

ARPANET
PROTOCOL HANDBOOK
APRIL 1976

Prepared for:

DEFENSE COMMUNICATIONS AGENCY
Code 535, Washington, D. C. 20305

By:

Elizabeth Feinler and Jon Postel
Network Information Center
Stanford Research Institute
Menlo Park, California 94025

Approved for public release; distribution unlimited

ia

ABSTRACT

The ARPANET Protocol Handbook is a collection of documents that describe the protocols of the ARPANET computer network as of April 1976. Protocols are the rules of communication between processes on the network. The protocols in use on the ARPANET form a tree structure. The basic protocol is the IMP-to-Host Protocol. Directly under that is the Host-to-Host Protocol. Spreading out beneath but still closely related are the process level protocols: TELNET, File Transfer, Remote Job Entry, and Graphics. Interspersed along the way are a few small protocols such as the Initial Connection Protocol, and the definition of the standard character set.

CONTENTS

ABSTRACT	i
TABLE-OF-CONTENTS	ii
INTRODUCTION	1
IMP-HOST PROTOCOLS	
Interface Message Processor, Specifications for the Interconnection of a Host and an IMP (BBN Rept. 1822)	3
NETWORK CONTROL PROTOCOLS	
Host/Host Protocol for the ARPANET (NIC 8246)	7
INITIAL CONNECTION PROTOCOLS	
Official Initial Connection Protocol (NIC 7101)	41
(Possible Deadlock) (RFC 202, NIC 7155)	47
Official TELNET-Logger Initial Connection Protocol (NIC 7103)	49
TELNET PROTOCOLS	
TELNET Protocol Specification (NIC 18639)	53
TELNET Option Specifications (NIC 18640)	67
TELNET Binary Transmission Option (NIC 15389)	71
TELNET Echo Option (NIC 15390)	75
TELNET Reconnection Option (NIC 15391)	81
TELNET Suppress Go Ahead Option (NIC 15392)	91
TELNET Approximate Message Size Negotiation Option (NIC 15393)	93
Revised TELNET Status Option (RFC 651, NIC 31154)	97

TELNET Timing Mark Option (NIC 16238)	101
Remote Controlled Transmission and Echoing TELNET Option (NIC 19859)	105
TELNET Output Line Width Option (NIC 20196)	119
TELNET Output Page Size Option (NIC 20197)	123
TELNET Output Carriage-Return Disposition Option (RFC 652, NIC 31155)	127
TELNET Output Horizontal Tabstops Option (RFC 653, NIC 31156)	131
TELNET Output Horizontal Tab Disposition Option (RFC 654, NIC 31157)	135
TELNET Output Formfeed Disposition Option (RFC 655, NIC 31158)	139
TELNET Output Vertical Tabstops Option (RFC 656, NIC 31159)	143
TELNET Output Vertical Tab Disposition Option (RFC 657, NIC 31160)	147
TELNET Output Linefeed Disposition (RFC 658, NIC 31161)	151
TELNET Extended ASCII Option (RFC 698, NIC 32964)	155
TELNET Extended-Options-List Option (NIC 16239)	159
USA Standard Code for Information Interchange (NIC 11246)	161
FILE TRANSFER PROTOCOLS	
File Transfer Protocol (RFC 542, NIC 17759)	177
Revised FTP Reply Codes (RFC 640, NIC 30843)	219
MAIL PROTOCOLS	
Mail Protocol (NIC 24664)	237
Message Transmission Protocol	241

REMOTE JOB ENTRY PROTOCOLS

 Remote Job Entry Protocol (RFC 407, NIC 12112) 249

NETWORK GRAPHICS PROTOCOLS

 A Network Graphics Protocol (NIC 24308) 273

DATA CONFIGURATION PROTOCOLS

 Data Reconfiguration Service -
 An Implementation Specification
 (RFC 166, NIC 6780) 333

Numbers listed in the Table-of-Contents refer to the page numbers in parentheses. Center paging pertains only to the protocol in which it occurs. Center paging has been included so that page references within the text of a given protocol that refer to the original paging will still have meaning.

INTRODUCTION

WHAT IS THE PROTOCOL HANDBOOK?

The ARPANET Protocol Handbook is a document that describes specifications for the accepted protocols in use on the ARPANET computer network. The Handbook is complete except for the IMP-HOST Protocol (Interface Message Processor, Specifications for the Interconnection of a Host and an IMP, Rept. 1822, Bolt Beranek & Newman, Inc., Cambridge, Mass., Rev. Jan. 1976.). Since this particular protocol is very long and is distributed to the Liaison of each new host on the ARPANET, it has been omitted here. It may be obtained from the National Technical Information Service (NTIS), Springfield, Va. 22161 as AD A018565.

HOW IS THE PROTOCOL INFORMATION OBTAINED?

Information included in this handbook is supplied by the Coordinator of the ARPANET Network Working Group to the Network Information Center at the request of the Defense Communications Agency. Dr. Jon Postel, Augmentation Research Center, Stanford Research Institute, Menlo Park, Calif. 94025 is the current Network Working Group Coordinator.

ONLINE ACCESS TO ARPANET PROTOCOLS VIA THE ARPANET

Users of the ARPANET may access the Protocols online at OFFICE-1 (Host 43, dec.) by logging in as NICGUEST, Password ARPA, and then typing the word 'NIC' followed by a carriage return. The files are available for FTP from OFFICE-1 also.

RELATED INFORMATION

Before a proposed protocol is accepted for use on the ARPANET it is, discussed, reviewed, and often revised by members of the Network Working Group and other interested parties. This dialog is captured in a set of technical notes known as Requests for Comments (RFCs). RFCs are available to ARPANET users through FTP from the directory <NETINFO> at OFFICE-1. Members of the ARPANET who wish to be on distribution for online notices of new RFCs should contact Jon Postel (POSTEL@BBNB).

ORDERING ADDITIONAL COPIES OF THE CURRENT NETWORK PROTOCOLS

Additional copies of the Protocol Handbook may be ordered from the Network Information Center (NIC), Stanford Research Institute, Menlo Park, California 94025, or from NTIS (address given above). The NIC charges a fee of \$10.00, payable in advance, to cover printing and handling.

IMP-HOST PROTOCOL

This Protocol has been omitted. It is available from the National Technical Information Service (NTIS), Springfield, Va. 22161 as AD A018565 or from the Network Control Center, Bolt, Beranek, and Newman, Inc., Cambridge, Massachusetts.

NETWORK CONTROL PROTOCOLS

Preceding page blank

(6)

Alex McKenzie
Bolt Beranek and Newman, Inc.
Cambridge, Mass.

Obsoletes NIC 7147

Host/Host Protocol for the ARPA Network

Preceding page blank

PREFACE

This document specifies a protocol for use in communication between Host computers on the ARPA Network. In particular, it provides for connection of independent processes in different Hosts, control of the flow of data over established connections, and several ancillary functions. Although basically self-contained, this document specifies only one of several ARPA Network protocols; all protocol specifications are collected in the document Current Network Protocols, NIC #7104 (AD A003890).

This document superceded NIC #7147 of the same title. Principal differences between the documents include:

- . prohibition of spontaneous RET, ERP, and RRP commands
- . a discussion of the problem of unanswered CLS commands (page 11)
- . a discussion of the implications of queueing and not queueing RFCs (page 9)
- . the strong recommendation that received ERR commands be logged, and some additional ERR specifications.

In addition to the above, several minor editorial changes have been made.

