 bits unless specified otherwise.
- o Bit Map -- a string of bits, each having a separately defined meaning.
- o Compound -- a value which consists of more than one field.

The characteristics are defined below.

- o IGNORE-INPUT -- causes all input from the server terminal interface to be discarded without processing
 - Boolean (initial value is FALSE)
 - If TRUE, the handler discards all input.
 - If FALSE, the handler processes all input.
- o CHARACTER-ATTRIBUTES -- specifies, on a per character basis, the attributes of the character. The format of this compound characteristic is defined in subsection 4.17.2.1.

CHARACTER-ATTRIBUTES defines the following attributes for each character:

- Out-of-band handling -- specifies whether a particular character is an out-of-band character and the type of out-of-band character it is. If a character is an out-of-band character, it is passed to the operating system independently from other input characters. The actions specified for the out-of-band characters should be performed in the order specified so the order of arrival of out-of-band messages and Read messages at the host will be predictable.

(a) Not out-of-band character

- (b) Immediate-clear out-of-band character. The character is placed in the out-of-band buffer. The type-ahead buffer is then cleared (regardless of the enabled or disabled state of control-X) and the current Read (if one is outstanding) is terminated.

- (c) Deferred-clear out-of-band character. This is actually a double control character (two consecutive, identical, control characters). If only a single control character of this type, it is treated as though it were not an out-of-band character. A deferred-clear out-of-band character produces the same effect as an immediate-clear out-of-band character except that only one of the two characters is placed in the out-of-band buffer for transmission to the operating system.
 - (d) Immediate-hello out-of-band character. This character is passed to the operating system independently of other input characters, but it does not clear the type-ahead buffer or terminate the current Read as with the "clear" out-of-band characters.
- Include flag -- indicates whether a copy of a hello out-of-band character should be included in the normal data stream.
 - Out-of-band discard flag -- indicates whether or not an entered clear out-of-band character sets the output discard state to "discard".
 - Control character echoing -- specifies how a control character or DEL should be echoed.
 - Disable/enable special character function -- specifies whether the special functions associated with certain well known control characters are enabled or disabled.
- o CONTROL-O-PASS-THROUGH -- specifies whether control-O characters are passed through as input data characters when enabled as a control character.
 - Boolean (initial value is FALSE)
 - if TRUE, the control-O is passed through as input data in addition to performing its control functions.
 - if FALSE, the control-O is not passed through as data.
 - o RAISE-INPUT -- specifies whether the lowercase alphabetic characters are to be converted to uppercase on input.
 - Boolean (initial value is FALSE).
 - If TRUE, the characters whose decimal values are 97 --> 122, inclusive, have their values decremented by 32 (decimal). This is done by the input process as characters are moved from the type-ahead buffer to the input buffer.
 - If FALSE, no conversion is done.

- o NORMAL-ECHO -- specifies how characters other than control characters and DEL are echoed.
 - Boolean (initial value is TRUE)
 - If TRUE, echo the characters.
 - If FALSE, don't echo the characters.
- o INPUT-ESCAPE-SEQUENCE-RECOGNITION-ENABLE -- specifies whether input escape sequences are recognized.
 - Boolean (initial value TRUE).
 - If TRUE, input escape sequences are recognized and processed as described in Section 4.10.1, Input Escape Sequences.
 - If FALSE, input escape sequences are not recognized and the characters in the escape sequence are treated as normal data.
- o OUTPUT-ESCAPE-SEQUENCE-RECOGNITION-ENABLE -- specifies whether output escape sequences are recognized.
 - Boolean (initial value TRUE).
 - If TRUE, output escape sequences are recognized and processed as described in Section 4.12.3, Output Escape Sequences.
 - If FALSE, output escape sequences are not recognized and the characters in the escape sequence are treated as normal data.
- o INPUT-COUNT-STATE -- defines how the automatic transmission of input state messages is to be handled.
 - Integer (initial value DO-NOT-SEND)
 - DO-NOT-SEND -- do not automatically send input state messages when the number of input characters changes between zero and non-zero.
 - NO-READ-SEND -- the input state message is automatically sent only when there is no outstanding READ.
 - SEND -- the input state message is automatically sent when the combined input buffer and type-ahead count changes between zero and non-zero except when a Read terminates (in which case, the same information is sent in the Read Data message).

- o AUTO-PROMPT -- specifies whether a control-A is sent to the terminal after the prompt, if any, before the Read.
 - Boolean (initial value is FALSE).
 - If TRUE, a control-A is sent.
 - If FALSE, a control-A is not sent.
- o ERROR-PROCESSING -- specifies whether the following types of error (received together with an input character from the logical terminal service) are ignored by the pre-input process (i.e., the pre-input process discards the error and character so the input process never sees either the error or the input character) or are queued in the type-ahead buffer (together with the character on which the error occurred). Characters received with one of the following error indications are not processed according to the normal precedence rules specified for the pre-input process (see Section 2.5.1). Instead, they are always processed with the lowest precedence (so an error can't accidentally cause something dreadful to happen, e.g., a parity error turn an ordinary X into a control-X):
 - Bit map with one bit for each type of error. The values for each bit are "ignore" (0) or "queue" (1) (initial value for each error type is "ignore").
 - Error types are:
 - line break
 - framing error
 - parity error
 - receiver overrun

3.4 Termination Set

The termination set is a 256 bit map whose initial value is zero. There is one bit for each of the characters whose value is in the range 0-255. A character is a termination character if its corresponding bit is set.

4.0 OPERATION

This chapter considers the handler as a distributed system. Figure 4-1 shows the handler's structure.