Although there are many individuals associated with the network who are knowledgeable about protocol issues, individuals with questions pertaining to network protocols should initially contact one of the following:

Alex McKenzie
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138
(617) 491-1850 ext. 441

Jon Postel
Stanford Research Institute
Augmentation Research Center
333 Ravenswood Avenue
Menlo Park, California 94025
(415) 326-6200 ext 3718

Preceding page blank

TABLE OF CONTENTS

I. INTRODUCTION.....	3
An overview of the multi-leveled protocol structure in the ARPA Network.	
II. COMMUNICATION CONCEPTS.....	5
Definitions of terminology and a description of the overall strategy used in Host-to-Host communications.	
III. NCP FUNCTIONS.....	8
The meat of the document for the first-time reader. Host-to-Host "commands" are introduced with descriptions of conditions of their use, discussion of possible problems, and other background material.	
Connection Establishment	8
Connection Termination	10
Flow Control	12
Interrupts	14
Test Inquiry	15
Reinitialization	15
IV. DECLARATIVE SPECIFICATIONS.....	17
Details for the NCP implementer. A few additional "commands" are introduced, and those described in Section III are reviewed. Formats and code and link assignments are specified.	
Message Format	17
Link Assignment	19
Control Messages	19
Control Commands	19
Opcode Assignments	25
Control Command Summary	26

Preceding page blank

I. INTRODUCTION

The ARPA Network provides a capability for geographically separated computers, called Hosts, to communicate with each other. The Host computers typically differ from one another in type, speed, word length, operating system, etc. Each Host computer is connected into the network through a local small computer called an Interface Message Processor (IMP). The complete network is formed by interconnecting these IMPs, all of which are virtually identical, through wideband communications lines supplied by the telephone company. Each IMP is programmed to store and forward messages to the neighboring IMPs in the network. During a typical operation, a Host passes a message to its local IMP; the first 32 bits of this message include the "network address" of a destination Host. The message is passed from IMP to IMP through the Network until it finally arrives at the destination IMP, which in turn passes it along to the destination Host.

Specifications for the physical and logical message transfer between a Host and its local IMP are contained in Bolt Beranek and Newman (BBN) Report No. 1822 (AD A018565). These specifications are generally called the first level protocol or Host/IMP Protocol. This protocol is not by itself, however, sufficient to specify meaningful communication between processes running in two dissimilar Hosts. Rather, the processes must have some agreement as to the method of initiating communication, the interpretation of transmitted data, and so forth. Although it would be possible for such agreements to be reached by each pair of Hosts (or processes) interested in communication, a more general arrangement is desirable in order to minimize the amount of implementation necessary for Network-wide communication. Accordingly, the Host organizations formed a Network Working Group (NWG) to facilitate an exchange of ideas and to formulate additional specifications for Host-to-Host communications.

The NWG has adopted a "layered" approach to the specification of communications protocol. The inner layer is the Host/IMP protocol. The next layer specifies methods of establishing communications paths, managing buffer space at each end of a communications path, and providing a method of "interrupting" a communications path. This protocol, which will be used by all higher-level protocols, is known as the second level protocol, or Host/Host protocol. (It is worth noting that, although the IMP sub-network provides a capability for message switching, the Host/Host protocol is based on the concept of line switching.) Examples of further layers of protocol currently developed or anticipated include:

- 1) An initial Connection Protocol (ICP) which provides a convenient standard method for several processes to gain simultaneous access to some specific process (such as the "logger") at another Host.

Preceding page blank

- 2) A Telecommunication Network (TELNET) protocol which provided for the "mapping" of an arbitrary Keyboard-printer terminal into a network Virtual Terminal (NVT), to facilitate communication between a terminal user at one Host site and a terminal-serving process at some other site and a terminal-serving process at some other site which "expects" to be connected to a (local) terminal logically different from the (remote) terminal actually in use. The TELNET protocol specifies use of the ICP to establish the communication path between the terminal user and the terminal-service process.
- 3) A Data Transfer protocol to specify standard methods of formatting data for shipment through the network.
- 4) A File Transfer protocol to specify methods for reading, writing, and updating files stored at a remote Host. The File Transfer protocol specifies that the actual transmission of data should be performed in accordance with the Data Transfer protocol.
- 5) A Graphics protocol to specify the means for exchanging graphics display information.
- 6) A Remote Job Service (RJS) protocol to specify methods for submitting input to, obtaining output from, and exercising control over Hosts which provide batch processing facilities.

The remainder of this document describes and specifies the Host/Host, or second level, protocol as formulated by the Network Working Group.

II. COMMUNICATION CONCEPTS

The IMP sub-network imposes a number of physical restrictions on communications between Hosts; these restrictions are presented in BBN Report Number 1822. In particular, the concepts of leaders, messages, padding, links, and message types are of interest to the design of Host/Host protocol. The following discussion assumes that the reader is familiar with these concepts.

Although there is little uniformity among the Hosts in either hardware or operating systems, the notion of multiprogramming dominates most of the systems. These Hosts can each concurrently support several users, with each user running one or more processes. Many of these processes may want to use the network concurrently, and thus a fundamental requirement of the Host/Host protocol is to provide for process-to-process communication over the network. Since the first level protocol only takes cognizance of Hosts, and since the several processes in execution within a Host are usually independent, it is necessary for the second level protocol to provide a richer addressing structure.

Another factor which influenced the Host/Host protocol design is the expectation that typical process-to-process communication will be based, not on a solitary message, but rather upon a sequence of messages. One example is the sending of a large body of information, such as a data base, from one process to another. Another example is an interactive conversation between two processes, with many exchanges.

These considerations led to the introduction of the notions of connections, a Network Control program, a "control link", "control commands", connection byte size, message headers, and sockets.

A connection is an extension of a link. A connection couples two processes so that output from one process is input to the other. Connections are defined to be simplex (i.e., unidirectional), so two connections are necessary if a pair of processes are to converse in both directions.

Processes within a Host are envisioned as communicating with the rest of the network through a Network Control Program (NCP), resident in that Host, which implements the second level protocol. The primary function of the NCP is to establish connections, break connections, and control data flow over the connections. We will describe the NCP as though it were part of the operating system of a Host supporting multiprogramming, although the actual method of implementing the NCP may be different in some Hosts.

In order to accomplish its tasks, the NCP of one Host must communicate with the NCPs of other Hosts. To this end, a particular link between

each pair of Hosts has been designated as the CONTROL LINK. Messages transmitted over the control link are called CONTROL MESSAGES, (NOTE: that in BBN Report Number 1822, messages of non-zero type are called control messages, and are used to control the flow of information between a Host and its IMP. In this document, the term "control message" is used for a message of type zero transmitted over the control link. The IMPs take no special notice of these messages) and must always be interpreted by an NCP as a sequence of one or more CONTROL COMMANDS. For example, one kind of control command is used to initiate a connection, while another kind carries notification that a connection has been terminated.

The concept of a message, as used above, is an artifact of the IMP sub-network; network message boundaries may have little intrinsic meaning to communication processes. Accordingly, it has been decided that the NCP (rather than each transmitting process) should be responsible for segmenting interprocess communication into network messages. Therefore, it is a principal of the second level protocol that no significance may be inferred from message boundaries by a receiving process. **The only exception to this principle is in control messages, each of which must contain an integral number of control commands.**

Since message boundaries are selected by the transmitting NCP, the receiving NCP must be prepared to concatenate successive messages from the network into a single (or differently divided) transmission for delivery to the receiving process. The fact that Hosts have different word sizes means that a message from the network might end in the middle of a word at the receiving end, and thus the concatenation of the next message might require the receiving Host to carry out extensive bit-shifting. Because bit-shifting is typically very costly in terms of computer processing time, the protocol includes the notions of connection byte size and message headers.

As part of the process of establishing a connection, the processes involved must agree on a CONNECTION BYTE SIZE. Each message sent over the connection must then contain an integral number of bytes of this size. Thus the pair of processes involved in a connection can choose a mutually convenient byte size, for example, the least common multiple of their Host word lengths. It is important to note that the ability to choose a byte size MUST be available to the processes involved in the connection; an NCP is prohibited from imposing an arbitrary byte size on any process running in its own Host. In particular, an outer layer of protocol may specify a byte size to be used by that protocol. If some NCP is unable to handle that byte size, then the outer layer of protocol will not be implementable on that Host.

III. NCP FUNCTIONS

The functions of the NCP are to establish connections, terminate connections, control flow, transmit interrupts, and respond to test inquiries. These functions are explained in this section, and control commands are introduced as needed. In Section IV the formats of all control commands are presented together.

CONNECTION ESTABLISHMENT

The commands used to establish a connection are STR (sender-to-receiver) and RTS (receiver-to-sender).

8*	32	32	8
<hr/>			
! STR !	send socket	! receive socket	! size !
<hr/>			
8	32	32	8
<hr/>			
! RTS !	receive socket	! send socket	! link !
<hr/>			

(*NOTE: The number shown above each control command field is the length of that field in bits.

The STR command is sent from a prospective sender to a prospective receiver, and the RTS from a prospective receiver to a prospective sender. The send socket field names a socket local to the prospective sender; the receive socket field names a socket local to the prospective receiver. In the STR command, the "size" field contains an unsigned binary number (in the range 1 to 255; zero is prohibited) specifying the byte size to be used for all messages over the connection. In the RTS command, the "link" field specifies a link number; all messages over the connection must be sent over the link specified by this number. These two commands are referred to as requests-for-connection (RFCs). An STR and an RTS match if the receive socket fields match and the send socket fields match. A connection is established when a matching pair of RFCs have been exchanged. **Hosts are prohibited from establishing more than one connection to any local socket.**

With respect to a particular connection, the Host containing the send socket is called the SENDING HOST and the Host containing the receive socket is called the RECEIVING HOST. A Host may connect one of its receive sockets to one of its send sockets, thus becoming both the sending Host and the receiving Host for that connection. These terms apply only to data flow; control messages will, in general, be transmitted in both directions.

The IMP sub-network requires that the first 32 bits of each message (called the leader) contain addressing information, including destination Host address and link number. The second level protocol extends the required information at the beginning of each message to a total of 72 bits; these 72 bits are called the MESSAGE HEADER. A length of 72 bits is chosen since most Hosts either can work conveniently with 8-bit units of data or have word lengths of 18 or 36 bits; 72 is the least common multiple of these lengths. Thus, the length chosen for the message header should reduce bit-shifting problems for many Hosts. In addition to the leader, the message header includes a field giving the byte size used in the message, a field giving the number of bytes in the message, and "filler" fields. The format of the message header is fully described in Section IV.

Another major concern of the second level protocol is a method for reference to processes in other Hosts. Each Host has some internal scheme for naming processes, but these various schemes are typically different and may even be incompatible. Since it is not practical to impose a common internal process naming scheme, a standard intermediate name space is used, with a separate portion of the name space allocated to each Host. Each Host must have the ability to map internal process identifiers into its portion of this name space.

The elements of the name space are called SOCKETS. A socket forms one end of a connection, and a connection is fully specified by a pair of sockets. A socket is identified by a Host number and a 32-bit socket number. The same 32-bit number in different Hosts represents different sockets.

A socket is either a RECEIVE SOCKET or a SEND SOCKET, and is so marked by its low-order bit (0 = receive; 1 = send). This property is called the socket's GENDER. The sockets at either end of a connection must be of opposite gender. Except for the gender, second level protocol places no constraints on the assignment of socket numbers within a Host.

A Host sends an RFC either to request a connection, or to accept a foreign Host's request. Since RFC commands are used both for requesting and for accepting the establishment of a connection, it is possible for either of two cooperating processes to initiate connection establishment. As a consequence, a family of processes may be created with connection-initiating actions built-in, and the processes within this family may be started up (in different Hosts) in arbitrary order provided that appropriate queueing is performed by the Hosts involved (see below).

There is no prescribed lifetime for an RFC. A Host is permitted to queue incoming RFCs and withhold a response for an arbitrarily long time, or, alternatively, to reject requests (see Connection Termination below) immediately if it does not have a matching RFC outstanding. It may be reasonable, for example, for an NCP to queue an RFC that refers to some currently unused socket until a local process takes control of that socket number and tells the NCP to accept or reject the request. Of course, the Host which sent the RFC may be unwilling to wait for an arbitrarily long time, so it may abort the request. On the other hand, some NCP implementations may not include any space for queueing RFCs, and thus can be expected to reject RFCs unless the RFC sequence was initiated locally.

QUEUEING CONSIDERATIONS

The decision to queue, or not queue, incoming RFCs has important implications which NCP implementers must not ignore. Each RFC which is queued, of course, requires a small amount of memory in the Host doing the queueing. If each incoming RFC is queued until a local process seizes the local socket and accepts (or rejects) the RFC, but no local process ever seizes the socket, the RFC must be queued "forever." Theoretically this could occur infinitely many times (there is no reason not to queue several RFCs for a single local socket, letting the local process decide which, if any, to accept) thus requiring infinite storage for the RFC queue. On the other hand, if no queueing is performed the cooperating processes described above will be able to establish a desired connection only by accident (when they are started up such that one issues its RFC while the RFC of the other is in transit in the network - clearly an unlikely occurrence).

Perhaps the most reasonable solution to the problems posed above is for EACH NCP to give processes running in its own Host two options for attempting to initiate connections. The first option would allow a process to cause an RFC to be sent to a specified remote socket; with the NCP notifying the process as to whether the RFC were accepted or rejected by the remote host. The second option would allow a process to tell ITS OWN NCP to "listen" for an RFC to a specified local socket from some remote socket (the process might also specify the particular remote socket and/or Host it wishes to

communicate with) and to accept the RFC (i.e., return a matching RFC) if and when it arrives. Note that this also involves queueing (of "listen" requests), but it is internal queueing which is susceptible to reasonable management by the local Host. If this implementation were available, one of two cooperating processes could "listen" while the other process caused a series of RFCS to be sent to the "listening" socket until one was accepted. Thus, no queueing of incoming RFCS would be required, although it would do no harm.

It is the intent of the protocol that each NCP should provide either the "listen" option described above or a SUBSTANTIAL queueing facility. This is not, however, an absolute requirement of the protocol.

CONNECTION TERMINATION

The command used to terminate a connection is CLS (close).

8	32	32

! CLS !	my socket	! your socket !

The "my socket" field contains the socket local to the sender of the CLS command. The "your socket" field contains the socket local to the receiver of the CLS command. **Each side must send and receive a CLS command before connection termination is completed and the sockets are free to participate in other connections.**

It is not necessary for a connection to be established (i.e., for BOTH RFCS to be exchanged) before connection termination begins. For example, if a Host wishes to refuse a request for connection, it sends back a CLS instead of a matching RFC. The refusing Host then waits for the initiating Host to acknowledge the refusal by returning a CLS. Similarly, if a Host wishes to abort its outstanding request for a connection, it sends a CLS command. The foreign Host is obliged to acknowledge the CLS with its own CLS. Note that even though the connection was never established, CLS commands must be EXCHANGED before the sockets are free for other use.

After a connection is established, CLS commands sent by the receiver and sender have slightly different effects. CLS commands sent by the sender indicate that no more messages will be sent over the connection. **This command must not be sent if there is a message in transit over the connection.** A CLS command sent by the receiver acts as a demand on the sender to terminate transmission. However, since there is a delay in getting the CLS command to the sender, the receiver must expect more input.

A Host should "quickly" acknowledge an incoming CLS so the foreign Host can purge its tables. However, **there is no prescribed time period in which a CLS must be acknowledged.**

Because the CLS command is used both to initiate closing, aborting and refusing a connection, and to acknowledge closing, aborting and refusing a connection, race conditions can occur. However, they do not lead to ambiguous or erroneous results, as illustrated in the following examples:

EXAMPLE 1: Suppose that Host A sends B a request for connection, and then A sends a CLS to Host B because it is tired of waiting for a reply. However, just when A sends its CLS to B, B sends a CLS to A to refuse the connection. A will "believe" B is acknowledging the abort, and B will "believe" A is acknowledging its refusal, but the outcome will be correct.