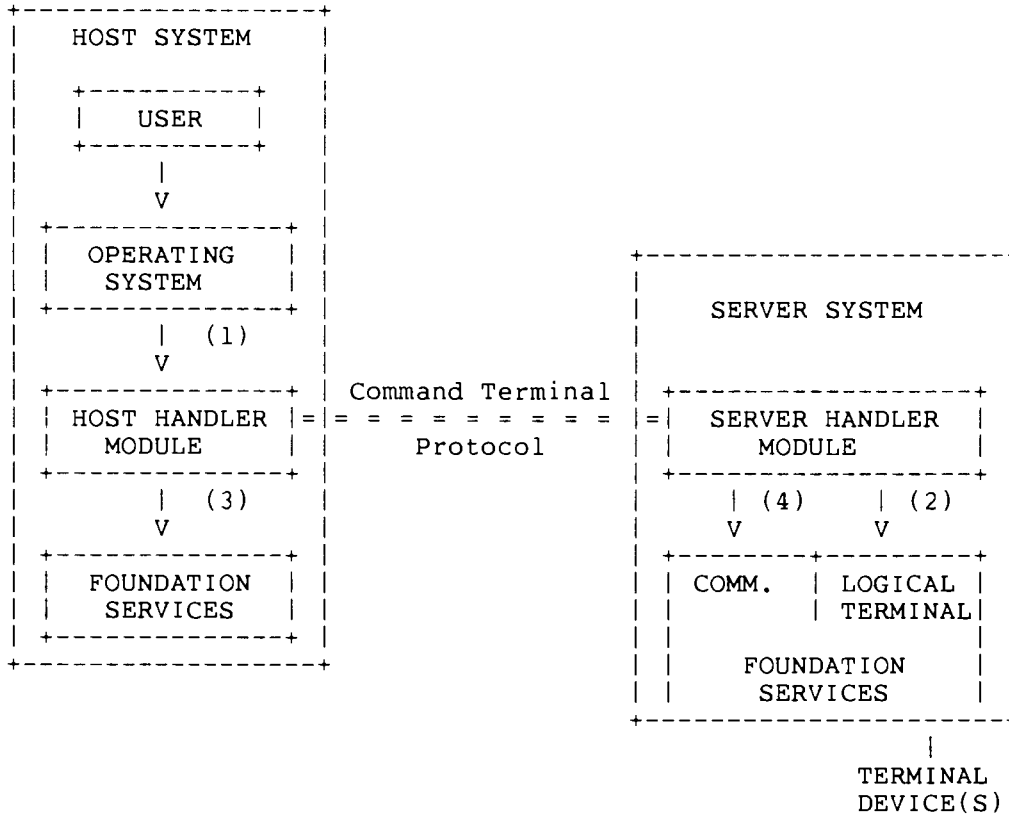


Figure 4-1 Structure of Distributed Terminal Handler

In Figure 4-1, interfaces (1) and (2) are equivalent to the interfaces of the same number in Figure 2-1. Interfaces (3) and (4) allow the two modules constituting the handler to make use of a binding between them to transmit and receive command terminal protocol messages. The functions of these interfaces (3 and 4) are described in the Foundation Services specification.

In the following descriptions of interfaces (1) and (2), the part of the handler residing in the host system is referred to as the host module, or more simply the host. Similarly, the part of the handler residing in the server system is referred to as the server module or the server.

4.1 Interfaces and Protocols

Examination of the interfaces (only interfaces (1) and (2) -- interfaces (3) and (4) are operating at a lower level and are not relevant for purposes of this discussion) specified in Section 3 and the protocol specified in this section will show that the functions offered are not identical. The protocol, for example, supports terminating a READ on a vertical change in the echoed image of input data, a function which is not available at interface (1).

This difference between the functions of the interfaces and the protocol is a result of the relation between the two. It is the interfaces that are of primary importance and the protocol is just a means to support these interfaces in a distributed environment. Standardized conceptual interfaces, such as (1) and (2), are one of the key factors in distributed processing. While the protocols which are used to support these interfaces are, in a certain sense, more visible than the conceptual interfaces themselves, it is important not to get their roles reversed. The idea of a layered architecture, used in both DNA and TSA, is based on conceptual interfaces.

4.2 Host/Server Division of Labor

This specification defines most of the distributed terminal handler functions to be performed by the server. The reason for this is that the goal of reasonable performance implies the need to handle user input editing requests close to the user. This, in turn, implies that the input process (and, therefore, the pre-input process and output procedure) of Figure 2-2 must reside in the server.

The host module primarily converts host terminal interface requests to protocol messages and vice versa. The host module is, therefore, not discussed in detail below.

Much of the algorithmic operation of the server is implied by the descriptions in Section 2. The descriptions below do not restate the operation implied by Section 2 but add information regarding protocol operation; the relation to the Foundation Services interface and; in some cases, details of algorithmic operation not found in Section 2.

4.3 Data Channels

The host and server modules communicate via a binding provided by the Foundation Services.

4.4 Protocol Message Overview

The message types in the command terminal protocol are summarized below.

Message -----	Direction -----	Definition -----
Initiate	H <--> S	carries initialization information
Start Read	H ---> S	carries READ request
Read Data	H <--- S	carries input data
Out-of-band	H <--- S	carries out-of-band input data
Unread	H ---> S	carries UNREAD request
Clear Input	H ---> S	carries Clear Input request
Write	H ---> S	carries WRITE request
Write Complete	H <--- S	carries write completion status
Discard State	H <--- S	carries a change to the output discard state due to a terminal user request (via an entered output-discard character)
Read Characteristics	H ---> S	requests characteristics
Characteristics	H <--> S	carries characteristics
Check Input	H ---> S	requests input count
Input Count	H <--- S	carries input count
Input State	H <--- S	indicates a change from zero to non-zero or vice versa in the number of characters in the input and type-ahead buffers combined

4.5 General Message Processing

The network command terminal protocol is a request/response protocol. That is, the host module sends one, or several related messages, to the server module in response to a single queued request (as described in the host terminal interface) from the operating system. The server module, in turn, sends one, or several related messages, to the host module in response. The response messages are not necessarily sent immediately or in a predictable order, but they are eventually sent. The server never sends data to the host except in response to a previously received Read request (conveyed to the server via a Start Read message).

The server receives all protocol messages on a single data channel of the underlying binding. These messages are generally processed to completion in the order they are received. This ensures that functions that the operating system expects to have executed sequentially are so handled.

4.6 Protocol Errors

A protocol error occurs when a received protocol message cannot be interpreted in a way that will ensure secure and synchronized operation. An example is when a server receives a second Start Read message without having completed the previous Read request. Unless otherwise specified, the occurrence of non-zero values in unused or reserved fields and 1's in unused or reserved bits in bitmaps can be ignored. All other errors in received messages constitute a protocol error with the single exception of the Initiate message -- see Section 4.15, Protocol Evolution, for a statement of protocol compatibility.

A module detecting a protocol error breaks the connection over which the protocol was operating. This process, known as unbinding, is defined in the Foundation Services specification.

4.7 Initialization

When a pair of host and server modules first communicate over a given binding, each sends an Initiate message to the other. It is a protocol error for a module to receive any other message when the binding is first used. Similarly, it is a protocol error to receive the Initiate message at any other time.

The Initiate message contains a version identification. An implementation of this version (1.0.0 in the protocol) assumes it is compatible with any future (higher-numbered) version; it is not a protocol error to receive a version number higher than 1.0.0.

The Initiate message is designed for easy extension; any parameter which either module receives which it does not understand is ignored.

4.8 Characteristics Management

The host sends a Read Characteristics message to elicit a Characteristics message from the server. If the operating system changes the characteristics when there is no Read request pending, the terminal characteristics will be as specified for subsequently issued Reads and Writes.

It is a protocol error if an invalid selector or value is specified in a Read Characteristics or Characteristics message.

4.9 Read Request Processing

The sections that follow discuss:

- o Issuing the Read
- o Unreading
- o Position modeling
- o Read completion
- o Input editing

4.9.1 Issuing the Read - When the host terminal interface Read function is executed, the host sends a Start Read message. This message contains the parameters from the Read function. The portion of data in the host's buffer after START-OF-DISPLAY is sent in the DATA field of the Start Read message. It is assumed that the Start Read protocol message size will be large enough to contain all the data to be sent to the server. If formatting has been selected (the FORMATTING-FLAG on the Read) and the last character output was a CR, then a LF is output (to avoid overprinting); in addition, if the first character of the preloaded input would echo as a LF, it is ignored (to avoid accidental double spacing).

If the Start Read contains a prompt, the input process writes out the characters in the prompt without performing any echo translation on these characters. Next, the input process invokes the foundation services RESET PAGE-STOP-POSITION function. If the Output-page-stop characteristic is true, this function causes the Foundation page-stop-position variable to be set 0; invoking this function when a Read is issued, allows the foundation "Output page stop" algorithm to produce the correct visual effect. If the output-page-stop characteristic is false, this function has no effect.