EXAMPLE 2: Suppose that Host A sends Host B an RFC followed by a CLS as in example 1. In this case, however, B sends a matching RFC to A just when A sends its CLS. Host A may "believe" that the RFC is an attempt (on the part of B) to establish a new connection or may understand the race condition; in either case it can discard the RFC since its socket is not yet free. Host B will "believe" that the CLS is breaking an ESTABLISHED connection, but the outcome is correct since a matching CLS is the required response, and both A and B will then terminate the connection.

Every NCP implementation is faced with the problem of what to do if a matching CLS is not returned "quickly" by a foreign Host (i.e., if the foreign Host appears to be violating protocol in this respect). One naive answer is to hold the connection in a partially closed state "forever" waiting for a matching CLS. There are two difficulties with this solution. First, the socket involved may be a "scarce resource" such as the "logger" socket specified by an Initial Connection Protocol (see NIC #7101) which the local Host cannot afford to tie up indefinitely. Second, a partially broken (or malicious) process in a foreign Host may send an unending stream of RFCs which the local Host wishes to refuse by sending CLS commands and waiting for a match. This could, in worst cases, require 2 to the 32nd factorial socket pairs to be stored before duplicates began to appear. Clearly, no host is prepared to store (or search) this much information.

A second possibility sometimes suggested is for the Host which is waiting for matching CLS commands (Host A) to send a RST (see page 17) to the offending Host (Host B), thus allowing all tables to be reinitialized at both ends. This would be rather unsatisfactory to any user at Host A who happened to be performing useful work on Host B via network connections, since these connections would also be broken by the RST.

Most implementers, recognizing these problems have adopted some unofficial timeout period after which they "forget" a connection even if a matching CLS has not been received. The danger with such an arrangement is that if a second connection between the same pair of sockets is later established, and a CLS finally arrives for the first connection, the second connection is likely to be closed. This situation can only arise, however, if one Host violates protocol in TWO ways; first by failing to respond quickly to an incoming CLS, and second by permitting establishment of a connection involving a socket which it believes is already in use. It has been suggested that the network adopt some standard timeout period, but the NWG has been unable to arrive at a period which is both short enough to be useful and long enough to be acceptable to every Host.

Timeout periods in current use seem to range between approximately one minute and approximately five minutes. **It must be emphasized that all timeout periods, although they are relatively common, reasonably safe, and quite useful, are in violation of the protocol since their use can lead to connection ambiguities.**

FLOW CONTROL

After a connection is established, the sending Host sends messages over the agreed-upon link to the receiving Host. The receiving NCP accepts messages from its IMP and queues them for its various processes. Since it may happen that the messages arrive faster than they can be processed, some mechanism is required which permits the receiving Host to quench the flow from the sending Host.

The flow control mechanism requires the receiving Host to allocate buffer space for each connection and to notify the sending Host of how much space is available. The sending Host keeps track of how much room is available and never sends more data than it believes the receiving Host can accept.

To implement this mechanism, the sending Host keeps two counters associated with each connection, a MESSAGE COUNTER and a BIT COUNTER. Each counter is initialized to zero when the connection is established and is increased by allocate (ALL) control commands sent from the receiving Host as described below. When sending a message, the NCP of the sending Host subtracts one from the message counter and the TEXT LENGTH (defined below) from the bit counter. The sender is prohibited from sending if either counter would be decremented below zero. The sending Host may also return (RET) command upon receiving a give-back (GVB) command from the receiving Host (see below).

The TEXT LENGTH of a message is defined as the product of the connection byte size and the byte count for the message; both of these quantities appear in the message header. Messages with a zero byte count, hence a zero text length, are specifically permitted. Messages with zero text

length do not use bit space allocation, but do use message space allocation. The flow control mechanisms do not pertain to the control link, since connections are never explicitly established over this link.

The control command used to increase the sender's bit counter and message counter is ALL (allocate).

8	8	16	32

! ALL		! link ! msg space !	bit space !

This command is sent only from the receiving Host to the sending Host, and is legal only when a connection using the link number appearing in the "link" field is established. The "msg space" field and the "bit space" field are defined to be unsigned binary integers specifying the amounts by which the sender's message counter and bit counter (respectively) are to be incremented. The receiver is prohibited from incrementing the sender's message counter above 2 to the 16th minus 1, or the sender's bit counter above 2 to the 32nd minus 1. In general this rule will require the receiver to maintain counters which are incremented and decremented according to the same rules as the sender's counters.

The receiving Host may request that the sending Host return all or part of its current allocation. The control command for this request is GVB (give-back).

8	8	8	8

! GVB ! link ! f(m) ! f(b) !			

This command is sent only from the receiving Host to the sending Host, and is legal only when a connection using the link number in the "link" field is established. The fields f(m) and f(b) are defined as the fraction (in 128ths) of the current message space allocation and bit space allocation (respectively) to be RETURNED. If either of the fractions is equal to or greater than one, ALL of the corresponding allocation must be returned. Fractions are used since, with messages in transit, the sender and receiver may not agree on the actual allocation at every point in time.

Upon receiving a GVB command, the sending Host must return AT LEAST (NOTE: In particular, fractional returns must be rounded up, not truncated) the requested portions of the message and bit space allocations. (A sending Host is prohibited from spontaneously returning portions of the message and bit space allocations.) The control command for performing this function is RET (return).

8 8 16

32

! RET ! link ! msg space ! bit space !

This command is sent only from the sending Host to the receiving Host, and is legal only when a connection using the link number in the "link" field is established and a GVB command has been received from the receiving Host. The "msg space" field and the "bit space" field are defined as unsigned binary integers specifying the amounts by which the sender's message counter and bit counter (respectively) have been decremented due to the RET activity (i.e., the amounts of message and bit space allocation being returned). NCPs are obliged to answer a GVB with a RET "quickly"; however, there is no prescribed time period in which the answering RET must be sent.

Some Hosts will allocate only as much space as they can guarantee for each link. These Hosts will tend to use the GVB command only to reclaim space which is being filled very slowly or not at all. Other Hosts will allocate more space than they have, so that they may use their space more efficiently. Such a Host will then need to use the GVB command when the input over a particular link comes faster than it is being processed.

INTERRUPTS

The second level protocol has included a mechanism by which the transmission over a connection may be "interrupted." The meaning of the "interrupt" is not defined at this level, but is made available for use by outer layers of protocol. The interrupt command sent from the receiving Host to the sending Host is INR (interrupt-by-receiver).

8 8

! INR ! link !

The interrupt command sent from the sending Host to the receiving Host is INS (interrupt-by-sender).

8 8

! INS ! link !

The INR and INS commands are legal only when a connection using the link number in the "link" field is established.

TEST INQUIRY

It may sometimes be useful for one Host to determine if some other Host is capable of carrying on network conversations. The control command to be used for this purpose is ECO (echo).

8 8

! ECO ! data !

The "data" field may contain any bit configuration chosen by the Host sending the ECO. Upon receiving an ECO command an NCP must respond by returning the data to the sender in an ERP (echo-reply) command.

8 8

! ERP ! data !

A Host should "quickly" respond (with an ERP command) to an incoming ECO command. However, there is no prescribed time period, after the receipt of an ECO, in which the ERP must be returned. A Host is prohibited from sending an ERP when no ECO has been received, or from sending an ECO to a Host while a previous ECO to that Host remains "unanswered." Any of the following constitute an "answer" to an ECO: information from the local IMP that the ECO was discarded by the network (e.g., IMP/Host message type 7 - Destination Dead), ERP, RST, or RRP (see below).

REINITIALIZATION

Occasionally, due to lost control messages, system "crashes", NCP errors, or other factors, communication between two NCPs will be disrupted. One possible effect of any such disruption might be that neither of the involved NCPs could be sure that its stored information regarding connections with the other Host matched the information stored by the NCP of the other Host. In this situation, an NCP may wish to reinitialize its tables and request that the other Host do likewise; for this purpose the protocol provides the pair of control commands RST (reset) and RRP (reset-reply).

8

! RST !

8

! RRP !

The RST command is to be interpreted by the Host receiving it as a signal to purge its NCP tables of any entries which arose from communication with the Host to which sent the RST. The Host sending the RST should likewise purge its NCP tables of any entries which arise from communication with the Host to which the RST was sent. The Host receiving the RST should acknowledge receipt by returning an RRP.

Once the first Host has sent an RST to the second Host, the first Host is not obliged to communicate with the second Host (except for responding to RST) until the second Host returns an RRP. In fact, to avoid synchronization errors, the first Host SHOULD NOT communicate with the second until the RST is answered. Of course, if the IMP subnetwork returns a "Destination Dead" (type 7) message in response to the control message containing the RST, an RRP should not be expected. If both NCPs decide to send RSTs at approximately the same time, then each Host will receive an RST and each must answer with an RRP, even though its own RST has not yet been answered.

Some Hosts may choose to "broadcast" RSTs to the entire network when they "come up." One method of accomplishing this would be to send an RST command to each of the 256 possible Host addresses; the IMP subnetwork would return a "Destination Dead" (type 7) message for each non-existent Host, as well as for each Host actually "dead." **However, no Host is ever obliged to transmit an RST command.**

Hosts are prohibited from sending an RRP when no RST has been received. Further, Hosts may send only one RST in a single control message and should wait a "reasonable time" before sending another RST to the same Host. Under these conditions, a single RRP constitutes an "answer" to ALL RSTs sent to that Host, and any other RRPs arriving from that Host should be discarded.

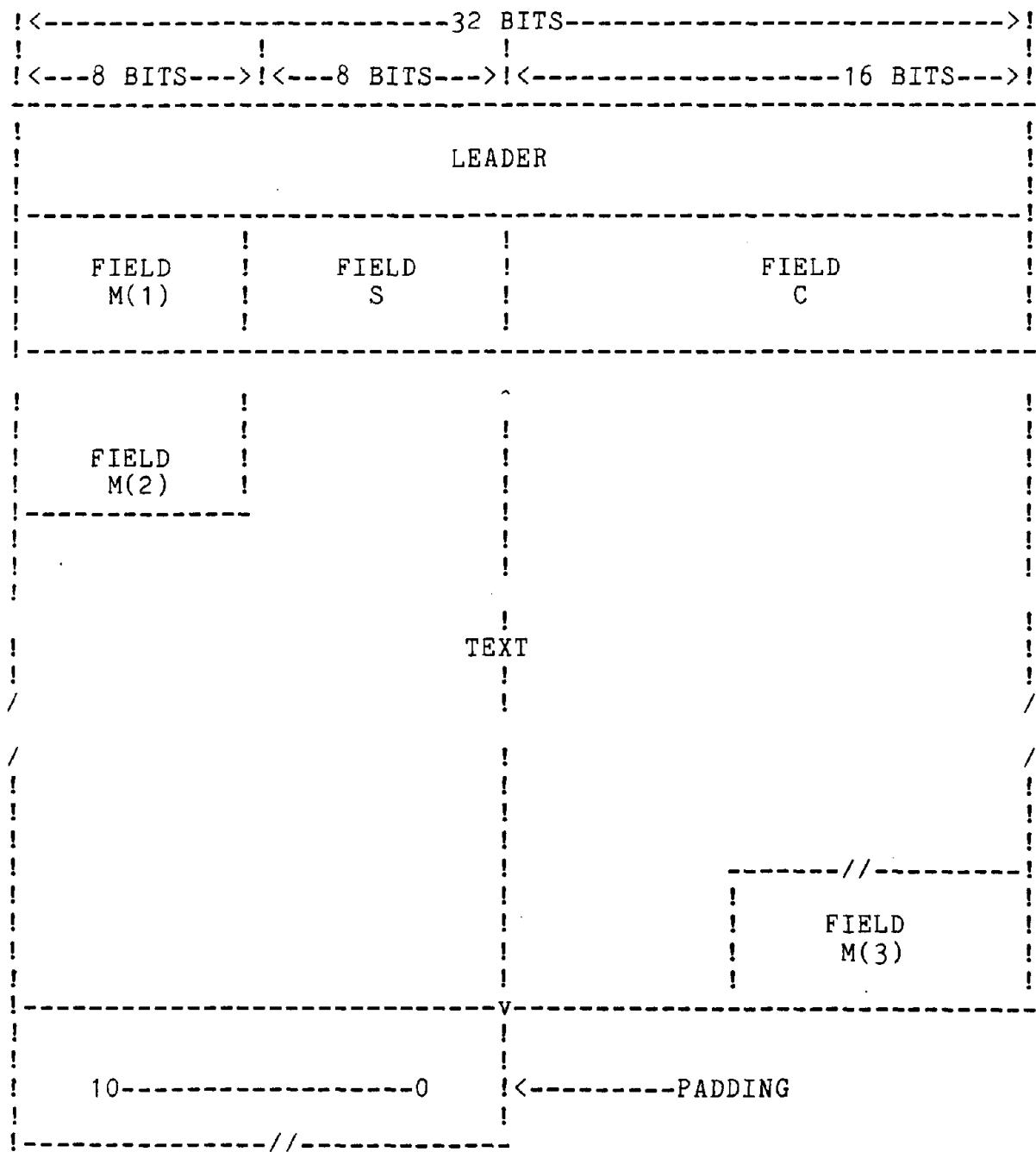
IV. DECLARATIVE SPECIFICATIONS

MESSAGE FORMAT

All Host-to-Host messages (i.e., messages of type zero) shall have a header 72 bits long consisting of the following fields (see Figure 1):

- Bits 1-32 Leader - The contents of this field must be constructed according to the specifications contained in BBN Report Number 1822.
- Bits 33-40 Field M(1) - Must be zero.
- Bits 41-48 Field S - Connection byte size. This size must be identical to the byte size on the STR used in establishing the connection. If this message is being transmitted over the control link the connection byte size must be 8.
- Bits 49-64 Field C - Byte Count. This field specifies the number of bytes in the text portion of the message. A zero value in the C field is explicitly permitted.
- Bits 65-72 Field M(2) - Must be zero.

Following the header, the message shall consist of a text field of C bytes, where each byte is S bits in length. Following the text there will be field M(3) followed by padding. The M(3) field is zero or more bits long and must be all zero; this field may be used to fill out a message to a word boundary.



The message header must, among other things, enable the NCP at the receiving Host to identify correctly the connection over which the message was sent. Given a set of messages from Host A to Host B, the only field in the header under the control of the NCP at Host B is the link number (assigned via the RTS control command). Therefore, each NCP must insure that, at a given point in time, for each connection for which it is the receiver, a unique link is assigned. Recall that the link is specified by the sender's address and the link number; thus a unique link number must be assigned to each connection to a given Host.

LINK ASSIGNMENT

Links are assigned as follows:

Decimal	Octal	Use
0	0	Control Messages
1	1	Reserved
2-71	2-107	Regular Messages
72-151	110-227	Reserved
152	230	PARC Universal Protocol
153	231	TIP Status Reporting
154	232	TIP Accounting
155-158	233-236	Internet Protocol
159-191	237-277	Measurements
192-195	300-303	Message Switching Protocol
196-255	304-377	Experimental Protocols

CONTROL MESSAGES

Messages sent over the control link have the same format as other Host-to-Host messages. The connection byte size (Field S in the message header) must be 8. Control messages may not contain more than 120 bytes of text; thus the value of the byte count (Field C in the message header) must be less than or equal to 120.

Control messages must contain an integral number of control commands. A single control command may not be split into parts which are transmitted in different control messages.