If the AUTO_PROMPT characteristic is TRUE, before the data in the Start Read's DATA field (the prompt) is echoed, a control-A is sent to the terminal (this is usually used by automatic input devices to determine when a Read is posted).

If a server receives a Start Read message while it has a Read pending, it is a protocol error.

4.9.2 Unreading - If a host wants to terminate a Read request it issued earlier, it may send an Unread message to the server. An Unread message terminates the current Read request (if any) when the Unread message is processed in the server. This is considered a termination condition and the Read request is completed as described below.

A host may specify that only an outstanding Read for which the input buffer is empty and the type-ahead buffer is also empty will be terminated.

If an Unread terminates a pending Read with a non-empty input and/or type-ahead buffer, the input escape sequence state machine is reset if input escape sequence recognition is enabled.

Since a Read completion and an Unread message may cross between the host and the server, an Unread received by the server when no Read is outstanding, is not a protocol error, and is ignored.

4.9.3 Position Modeling - The input editing functions in the server require knowledge of the active horizontal output position. The active horizontal and vertical positions are maintained by the Foundation Services and are available at the server terminal interface.

4.9.4 Read Completion - When a termination condition occurs in the server (including processing an Unread message), the current Read request is completed. All input data is sent to the host via a Read Data message. The maximum protocol message size must be selected to be large enough to contain both the Read message protocol header and the whole input buffer. If there was a prompt in the Start Read message, this prompt is not returned to the host in the Read Data message (the prompt is non-deletable and the host is assumed to know what it was in the event it is required).

4.9.5 Input Editing - Input editing can be handled in either of the following ways:

- o entirely within the server system (local) provided sufficient resources are available in the server to contain the entire input buffer specified with the Read request at interface (1), or
- o distributed between the server and host systems (as described below) when either
 - a. the server does not have sufficient resources to contain the complete input buffer, or
 - b. the host wants to provide more elegant handling of the presentation of the echoed image of the human user's input.

The server system's input editing algorithms do not assume the existence of "up-line" and "delete-line" functions on the presentation device and therefore input editing functions such as redisplay and clear input can not blank-out the image of the currently active input line on the screen or redisplay the input buffer in place. Instead, they will echo the input editing control characters, issue a CR, LF to advance to the next line on the presentation device, and then redisplay the edited input buffer. Hosts requiring a more elegant treatment of the image on the presentation device should use distributed input editing.

4.9.5.1 Distributed Input Editing - With distributed input editing, some of the input editing functions are handled in the server and the remainder in the host. Typically, the input editing operations that involve only the current physical line on the presentation device are handled in the server, e.g., delete character and delete word. Operations that involve more than just one line on the presentation device are handled in the host when distributed input editing is selected; that is, a delete around a line wrap. Also, operations that involve redisplaying in place or blanking-out one or more lines are handled in the host.

The host chooses whether to use local or distributed input editing. If a host chooses to implement and use distributed input editing, the host module must provide procedures to handle input editing functions.

4.9.5.2 Selecting Distributed Input Editing - The host selects distributed input editing by setting the following characteristics and Start Read message flags:

1. setting control-R and control-U to be termination characters rather than enabled input editing characters -- this causes the handling of redisplay and clear input to be passed to the host which can handle multiple line deletes and redisplay in place.
2. setting the Start Read message continuation-read flag TRUE when the host already has a portion of the input from the Read function -- this causes delete character and delete word to be passed to the host when the server can not handle the operation because the word or character to be deleted has already been sent to the host.
3. setting the Start Read message terminate-on-vertical-change flag -- this avoids the wrap problem by ensuring the server never has more data than can be contained on one line on the presentation device.

The effect of these characteristic and flag settings is to limit the server, for distributed input editing, to handling only character and word deletions where the item to be deleted is entirely on one line on the presentation device.

4.10 Other Input Processing

In addition to normal input processing, the server handles escape sequence recognition, case raising for input characters, input processing for out-of-band, control-V, control-X, control-O characters and errors on input.

4.10.1 Input Escape Sequences - Input Escape sequence recognition is enabled or disabled through the INPUT-ESCAPE-SEQUENCE-RECOGNITION-ENABLE characteristic and RECOGNIZE-INPUT-ESCAPE-SEQUENCE flag on the Read function. When enabled, input escape sequence handling is as follows:

1. Input escape sequences are parsed by a state machine in the input process and terminate the current Read.
2. Handling of the escape sequence as a whole (token) takes precedence over the processing of any individual character within the escape sequence which may be defined as a terminator.
3. Input escape sequences do not echo.
4. If an input clear type-ahead function (control-X) is invoked simultaneously with the parsing of an escape sequence, this will result in the Read being terminated with an "invalid escape sequence" status (the control-X will clear the type-ahead buffer) and the input escape sequence state machine is reset (the control-U put into the type-ahead buffer after clearing it will not be processed as part of an escape sequence).
5. The input escape sequence state machine is reset as a result of terminating a Read due to either a Clear-input or Unread from the host.
6. An input escape sequence is a token and will not be split across two or more Reads and remain valid. Buffering of the escape sequence is handled as follows:
 - o If the escape sequence can be contained in the current Read buffer, it is included with and terminates that Read.
 - o If the escape sequence won't fit in the current Read buffer and the Read buffer was non-empty before the start of the escape sequence, the current Read is terminated with a status of "terminated by absentee token". Normally, the host will post another Read to get the escape sequence.
 - o If the escape sequence goes into an empty Read buffer and is completely contained, the Read is terminated with a "valid escape sequence" status.

- o If the escape sequence goes into an empty Read buffer but is bigger than the buffer, the Read is terminated with an "invalid escape sequence" status after filling the buffer with characters from the escape sequence. The host may get the remaining characters from the escape sequence by posting another Read for them -- they are returned as normal data as the input process is no longer in the escape sequence state.

4.10.2 Raising Input - The host may request that the server raise the case of input characters by setting the RAISE-INPUT characteristic true. When this is done, the server examines every character as it moves it from the type-ahead buffer to the input buffer to see if it is a lowercase alphabetic character (in the range 97-122, decimal). If so, it is converted to the uppercase equivalent (by subtracting 32, decimal). This processing is done by the input process. The effect of this characteristic may be overridden for individual Read's by the RAISE-INPUT flag on the Read.

4.10.3 Out-of-band Processing - When an enabled out-of-band character or "line break" is recognized by the server, it is sent to the host via an out-of-band message. If it is a "clear" out-of-band character, the type-ahead buffer is cleared, and if a Read is pending, the Read is terminated. A copy of each "hello" out-of-band character is included in the normal data stream if the HELLO-OUT-OF-BAND-PASS-THROUGH characteristic is true. For those cases where out-of-band processing results in transmitting both a Out-of-band message and a Read Data message, the out-of-band message is always transmitted first.

4.10.4 Control-V - When the user enters a control-V, the character after the control-V is ignored as either a control character, an out-of-band character, or a termination character. The control-V is buffered as a normal data character.

4.10.5 Control-X - When the user enters a control-X character, the server clears the type-ahead buffer. If there is a Read active, the server also executes a clear input function, the latter being the same function as if the user had entered a control-U.