CONTROL COMMANDS

Each control command begins with an 8-bit opcode. These opcodes have values of 0, 1, ... to permit table lookup upon receipt. Private experimental protocols should be tested using opcodes of 255, 254, Most of the control commands are more fully explained in Section III.

NOP - No operation

8

! NOP !

The NOP command may be sent at anytime and should be discarded by the receiver. It may be useful for formatting control messages.

RST - Reset

8

! RST !

The RST command is used by one Host to inform another that all information regarding previously existing connections, including partially terminated connections, between the two Hosts should be purged from the NCP tables of the Host receiving the RST. Except for responding to RSTs, the Host which sent the RST is not obliged to communicate further with the other Host until an RRP is received in response.

RRP - Reset reply

8

! RRP !

The RRP command must be sent in reply to an RST command.

RTS - Request connection, receiver to sender

8

32

32

8

! RTS ! receive socket ! send socket ! link !

The RTS command is used to establish a connection and is sent from the Host containing the receive socket to the Host containing the send socket. The link number for message transmission over the connection is assigned with this command; the "link" field must be between 2 and 71, inclusive.

STR - Request connection, sender to receiver

8	32	32	8

! STR !	send socket	! receive socket	! link !

The STR command is used to establish a connection and is sent from the Host containing the send socket to the Host containing the receive socket. The connection byte size is assigned with this command; the size must be between 1 and 255, inclusive.

CLS - Close

8	32	32

! CLS !	my socket	! your socket !

The CLS command is used to terminate a connection. A connection need not be completely established before a CLS is sent.

ALL - Allocate

8	8	16	32

! ALL !	link ! msg space !	bit space	!

The ALL command is sent from a receiving Host to a sending Host to increase the sending Host's space counters. This command may be sent only while the connection is established. The receiving Host is prohibited from incrementing the sending Host's message counter above (2 to the 16th minus 1) or bit counter above (2 to the 32nd minus 1).

GVB - Give back

8	8	8	8

! GVB !	link ! f(m) ! f(b) !		

!	!		
		!-----bit fraction	
			!-----message fraction

The GVB command is sent from a receiving Host to a sending Host to request that the sending Host return all or part of its Message space and/or bit space allocations. The "fractions" specify what portion (in 128ths) of each allocation must be returned. This command may be sent only while the connection is established.

RET - Return

8	8	16	32
<hr/>			
! RET	! link	! msg space	! bit space
<hr/>			

The RET command is sent from the sending Host to the receiving Host to return all or a part of its message space and/or bit space allocations in response to a GVB command. This command may be sent only while the connection is established.

INR - Interrupt by receiver

8	8
<hr/>	
! INR	! link
<hr/>	

The INR command is sent from the receiving Host to the sending Host when the receiving process wants to interrupt the sending process. This command may be sent only while the connection is established.

INS - Interrupt by sender

8	8
<hr/>	
! INS	! link
<hr/>	

The INS command is sent from the sending Host to the receiving Host when the sending process wants to interrupt the receiving process. This command may be sent only while the connection is established.

ECO - Echo request

8	8
<hr/>	
! ECO	! data
<hr/>	

The ECO command is used only for test purposes. The data field may be any bit configuration convenient to the Host sending the ECO command.

ERP - Echo reply

8 8

! ERP ! data !

The ERP command must be sent in reply to an ECO command. The data field must be identical to the data field in the incoming ECO command.

ERR - Error detected

8 8 80

-----/ /----
! ERR ! code ! data !
-----/ /----

The ERR command MAY be sent whenever a second level protocol error is detected in the input from another Host. In the case that the error condition has a predefined error code, the "code" field specifies the specific error, and the data field gives parameters. For other errors the code field is zero and the data field is idiosyncratic to the sender. Implementers of Network Control Programs are expected to publish timely information on their ERF commands.

The usefulness of the ERR command is compromised if it is merely discarded by the receiver. Thus, sites are urged to record incoming ERRs if possible, and to investigate their cause in conjunction with the sending site. The following codes are defined. Additional codes may be defined later.

a. Undefined (Error code = 0)

The "data" field is idiosyncratic to the sender.

b. Illegal opcode (Error code = 1)

An illegal opcode was detected in a control message. The "data" field contains the ten bytes of the control message beginning with the byte containing the illegal opcode. If the remainder of the control message contains less than ten bytes, fill will be necessary; the value of the fill is zeros.

c. Short parameter space (Error code = 2)

The end of a control message was encountered before all the required parameters of the control command being decoded were found. The "data" field contains the command in error; the value of any fill necessary is zeros.

d. Bad parameters (Error code = 3)

Erroneous parameters were found in a control command. For example, two receive or two send sockets in an STR, RTS, or CLS; a link number outside the range 2 to 71 (inclusive); an ALL containing a space allocation too large. The "data" field contains the command in error; The value of any fill necessary is zeros.

e. Request on a non-existent socket (Error code = 4)

A request other than STR or RTS was made for a socket (or link) for which no RFC has been transmitted in either direction. This code is meant to indicate to the NCP receiving it that functions are being performed out of order. The "data" field contains the command in error; the value of any fill necessary is zeros.

f. Socket (link) not connected (Error code = 5)

There are two cases:

1. A control command other than STR or RTS refers to a socket (or link) which is not part of an established connection. This code would be used when one RFC had been transmitted, but the matching RFC had not. It is meant to indicate the failure of the NCP receiving it to wait for a response to an RFC. The "data" field contains the command in error; the value of any fill necessary is zeros.

2. A message was received over a link which is not currently being used for any connection. The contents of the "data" field are the message header followed by the first eight bits of text (if any) or zeros.

OPCODE ASSIGNMENT

Opcodes are defined to be eight-bit unsigned binary numbers. The values assigned to opcodes are:

NOP	=	0
RTS	=	1
STR	=	2
CLS	=	3
ALL	=	4
GVB	=	5
RET	=	6
INR	=	7
INS	=	8
ECO	=	9
ERP	=	10
ERR	=	11
RST	=	12
RRP	=	13

Control Command Summary

8				

! NOP !				

8	32		32	8

! RTS !	receive socket	!	send socket	! link !

8	32		32	8

! STR !	send socket	!	receive socket	! link !

8	32		32	

! CLS !	my socket	!	your socket	!

8	8	16		32

! ALL ! link ! msg space !			bit space	!

8	8	8	8	

! GVB ! link ! f(m) ! f(b) !				

8	8	16		32

! RET ! link ! msg space !			bit space	!

8	8			

! INR ! link !				

8	8			

! INS ! link !				

8 8

! ECO ! data !

8 8

! ERP ! data !

8 8 80

----- / / -----

! ERR ! code ! data !

----- / / -----

8

! RST !

8

! RRP !

INITIAL CONNECTION PROTOCOLS

Preceding page blank

Protocol Specification
NIC #7101

J. Postel
UCLA - NMC
Computer Science
11 June 71

Official Initial Connection Protocol

DOCUMENT #2

This document specifies the third level protocol used to connect a user process at one site with a server process at another site. In one instance, the user process will be a Telnet and the server process will be a Logger.

This document describes a family of Initial Connection Protocols (ICP's) suitable for establishing one pair of connections between any user process and any server process. The description will be at two levels, the third or user level, and the second or NCP level.

Third Level Description

Notation

There is no standard notation for describing system calls which initiate and close connections or cause data to be sent, so the following *ad hoc* notation will be used.

Init (local = ℓ , foreign = f , size = s)

causes the local Host to attempt to establish a connection between socket ℓ at the local Host and socket f , with a byte size of s for the connection.

ℓ is a 32 bit local socket number.

f is a 40 bit foreign socket number, the high-order eight bits of which specify the foreign Host, and

s is an eight bit non-zero byte size.

The sum of ℓ and f must be odd.

Listen (local = ℓ , size = s)

causes the local Host to wait for a request for connection to local socket ℓ with byte size s . The process will be awakened when a connection is established. The parameters ℓ and s are the same as for Init.