4.10.6 Control-O - The control-O character affects output discard handling, discussed in Section 4.12 below.

4.10.7 **Errors on Input** - Input characters from the Foundation Services can have four errors associated with them, i.e., line break, framing error, parity error, and receiver overrun error. The pre-input process, depending on the value of the characteristic ERROR-PROCESSING, may either discard these error characters (in which case the error is ignored as the input process never sees the error or the character on which it occurred) or it queues the character and the error along with normal input in the type-ahead buffer for the input process.

When the input process gets one of these errors, it terminates the current Read. If a Read is not active in the server, the error condition terminates the next Read that would normally include this character. Any characters preceding the error are returned as normal; the character on which the error occurred is the last character in the buffer returned when the Read is terminated. The character on which the error occurred is always returned, even if it is the only character returned with the terminated Read (and is probably garbage). What happens to the characters in the type-ahead buffer after the character on which the error occurred is implementation dependent.

4.11 **Write Request Processing**

A host sends output to the server via the Write message. The server may queue the information from a Write message. If it does, it also queues the FLAG field information. The Write messages are processed to completion in the order received.

Due to restrictions in the maximum length of a protocol message that the server can receive, the information from a single Write request by the operating system may be carried in multiple Write messages. When this occurs, the first such message contains all flags from the WRITE request, and the "beginning of message" flag in the message is set. Write messages containing intermediate data have the "beginning of message" and "end of message" flags cleared. The Write message containing the final data has the "end of message" set. (A single Write message containing all data from a single Write request has both the "beginning of message" and "end of message" flags set.) All Write message flags are ignored except when "beginning of message" is set.

It is a protocol error for a server to receive Write messages with the "beginning of message" and "end of message" flags set incorrectly. The following are protocol errors:

- o receiving the first Write message on the binding with "beginning of message" clear
- o receiving a Write message with "end of message" set followed by a message with "beginning of message" clear
- o receiving a message with "end of message" clear followed by a message with "beginning of message" set (there is an exception to this when the output discard state is toggled, as described in Section 4.12.1)

In the descriptions below, a "message" means the flags and data from a single Write request, regardless of the number of Write messages used to convey the request.

If a message has "don't discard" flagged, the output discard state is set to "don't discard" before the message data are written. If a message has "completion status requested" flagged, then the server sends a Write Completion message after writing the message data.

The Write message contains information regarding prefixing and postfixing data to the message data. The "prefix code" and "postfix code" flags values and the PREFIX-VALUE and POSTFIX-VALUE fields contain the information required to prefix and postfix the message data.

A host may send a Write message without data to effect control. In this case, the "beginning of message" and "end of message" flags are both set, and the Write message does not contain the DATA field. The rules stated above for "beginning of message" and "end of message" flags also apply. This means that the host cannot interrupt the sending of a sequence of Write messages containing output data to send control data.

4.12 Other Output Processing

In addition to writing data, the server handles the output discard state, locking, and output escape sequences. These are described below.

4.12.1 Output Discard State Handling - The server maintains two state variables used for output discard processing: one (the "requested state") that reflects what the user or operating system most recently requested, and one (the "real state") that represents how the server is currently processing output. Each state variable may take on the "discard" or "don't discard" value. The initial state of each is "don't discard" when a binding is first used.

When the user enters a control-O character, the requested state is toggled, and a Discard State message is sent to the host containing the new requested state. When the Server receives a Write message with the "set output discard state" flag set, it sets the requested state to "don't discard".

When the requested state makes a change to "discard", the server sets the real state to "discard" also. In this state, the server discards the data from all received Write messages. When the server sets the real state to "discard", it also clears the output "lock" if one is in effect. However, the server continues to process the flags from Write messages with the exception of "unlock" and "redisplay" flags.

When the server sets the requested state to "don't discard" because of Write message flag processing, it also sets the real state to "don't discard"; it does not change the real state to "don't discard" as a result of a control-O from the terminal user.

The host maintains a single version of the output discard state. It sets this version to (1) the most recent value from a received Discard State message, (2) "don't discard" as the result of a WRITE request from the operating system specifying DON'T-DISCARD, or (3) "don't discard" as the result of a Read request from the operating system. (The host returns this version of the state on a READ-DISCARD-STATE function.) When the host sets the state to "don't discard" as the result of receiving a Discard State message containing the "don't discard" state or as the result of the operating system executing a WRITE with the DON'T-DISCARD-FLAG true, it sends a Write message to the server with the "set output discard state" flag set. When the discard state is set to "don't discard" as the result of a Read request from the operating system, the server sets the real discard state to "don't discard" when it receives the Start Read message generated as a result of the Read request.

While the host's state is "discard", it discards any data (but not flags) from operating system Write requests.

The result of the operation described above is that a change to the real "discard" state is always a result of the entry of a control-O by the user and is acted upon immediately by the server. A change to the real "don't discard" state is always caused by host action. A control-O from the user to change the state back to "don't discard" requires a round trip protocol exchange between the server and the host.

It is possible for the server to have a resource failure in attempting to send a Discard State message. If so, the server loops, trying to send the message. (This looping is done by the pre-input process.) Eventually, either the message is successfully sent, or data being entered by the terminal user are lost.

Because a change to the real "discard" state may take place in the middle of a multi-message write, a Write message with the "set discard output state" flag set is not considered a protocol error even if the last Write message received did not have the "end of message" flag set.

4.12.2 Locking - Locking resolves the potential priority conflict between output data from a WRITE request and output resulting from echoed input. When the server is not "locked", echo output has priority on a character-by-character basis. When the server is "locked", no echo output is accepted until the server is "unlocked". The one exception to this is that out-of-band control characters are echoed by the pre-input process without regard to whether the server is locked. Their echoing has priority over other output.

Locking and unlocking are controlled by parameters in the Write message. If a Write message requests "locking", echoing is locked out before the data are written. If a Write message requests "unlock" and optionally "redisplay", echoing is unlocked after the data are written and the input buffer is optionally redisplayed.

Once "locked", there are three ways in which the server can be "unlocked":

1. All data are written and the Write message requested "unlock".
2. A control-O is entered which sets the read-discard-state to "discard".
3. A "clear" out-of-band character is entered.

4.12.3 Output Escape Sequences - If output escape sequence recognition is enabled (by characteristic only), output escape sequences are written to the Foundation Services with the transparent Write. After writing the escape sequence, the horizontal and vertical position are both set zero.

If either OUTPUT-DISCARD or OUT-OF-BAND-DISCARD are enabled and a discard character is entered while writing an escape sequence, a "cancel" character must be written to the Foundation Services (to return the terminal to a reasonable state).

4.13 Additional Status and Control Operation

In addition to the operation described above, the server provides the host with the ability to read and write characteristics, to clear input, to request the number of characters in the combined input and type-ahead buffers, and to be informed when this number changes from zero to non-zero or vice-versa.

4.13.1 Reading and Writing Characteristics - The host writes one or more characteristics by sending a Characteristics message to the server. The length of this message is limited by the maximum length of a protocol message that the server can receive. If the host wishes to write more characteristics than will fit in a single Characteristics message, it sends multiple messages.

If the host wishes to read one or more characteristics, it sends one or more Read Characteristics messages to the server. The server responds by sending one or more Characteristics messages containing the requested information.

4.13.2 Clearing Input - The host sends a Clear Input message to the server, requesting it to clear all input. Upon receipt of this message, the server clears the type-ahead buffer and the input buffer (if a Read is active).

4.13.3 Input Character Count Handling - The host requests the number of characters in the type-ahead and input buffers combined by sending a Check Input message to the server. Upon receipt of this message, the server sends an Input Count message to the host.

In addition to this operation, under the control of the INPUT-COUNT-STATE characteristic, the server sends an Input State message to the host whenever the number of characters in the type-ahead and input buffers combined goes from zero to non-zero or non-zero to zero. This operation occurs without being specifically requested by the host. The characteristic selects when an Input State message is sent:

1. only when there is no outstanding READ, or
2. all the time except when a Read request completes and the server sends a Read Data message with "finished" set and a completion code, the "more type-ahead data" flag indicating whether or not there is data in the type-ahead buffer.

4.14 Foundation Services Interface Events

Two significant events occur at the interface between the Foundation Services and the handler module (either the host or the server module). These are: (1) the establishment of a binding and (2) the breaking of a binding (unbinding).

The host and server modules participate in binding and unbinding as defined in the Network Virtual Terminal Foundation Services specification.

When a binding is established, the state variables and characteristics of the host and server modules are initialized. Each sends an Initiate message to the other.

When a binding is broken, the server module clears the input and type-ahead buffers, loses knowledge of any pending read and temporary read states (e.g., "disable all control characters"), sets the output locking state variable to "unlock", and sets both the requested and real output discard states to "don't discard".

No interface activity is defined at the host terminal interface because it is assumed that the operating system can directly view the state of the binding.

NOTE

While the Foundation Connection Management and Mode Management functions are specified using a subroutine interface (i.e., a polling model which implies the mode modules must periodically poll, using the READ-PORTAL-BINDING-STATE, READ-LOGICAL-TERMINAL-BINDING-STATE, READ-PORTAL-MODE-STATE, and READ-LOGICAL-TERMINAL-MODE-STATE functions, to detect changes in state), actual implementations would probably use an event queuing mechanism as suggested in Host Terminal Interface, Section 2.1.

4.15 Protocol Evolution

Extensibility is a requirement of this specification. The philosophy guiding the operation of a system in meeting this goal is the following. Compatibility is the responsibility of the implementation using the higher version of the protocol. Compatibility is guaranteed only across one major version number (that part of the version number before the decimal point) of the protocol and means the higher version must use a subset of its protocol that is consistent with correct operation of the implementation using the lower numbered version of the protocol. Undefined fields, subfields and illegal values are protocol errors except for the Initiate message where they are ignored (so the version numbers can be exchanged).

4.16 Network Command Terminal Protocol Messages

The following notation is used to describe the protocol messages:

field (length) : coding = description of field.

where

field = the name of the field being described.

length = the length of the field, which can be indicated in one of three ways:

1. A number meaning number of 8-bit bytes (octet).
2. A number followed by a "B" meaning number of bits.

3. The letters "I-n" means this is an image field, with n being a number that specifies the maximum length of the field in 8-bit bytes. The image is preceded by a 1-byte count of the length of the remainder of the field. Image fields are variable in length and may be null (count=0). All 8 bits of each byte are used as information bits. The meaning and interpretation of each image field is as defined with that specific field.
4. "(length)" omitted -- the size of the field is not self defining. The field length is determined from the total size of the protocol message by subtracting the position of the start of the field from the total size of the message. (length) is omitted to circumvent the 255-byte limitation of the (I-n) specification. (length) can be omitted only where the field is the final field in the message.

coding = the representation type used,

where

A = 7-bit ASCII

B = binary

BM = a bit map of "length" bytes, which may contain subfields. A BM field is described as follows:

..AA .BBB depicts bits 0-7 in a byte (grouped for convenience into two hex-digit sections). The low-order bit is on the right. A dot (.) means that the corresponding bit is reserved; it must be transmitted as zero. One or more capital letters (e.g., AA) define a subfield whose width is equal to the number of identical letters. The values of each such subfield are defined independently. In this example, the AA subfield would have value 0-3 defined, and the BBB subfield would have values 0-7 defined.

C = a binary field containing the constant value shown to the right of the equal sign.

The following rules apply to the notation:

1. If length and coding are omitted, field represents a number of subfields specified in the description.
2. Any bit or field described as "reserved" shall be zero unless otherwise specified.

3. All fields are presented to the Terminal Communication Services with the least-significant byte first. In an ASCII field, the left-most character is contained in the low-order byte.
4. All numbers are in decimal unless otherwise specified.
5. Byte positions within protocol messages and fields within messages all start at zero (0).

4.16.1 **General Message Format** - All protocol messages have the following form:

MSGTYPE MSGDATA

where

MSGTYPE (1) : B = This field contains one of the following message types:

Value	Message
-----	-----
1	Initiate
2	Start Read
3	Read Data
4	Out-of-band
5	Unread
6	Clear Input
7	Write
8	Write Completion
9	Discard State
10	Read Characteristics
11	Characteristics
12	Check Input
13	Input Count
14	Input State
15	reserved for VMS
16	reserved for VMS
17	reserved for VMS

In the message descriptions below, the entire message format, including MSGDATA, is described for each message. For each message type, the value of MSGTYPE is considered a constant.

A "character pointer" is an integer defining the position of a character in a buffer. The first character in a buffer has position 0.

4.16.2 **Initiate (H <--> S)** - This message initiates command terminal protocol on a binding

MSGTYPE FLAGS VERSION PARAMETER MESSAGES

where

MSGTYPE (1) : C = 1

FLAGS (1) : C = 0

VERSION = A field identifying the protocol and software (implementation) version numbers. This field is subdivided as follows:

VERSION ECO MOD REVISION

where

VERSION(1) : B = protocol version number

ECO(1) : B = ECO number for this version of the protocol

MOD(1) : B = customer modification number

REVISION(8): A = software revision number

PARAMETER = A field containing a repeating parameter set (there will be as many instances of the parameter set as are necessary). The parameter set consists of two subfields as follows:

PARMTYPE VALUE

where

PARMTYPE(1): B = parameter type which specifies the purpose of the following VALUE subfield. It assumes the following values:

Value	Meaning
0	Illegal
1	The VALUE subfield (valid in both directions) specifies the maximum size protocol message the sender of the Initiate message can accept from its partner on the binding. As a lower bound, a server must be able to handle protocol messages of at least 132 (data) + 5 (CTERM header)

+ 2 (Foundation header) bytes = 139 bytes in length; a host must be able to handle protocol messages of at least 80 (data) + 8 (CTERM header) + 2 (Foundation header) bytes = 90 bytes in length.

- 2 The VALUE subfield is the maximum size (in bytes) input buffer that a server can support; a host can not request, in the MAX-LENGTH field of the Start Read message, a Read which exceeds this value. Servers must be able to handle at least an 80 byte input buffer. A host must be able to handle a protocol message of size input buffer size + 8 (CTERM header) + 2 (Foundation header) bytes = input buffer size + 10 bytes. Valid only in the server to host (H <--- S) direction.
- 3 The VALUE subfield is a bit map specifying which CTERM messages are supported by this implementation (the system transmitting this Initiate message). A system indicates a message type is supported by setting the corresponding bit in the bit map to one (1). The bits of the full bit map are numbered from 0 through 255. Bit 0 of the bit map is not used. The next 255 bits in the full bit map correspond to the 255 possible CTERM messages, where MSGTYPE = 1 corresponds to bit 1 of the full bit map,, and MSGTYPE = 255 corresponds to bit 255. The full bit map is contained in 32 bytes; this field is transmitted low-order byte to high-order byte. Within a byte, the bits are transmitted low-order bit to high-order bit. Where there are trailing zero bytes (as will usually be the case), the trailing zero bytes are not transmitted and the count is adjusted accordingly.