Official Initial Connection Protocol
NIC 7101 (June 11, 1971)

Protocol Specification
NIC #7101

J. Postel
UCLA - NMC
Computer Science
11 June 71

Send (socket = ℓ , data = d)

The data named by d is sent over the connection attached to local socket ℓ . ℓ must be a send socket attached to a connection. d is the name of a data area.

Receive (socket = ℓ , data = d)

The receive side counterpart to send.

Close (socket = ℓ)

Any connection currently attached to local socket ℓ is closed.

A Family of ICP's

Briefly, a server process at a site attaches a well-advertised send socket L and listens. A user process initiates connection to L from its receive socket U. The byte size for this connection is 32. The server process then transmits a 32-bit even number S and closes the connection. The 32-bit number S and its successor, S+1, are the socket numbers the server will use. The final steps are for sockets S and S+1 at the server site to be connected to sockets U+3 and U+2 respectively at the user site.

Using the notation, the server executes the following sequence:

```
Listen (local = L, size = 32)
[Wait until a user connects]
Send (socket = L, data = S)
Close (socket = L)
Init (local = S, foreign = U+3, size = Bu)
Init (local = S+1, foreign = U+2, size = Bs)
```

The user executes the following:

```
Init (local = U, foreign = L, size = 32)
Receive (socket = U, data = S)
Optional Close (socket = U)
```

Protocol Specification
NIC #7101

J. Postel
UCLA - NMC
Computer Science
11 June 71

```
Listen (local = U+3, size = Bu)
or
Init (local = U+3, foreign = S, size = Bu)

Listen (local = U+2, size = Bs)
or
Init (local = U+2, foreign = S+., size = Bs)
```

Note that L is a send socket (odd), while S and U are receive sockets (even). Where L, S or U are used as values of *local*, they are 32-bit numbers; where they are values of *foreign*, they are 40-bit numbers. The parameters B_s and B_u are the byte sizes to be sent by the server and user, respectively. If the user side declines to close socket U, then it must be handled automatically by the second level. (i.e. the NCP must send a matching CLS when it receives a CLS).

Examination of the above sequences reveals that an ICP is characterized by three numbers L, B_s and B_u , and must meet the restrictions that

- (a) L is a send socket,
- (b) B_s and B_u are legal byte sizes, and
- (c) for each L there is only one pair of associated byte sizes.

This last restriction prevents two distinct services from being available through the same socket and distinguished only by the byte sizes.

Second Level Description

Notation

The following notation will be used for the NCP Control Command used in ICP.

STR(ls, fs, s)

ls = local send socket
fs = foreign receive socket
s = byte size

Official Initial Connection Protocol
NIC 7101 (June 11, 1971)

Protocol Specification
NIC #7101

J. Postel
UCLA - NMC
Computer Science
11 June 71

RTS(ls, fs, l)

ls = local receive socket
fs = foreign send socket
l = link

ALL(l, m, b)

l = link
m = message allocation
b = bit allocation

CLS(ls, fs)

ls = local socket
fs = foreign socket

The same family of ICP's is now described again.

Server

- S1: Listen on socket L.
- S2: Wait for a match.
- S3: STR(L, U, s₁)
- S4: Wait for allocation.
- S5: Send data S in s₁ bit bytes as allowed by allocation m₁, b₁.
- S6: CLS(L, U)
- S7: RTS(S, U+3, l₂)
- S8: STR(S+1, U+2, s₃)

User

- U1: RTS(U, L, l₁).
- U2: Wait for match.
- U3: ALL(l₁, m₁, b₁)
- U4: Receive data S in s₁ bit bytes.
- U5: CLS(U, L)
- U6: STR(U+3, S, s₂)
- U7: RTS(U+2, S+1, l₃)

The labels here imply no ordering except that ordering required by the Host-Host Protocol. Note that steps S7 and S8 can be reversed as can U6 and U7. Also, notice that at any time after S2 the server could initiate steps S7 and S8 in parallel with steps S3 through S6, and that at any time after U4 the user could initiate steps U6 and U7 in parallel with step U5.

Protocol Specification
NIC #7101

J. Postel
UCLA - NMC
Computer Science
11 June 71

Following the above exchanges ALL commands would be exchanged and data transfers could begin.

At this level the parameters of the above ICP family are L , m_1 , b_1 , δ_1 δ_2 δ_3 ℓ_1 ℓ_2 ℓ_3 .

L is a well known socket number and will be specified for each type of service.

m_1 and b_1 are allocation quantities for the transfer of a socket number.

m_1 is specified to be at least 1.

b_1 is specified to be at least 32.

δ_1 , δ_2 and δ_3 are byte sizes. Only δ_1 is to be specified as δ_2 and δ_3 are to be left to the process involved.

δ_1 is specified to be 32.

ℓ_1 , ℓ_2 , and ℓ_3 are links and are not specified.

It is legal for the NCP to receive RTS or STR before the corresponding local Init or Listen is issued.

7155

Network Working Group
 Request for Comments: #202
 NIC #7155
 Categories: D
 References: Document #2
 Obsoletes: None

Steve Wolfe
 UCLA-CCN
 Jon Postel
 UCLA-NMC
 26 July 71

We have noticed a possible deadlock situation which may arise using the Initial Connection Protocol (ICP) specified in Document #2 (NIC #7101 in the Current Network Protocols Notebook NIC #7104).

If on both sides one RFC is issued and a "wait for match" is required before the second RFC is issued, it is possible that the first RFC's will not match. In particular a deadlock will occur if both sides open their send or both sides open their receive sockets first.

Briefly the ICP is:

<u>Server</u>	<u>User</u>
S1: Listen on socket L.	U1: RTS(U, L, $\underline{\ell}_1$).
S2: Wait for a match	U2: Wait for match.
S3: STR(L, U, \underline{s}_1)	
S4: Wait for allocation.	U3: ALL($\underline{\ell}_1$, \underline{m}_1 , \underline{b}_1)
S5: Send data S in \underline{s}_1 bit bytes as allowed by allocation \underline{m}_1 , \underline{b}_1 .	U4: Receive data S in \underline{s}_1 bit bytes
S6: CLS(L, U)	U5: CLS(U, L)
S7: RTS(S, U+3, $\underline{\ell}_2$)	U6: STR(U+3, S, \underline{s}_2)
S8: STR(S+1, U+2, \underline{s}_3)	U7: RTS(U+2, S+1, $\underline{\ell}_3$)

"The labels here imply no ordering except that ordering required by the Host-Host Protocol. Note that steps S7 and S8 can be reversed as can U6 and U7. Also, notice that at any time after S2 the server could initiate steps S7 and S8 in parallel with steps S3 through S6, and that at any time after U4 the user could initiate steps U6 and U7 in parallel with step U5."

We recommend that the server perform steps 7 and 8 before waiting for the user to perform step 6 or 7. It is also suggested that the user issue the RFC's in steps 6 and 7 without waiting for the server. (If the user is only Listening then both Listens should be issued without waiting for the server.) If for some reason a host must delay between issuing RFC's it must issue the RFC's involving sockets S and U+3 first.

(47)

Preceding page blank

Protocol Specification
NIC #7103

J. Postel
UCLA - NMC
Computer Science
15 June 71

Official Telnet-Logger Initial Connection Protocol

DOCUMENT #3

For connecting Telnet and Logger processes, the ICP parameters are L=1, $B_u = \underline{\Delta}_2 = 8$, and $B_s = \underline{\Delta}_3 = 8$. (To clarify the socket number required, L = X'00000001').

Preceding page blank

TELNET PROTOCOLS

Preceding page blank

TELNET Protocol Specification

INTRODUCTION

The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

GENERAL CONSIDERATIONS

A TELNET connection consists of a pair of standard Host/Host Protocol connections over which passes data with interspersed TELNET control information. The pair of connections are typically established by the Initial Connection Protocol. Details on the Host/Host and Initial Connection Protocols may be found in NIC #7104.