Support of the first 14 CTERM messages is mandatory. The TRG will maintain a registry of optional CTERM messages. Receipt of an unsupported message type is a protocol error.

VALUE(I-255) = value of specified parameter.

4.16.3 **Start Read (H ---> S)** - This message requests a read and describes an input buffer to the server.

MSGTYPE FLAGS MAX-LENGTH END-OF-DATA TIMEOUT
END-OF-PROMPT START-OF-DISPLAY LOW-WATER
TERMINATION-SET DATA

where

MSGTYPE (1) : C = 2

FLAGS (2) : BM = Flags:EE ZZQT NDDD IIKV FCUU

UU = underflow-handling definition
0: ignore underflow
1: write BEL to terminal
2: terminate

C = "clear type-ahead" flag
0: don't clear type-ahead
1: clear type-ahead

F = "formatting" flag
0: no special action
1: if last character output was CR,
output LF; also, if first
character of preloaded input is
LF, ignore it.

V = "terminate on vertical change" flag
0: do not terminate on vertical
position change
1: terminate read if there is a
vertical position change on the
presentation device while echoing
an input character

K = "continuation read" flag
0: this is not a continuation of a
previous read
1: this read is a continuation of a
previous read which terminated on
some condition other than that
specified for the user's Read
request. It indicates that input
data from the previous read is
buffered in the host. It is used
for distributed input editing; in
particular, delete character and
delete word processing

NOTE

When this flag has the value 1, the UU flag bits above must have the value 2 ("terminate" value).

- II = "raise input" flag
0: parameter not present in Read request -- use characteristic
1: no conversion for this Read only
2: for this Read only, convert lowercase alphas to uppercase
- DDD = disable control definition
0: parameter not present in Read request -- use characteristics
1: disable ^U and ^R only
2: disable all editing control characters
3: disable all control characters except XON and XOFF
- N = "no-echo" flag
0: echo according to characteristics
1: do not echo input characters (this read only) regardless of characteristics value
- T = "terminator echo" flag
0: do not echo terminator
1: echo terminator
- Q = TIME-OUT field present flag
0: TIME-OUT field not present in Read request -- the default is an infinite time-out, no time-out).
1: TIME-OUT field present in Read request
- ZZ = "non-default terminator set" flag
0: use terminator set specified by previous Read -- no terminator set specified and NON-DEFAULT-TERMINATOR-SET flag is TRUE
1: use terminator set specified by this Read
2: use universal termination set (i.e., all control characters except ^R, ^U, ^W, BS and HT)

EE = "recognize input escape sequences" flag
0: parameter not present in Read request -- use characteristic
1: do not perform input escape sequence recognition for this Read only
2: for this Read only, perform input escape sequence recognition

MAX-LENGTH (2) : B = Input buffer length, in characters.

END-OF-DATA (2) : B = Character position of the character after the last character currently in the buffer.

TIMEOUT (2) : B = Intercharacter arrival time in seconds. A timeout break condition will occur if another character does not arrive within this time period. A value of 0 means take the characters currently in the type-ahead buffer and terminate the Read immediately without waiting for further input.

END-OF-PROMPT (2) : B = Character position of first character beyond the Read-only (prompt) section of the buffer.

START-OF-DISPLAY(2) : B = Character position of the first character to be (re-)displayed.

LOW-WATER (2) : B = Same field as in Read Data (below). (Generally set to END-OF-DATA on a Start Read, but does not have to be).

TERMINATION-SET (I-32) : BM = The termination set for the Read. The full 256-bit bit-mask is contained in 32 bytes. However, if there are trailing zero bytes in the 32-byte bit-mask (as will generally be the case), trailing zero bytes are not transmitted. This field is transmitted low-order byte to high-order byte. Within a byte, the bits are transmitted low-order bit to high-order bit. Therefore, for example, the character whose value is 65 is a terminator for this read only if the 2nd bit (bits numbered 1 to 8) of the 9th byte (bytes numbered 1 to 32) is set.

DATA : A = Input buffer data

4.16.4 **Read Data (H <--- S)** - The server uses this message to transfer input data to the host.

MSGTYPE FLAGS LOW-WATER VERTICAL-POSITION
HORIZONTAL-POSITION TERMINATION-POSITION DATA

where

MSGTYPE (1) : C = 3

FLAGS (1) : BM = Flags: ...T CCCC

CCCC = completion code

0: termination character
1: valid escape sequence
2: invalid escape sequence
3: out-of-band character
4: input buffer full
5: timeout
6: unread
7: underflow
8: absentee token
9: vertical position change
10: line break
11: framing error
12: parity error
13: receiver over-run

NOTE

The completion code for an immediate timed read (i.e., zero timeout) will be "timeout" only if no other termination condition (e.g., "termination character") is encountered in processing the characters in the type-ahead buffer.

T = "more type-ahead data" flag

0: there is no data in the type-ahead buffer
1: there is data in the type-ahead buffer

LOW-WATER (2) : B = Character position of the last character that has not been modified on this Read.

VERTICAL-POSITION (1) : B = The relative vertical position change (number of lines) on the presentation device since the beginning of this read.

HORIZONTAL-POSITION

(1) : B = The relative horizontal position change on the presentation device since the beginning of this READ.

TERMINATION-POSITION

(2) : B = Number of DATA characters (excluding terminators, escape sequences, etc.) in the input buffer. Thus, when added to the address of the head of the buffer, it points to the character after the last data character in the buffer; if there is a terminator or escape sequence in the buffer, it points to the terminator or the first character of the escape sequence.

DATA : A = Input buffer data

NOTE

The VERTICAL and HORIZONTAL-POSITION parameters are undefined under the following circumstances:

1. Unread received by server.
2. Read with escape sequences embedded in either PROMPT or DATA fields.
3. Full duplex mode (i.e., unlocked Read Echos and Writes are taking place simultaneously).

The host resets its cursor position to 0,0 under the following conditions:

1. A Read is received and started by the server.
2. A Read completes for any reason.
3. The input buffer is redisplayed.

4.16.5 **Out-of-Band (H <--- S)** - This message contains an out-of-band character which has been received by the server.

MSGTYPE FLAGS CHARACTER

where

MSGTYPE (1) : C = 4

```

FLAGS (1)          : BM = Flags: .... ...D
                    D = discard control
                    0: do not alter output discard state
                    1: set output discard state to
                       "discard"

CHARACTER (1)      : A = Out-of-band character received.

```

4.16.6 **Unread (H ---> S)** - Requests the termination of the current Read, if there is a Read active.

MSGTYPE FLAGS

where

```

MSGTYPE (1)        : C = 5

FLAGS (1)          : BM = Flags: .... ...C
                    C = Unread condition
                    0: unconditional Unread
                    1: only do unread if the input and
                       type-ahead buffers are empty

```

4.16.7 **Clear Input (H ---> S)** - Requests clearing of the type-ahead and input buffers.

MSGTYPE FLAGS

where

```

MSGTYPE (1)        : C = 6

FLAGS (1)          : C = 0

```

4.16.8 **Write (H ---> S)** - Carries function request flags and, optionally, output data.

MSGTYPE FLAGS PREFIX-VALUE POSTFIX-VALUE DATA

```

MSGTYPE (1)        : C = 7

FLAGS (2)          : BM = Flags: .... TSQQ PPEB DLUU
                    UU = lock handling definition
                    0: unlock; default value for use if
                       locking parameter not specified
                       in WRITE request

```

1: lock before output; do not unlock after output
 2: lock before output and unlock after output
 3: lock before output and unlock after output; redisplay after output

L = "newline" flag
 0: no special action
 1: at the end of the Write, output a line-feed and set an internal "skip-line-feed" flag so that if the first character in the next Write is a line-feed, double spacing is avoided

D = "set output discard state" flag
 0: do not modify output discard state
 1: set output discard state to "do not discard" (before transmitting this data, if any)

B = "beginning of message" flag
 0: the data contained in this message is not the beginning of a host data message
 1: the data contained in this message is the beginning of a host data message

E = "end of message" flag
 0: the data contained in this message is not the end of a host data message
 1: the data contained in this message is the end of a host data message

PP = prefix code
 0: there is no prefix data; ignore the PREFIX-VALUE field
 1: there is a "newline" count contained in the PREFIX-VALUE field
 2: there is a prefix character contained in the PREFIX-VALUE field

QQ = postfix code
 0: there is no postfix data; ignore the POSTFIX-VALUE field
 1: there is a "newline" count contained in the POSTFIX-VALUE field

2: there is a postfix character contained in the POSTFIX-VALUE field

S = "completion status requested" flag
0: do not send a Write Completion message on completion of this request
1: send a Write Completion message on completion of this request

T = "transparent" flag
0: no special action
1: data from this Write message is written to Foundation Services using transparent Write-characteristic

PRE-FIX-VALUE (1) : B = 0, pre-fix newline count, or character, as defined by the PP flag.

POST-FIX-VALUE (1) : B = 0, post-fix newline count, or character, as defined by the QQ flag.

DATA : A = Output data. (This field may be absent.)

4.16.9 Write Completion (H <--- S) - Carries write completion information.

MSGTYPE FLAGS HORIZONTAL-POS VERTICAL-POS

where

MSGTYPE (1) : C = 8

FLAGS (1) : BM = Flags:D

D = discard status
0: no output lost due to output discard (control-O) by user
1: some output lost due to output discard (control-O) by user

HORIZONTAL-POS (2) : B = The relative horizontal position change on the presentation device since the beginning of this Write.

VERTICAL-POS (2) : B = The relative vertical position change (number of lines) on the presentation device since the beginning of this Write.

NOTE

The VERTICAL and HORIZONTAL-POS parameters are undefined under the following circumstances:

1. WRITE with embedded escape sequences.
2. Full duplex mode (i.e., unlocked Read echos and Writes are taking place simultaneously).

The host resets its cursor position to 0,0 under the following conditions:

1. A Read is received and started by the server.
2. A Read completes for any reason.
3. The input buffer is redisplayed.

4.16.10 **Discard State (H <--- S)** - Carries the output discard state most recently selected by the terminal user.

MSGTYPE FLAGS

where

MSGTYPE (1) : C = 9

FLAGS (1) : BM = Flags:D

D = discard control
0: output is to be discarded
1: output is not to be discarded

4.16.11 **Read Characteristics (H ---> S)** - Requests the current values of selected terminal characteristics to be returned to the host.

MSGTYPE FLAGS PARAMETER

where

MSGTYPE (1) : C = 10

FLAGS (1) : C = 0

PARAMETER = This field may be repeated multiple times. This field contains a parameter set as two subfields as follows:

SELECTOR CHARACTER

where:

SELECTOR(2) : BM = a characteristic selector (see the Selector Values for Characteristics, Section 4.17)

CHARACTER(1) : A = a subselector used only when the read characteristics is for the CHARACTER-ATTRIBUTES characteristic. This field is omitted if this is not for the CHARACTER-ATTRIBUTES characteristic

4.16.12 Characteristics (H <--> S) - In the host-to-server direction, modifies specified terminal characteristics. In the server-to-host direction, contains the values of previously requested characteristics.

MSGTYPE FLAGS PARAMETER

where

MSGTYPE (1) : C = 11

FLAGS (1) : C = 0

PARAMETER = This field may be repeated multiple times. This field contains a parameter set as two subfields as follows:

SELECTOR VALUE

where:

SELECTOR(2) : BM = a characteristic selector (see Section 4.17)

VALUE : = value field for characteristic (see Section 4.17.2 for format of each characteristic)

4.16.13 **Check Input (H ---> S)** - Requests the count of characters in the input and type-ahead buffers (combined) to be returned in an Input Count message.

MSGTYPE FLAGS

where

MSGTYPE (1) : C = 12

FLAGS (1) : C = 0

4.16.14 **Input Count (H <--- S)** - Carries the count of characters in the input and type-ahead buffers (combined); returned in response to a Check Input message.

MSGTYPE FLAGS COUNT

where

MSGTYPE (1) : C = 13

FLAGS (1) : C = 0

COUNT (2) : B = The requested input character count

4.16.15 **Input State (H <--- S)** - Carries a change from zero to non-zero or vice versa in the count of characters in the input and type-ahead buffers (combined).

MSGTYPE FLAGS

where

MSGTYPE (1) : C = 14

FLAGS (1) : BM = Flags:Z

Z = count change flag
0: count became zero
1: count became non-zero

4.16.16 **Reserved for VMS** -

MSGTYPE FLAGS

where

MSGTYPE (1) : C = 15

4.16.17 **Reserved for VMS -**

MSGTYPE FLAGS

where

MSGTYPE (1) : C = 16

4.16.18 **Reserved for VMS -**

MSGTYPE FLAGS

where

MSGTYPE (1) : C = 17

4.17 **Selector Values for Characteristics**

Characteristics SELECTOR values are carried in Read characteristics and characteristics messages. A SELECTOR is a 16-bit value whose format is:

LLLLLLLL IIIIIIII

LLLLLLLL: type of terminal characteristic identifier:

- 0: Foundation Physical Terminal Characteristic
- 1: Foundation Logical Terminal Characteristic
- 2: Network Command Terminal Logical Terminal Characteristic
- 3-127: RESERVED for future expansion

IIIIIIII: characteristic identifier (as defined in the table below for Network Command Terminal characteristics or in the Network Virtual Terminal Foundation specification for the Foundation Characteristics)

Characteristic identifier types in the range 128 thru 255 are reserved for implementors for "local" characteristics that exist only within the host or server that implements them. This allows implementations to define "local" characteristics at will with some reasonable guarantee that these "local" characteristics will neither impact nor interfere with other TSA implementations. It is a protocol error to receive a characteristic identifier in the range 128-255 from a remote implementation.

Characteristics VALUES are carried in characteristics messages. The form of a VALUE depends on its type. The type of each characteristic is defined in the tables below. Types are Boolean, Bit Map, Integer, String, and Compound. The format of each type is defined below.

Type	Format	Definition
----	-----	-----
Boolean:	BM(1)	Low-order bit is the value (T = 1, F = 0)
Bit Map:	BM(x)	x specified individually
Integer:	B(2)	16-bit integer
String:	A(I-255)	String of characters
Compound:		Compound characteristics value -- the value contains more than one field. The format of each compound characteristic value is defined in Section 4.17.2.1.

4.17.1 Foundation-maintained Characteristics - The following characteristics are maintained by the Foundation Services in the server system. This is not a complete list of Foundation characteristics, but, rather, a list of those Foundation characteristics necessary for the correct operation of the Network Command Terminal. They may be read and, if enabled, written via the command terminal protocol. Refer to the Network Virtual Terminal Foundation Services specification for the identifier values for these characteristics and a definition of semantics and range of values for each characteristic.

Characteristic	Type
-----	----
MODE-WRITING-ALLOWED	Boolean
TERMINAL-ATTRIBUTES	Bit Map BM(2)
TERMINAL-TYPE	String
OUTPUT-FLOW-CONTROL	Boolean
OUTPUT-PAGE-STOP	Boolean
FLOW-CHARACTER-PASS-THROUGH	Boolean
INPUT-FLOW-CONTROL	Boolean
LOSS-NOTIFICATION	Boolean
LINE-WIDTH	Integer
PAGE-LENGTH	Integer

Characteristic -----	Type -----
STOP-LENGTH	Integer
CR-FILL	Integer
LF-FILL	Integer
WRAP	Integer
HORIZONTAL-TAB	Integer
VERTICAL-TAB	Integer
FORM-FEED	Integer
INPUT-SPEED	Integer
OUTPUT-SPEED	Integer
CHARACTER-SIZE	Integer
PARITY-ENABLE	Boolean
PARITY-TYPE	Integer
MODEM-PRESENT	Boolean
AUTO-BAUD-DETECT	Boolean
MANAGEMENT-GUARANTEED	Boolean
SWITCH-CHARACTER-1	String
SWITCH-CHARACTER-2	String
EIGHT-BIT	Boolean
TERM-MANAGEMENT-ENABLED	Boolean

4.17.2 **Handler-Maintained Characteristics** - The following characteristics are maintained by the Network Command Terminal module in the server system. They may be read and written via the command terminal protocol.

<u>Characteristic</u>	<u>Identifier</u>	<u>Type</u>
IGNORE-INPUT	1	Boolean
CHARACTER-ATTRIBUTES	2	Compound
CONTROL-O-PASS-THROUGH	3	Boolean

Characteristic -----	Identifier -----	Type ----
RAISE-INPUT	4	Boolean
NORMAL-ECHO	5	Boolean
INPUT-ESCAPE-SEQUENCE- RECOGNITION-ENABLE	6	Boolean
OUTPUT-ESCAPE-SEQUENCE- RECOGNITION-ENABLE	7	Boolean
INPUT-COUNT-STATE	8	Integer
AUTO-PROMPT	9	Boolean
ERROR-PROCESSING	10	Bit Map BM(1)

4.17.2.1 **CHARACTER-ATTRIBUTES** Compound Characteristic - This is a fixed length characteristic.

CHARACTER MASK ATTRIBUTES

where

CHARACTER (1) : A = Character to which the following attributes apply.

MASK (1) : BM = An 8-bit bit mask which is used to select bits in the following attributes field which are to be changed. Bits in the ATTRIBUTES field which are to retain their old value (to be left unchanged), should have the corresponding bits in the MASK field set zero; significant bits in the following ATTRIBUTES field should have the corresponding bits in the MASK field set to 1.

ATTRIBUTES (1) : BM = Attribute-values: .FEE DIOO

OO = out-of-band handling

0: not out-of-band character -- used to cancel any previous out-of-band definition (initial default for all characters is NOT out-of-band)

1: immediate clear (valid only for control characters)

2: deferred clear (valid only for control characters)

3: immediate hello

I = include flag -- applies only when character is "immediate hello"
 0: do not include character in input data stream (default)
 1: include character in input data stream

D = out-of-band discard flag; whether a "clear" out-of-band character sets the output discard state to "discard"
 0: do not alter output discard state (default)
 1: set output discard state to "discard"

EE = control character echoing characteristic
 0: don't echo the character
 1: echo the character as itself
 2: echo in standard form (initial default value for all control characters) where standard form is
 CR echoes as CR,LF
 LF echoes as CR,LF
 ESC echoes as "\$" (dollar sign)
 all other characters echo as a "^" (up-arrow) followed by the printing character equal to $\langle \text{VALUE} + 64 \rangle \bmod 128$ (for example, VT echoes as ^K)
 3: echo in standard form followed by echo as itself

F = disable/enable special character function switch. The special character function switch is used to enable or disable the special functions associated with certain well known control characters, i.e.,
 OUTPUT-DISCARD (control-O),
 DELETE-CHARACTER (DEL), DELETE-WORD (control-W), CLEAR-INPUT (control-U),
 CLEAR-TYPE-AHEAD (control-X),
 REDISPLAY-INPUT (control-R), and
 QUOTE (control-V).
 0: disable special function for character
 1: enable special function for character (initial default value for control characters listed above)

APPENDIX A

Escape Sequence Recognition Algorithm

The following description of an algorithm for recognizing escape sequences is provided for the convenience of implementors.

The escape sequence recognizer can be described as a sequence of simple pattern matching rules. Each rule has an optional label, a match specification, and a label to branch to if the match is successful. Match specifications can be of two kinds: a quoted character, or a range of character codes, given as a pair of decimal numbers (minimum:maximum).

Processing begins by applying the first rule to the first character after an ESC. If a match succeeds, the rule designated by the success branch is applied to the next character in the stream. If a match fails, the next rule in the sequence is applied to the same character.

A special flag, "*", (indicated by the success branch) means that if this match succeeds, the recognition of an escape sequence is complete; if the match fails, the characters examined are not part of any valid escape sequence.

The comment included with a rule applies to the current character, or to the escape sequence being recognized, if the match succeeds.

Label	Match	Success Branch	Comment
-----	-----	-----	-----
	";"	10	VT52 sequence
	"?"	10	" "
	"O"	20	" "
	"Y"	30	VT52 seq. (fixed length)
	"["	15	ANSI control sequence
10	32:47	10	intermediate character
	48:126	*	final character
15	48:63	15	ANSI sequence parameter
20	32:47	20	intermediate character
	64:126	*	final character
30	32:126	40	fixed length VT52 seq.
40	32:126	*	" "

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____

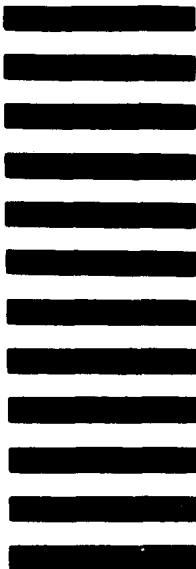
or
Country

---Do Not Tear - Fold Here and Tape---

digital



No Postage
Necessary
if Mailed in the
United States



BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE DOCUMENTATION
1925 ANDOVER STREET TWO/E07
TEWKSBURY, MASSACHUSETTS 01876

---Do Not Tear - Fold Here and Tape---

Cut Along Dotted Line