The TELNET Protocol is built upon three main ideas: first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

1) When a TELNET connection is first established, each end is assumed to originate and terminate at a "Network Virtual Terminal", or NVT. An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. This eliminates the need for "server" and "user" Hosts* to keep information about the characteristics of each other's terminals and terminal handling conventions. All Hosts, both user and server, map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing Hosts a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals).

*Note: The "user" Host is the Host to which the physical terminal is normally attached, and the "server" host is the Host which is normally providing some service. As an alternate point of view, applicable even in terminal-to-terminal or process-to-process communications, the "user" Host is the Host which initiated the communication.

2) The principle of negotiated options takes cognizance of the fact that many sites will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, service. Independent of, but structured within, the TELNET Protocol various "options" will be sanctioned which can be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow a user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their TELNET connection. Such options could include changing the character set, the echo mode, the line width, the page length, etc.

The basic strategy for setting up the use of options is to have either party (or both) initiate a REQUEST that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse a request to disable, some option since all parties must be prepared to support the NVT.

The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops--each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

- a) Parties may only request a change in option status; i.e., a party may not send out a "request" merely to announce what mode it is in.
- b) If a party receives what appears to be a request to enter some mode it is already in, the request should NOT be acknowledged.
- c) Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take effect. (It should be noted that some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a

site may wish to buffer data, after requesting an option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.)

Option requests are likely to flurry back and forth when a TELNET connection is first established, as each party attempts to get the best possible service from the other party. Beyond that, however, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as will be explained later, uses a transmission discipline well suited to the many "line at a time" applications such as BASIC, but poorly suited to the many "character at a time" applications such as NLS. A server might elect to devote the extra processor overhead required for a "character at a time" discipline when it was suitable for the local process and would negotiate an appropriate option. However, rather than then being permanently burdened with the extra processing overhead, it could switch (i.e., negotiate) back to NVT when the more "taut" control was no longer necessary.

It is possible for requests initiated by processes to simulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something CHANGES. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention.

Option designers should not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options--since it is correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use "DO, DON'T, WILL, WON'T" to establish that both parties UNDERSTAND the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length--such a "sub-negotiation" perhaps including fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that such expanded negotiations should never begin until some prior (standard) negotiation has

established that both parties are capable of parsing the expanded syntax.

In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing OPTION XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all Hosts may implement their TELNET processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood.

As much as possible, the TELNET protocol has been made server-user symmetrical so that it easily and naturally covers the user-user (linking) and server-server (cooperating processes) cases. It is hoped, but not absolutely required, that options will further this intent. In any case, it is explicitly acknowledged that symmetry is an operating principle rather than an ironclad rule.

A companion document, "TELNET Option Specifications," should be consulted for information about the procedure for establishing new options. That document, as well as descriptions of all currently defined options, is contained in the TELNET section of Current Network Protocols (NIC #7104).

THE NETWORK VIRTUAL TERMINAL

The Network Virtual Terminal (NVT) is a bi-directional character device. The NVT has a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the TELNET connection and, if "echos" are desired, to the NVT's printer as well. "Echos" will not be expected to traverse the network (although options exist to enable a "remote" echoing mode of operation, no Host is required to implement this option). The code set is seven-bit USASCII in an eight-bit field), except as modified herein. Any code conversion and timing considerations are local problems and do not affect the NVT.

Transmission of Data

Although a TELNET connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until

options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the TELNET connection:

- 1) Insofar as the availability of local buffer space permits, data should be accumulated in the Host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs. This signal could be generated either by a process or by a human user.

The motivation for this rule is the high cost, to some Hosts, of processing network input interrupts, coupled with the default NVT specification that "echos" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore implementers are cautioned to provide methods of locally signalling that all buffered data should be transmitted immediately.

- 2) When a process has completed sending data to an NVT printer AND has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a TELNET connection cannot proceed without input from the other end), the process must transmit the TELNET Go Ahead (GA) command.

This rule is NOT intended to require that the TELNET GA command be sent from a terminal at the end of each line, since server Hosts do not normally require a special signal (in ADDITION to end-of-line or other locally-defined characters) in order to commence processing. Rather, the TELNET GA is designed to help a user's local Host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741. A description of this type of terminal may help to explain the proper use of the GA command.

The terminal-computer connection is always under control of either the user or the computer. Neither can unilaterally seize control from the other; rather the controlling end must relinquish its control explicitly. At the terminal end, the hardware is constructed so as to relinquish control each time that a "line" is terminated (i.e., when the "New Line" key is typed by the user). When this occurs, the attached (local)

computer processes the input data, decides if output should be generated, and if not returns control to the terminal. If output should be generated, control is retained by the computer until all output has been transmitted.

The difficulties of using this type of terminal through the network should be obvious. The "local" computer is no longer able to decide whether to retain control after seeing an end-of-line signal or not; this decision can only be made by the "remote" computer which is processing the data. Therefore, the TELNET GA command provides a mechanism whereby the "remote" (server) computer can signal the "local" (user) computer that it is time to pass control to the user of the terminal. It should be transmitted at those times, and only at those times, when the user should be given control of the terminal. Note that premature transmission of the GA command may result in the blocking of output, since the user is likely to assume that the transmitting system has paused, and therefore he will fail to turn the line around manually.

The foregoing, of course, does not apply to the user-to-server direction of communication. In this direction, GAs may be sent at any time, but need not ever be sent. Also, if the TELNET connection is being used for process-to-process communication, GAs need not be sent in either direction. Finally, for terminal-to-terminal communication, GAs may be required in neither, one, or both directions. If a Host plans to support terminal-to-terminal communication it is suggested that the Host provide the user with a means of manually signalling that it is time for a GA to be sent over the TELNET connection; this, however, is not a requirement on the implementer of a TELNET process.

Standard Representation of Control Functions

As stated in the Introduction to this document, the primary goal of the TELNET protocol is the provision of a standard interfacing of terminal devices and terminal-oriented processes through the network. Early experiences with this type of interconnection have shown that certain functions are implemented by most servers, but that the methods of invoking these functions differ widely. For a human user who interacts with several server systems, these differences are highly frustrating. TELNET, therefore, defines a standard representation for five of these functions, as described below. These standard representations have standard, but not required, meanings (with the exception that the IP function may be required by other protocols which use TELNET); that is, a system which does not provide the function to local users need not

provide it to network users and may treat the standard representation for the function as a No-operation. On the other hand, a system which does provide the function to local users is obliged to provide the same function a network user who transmits the standard representation for the function.

INTERRUPT PROCESS (IP) - Many systems provide a function which suspends, interrupts, aborts, or terminates the operation of a user process. This function is frequently used when a user believes his process is in an unending loop, or when an unwanted process was inadvertently activated. IP is the standard representation for invoking this function. It should be noted by implementers that IP may be REQUIRED by other protocols which use TELNET, and therefore should be implemented if these other protocols are to be supported.

ABORT OUTPUT (AO) - Many systems provide a function which allows a process, which is generating output, to run to completion (or to reach the same stopping point it would reach if running to completion) but without sending the output to the user's terminal. Further, this function typically clears any output already produced but not yet actually printed (or displayed) on the user's terminal. AO is the standard representation for invoking this function. For example, some subsystem might normally accept a user's command, send a long text string to the user's terminal in response, and finally signal readiness to accept the next command by sending a "prompt" character (preceded by <CR><LF>) to the user's terminal. If the AO were received during the transmission of the text string, a reasonable implementation would be to suppress the remainder of the text string, but transmit the prompt character and the preceding <CR><LF>. (This is possibly in distinction to the action which might be taken if an IP were received; the IP might cause suppression of the text string and an exit from the subsystem.)

It should be noted, by systems which provide this function, that there may be buffers external to the system (in the network and the user's "local" Host) which should be cleared; the appropriate way to do this is to transmit the "Synch" signal described below.

ARE YOU THERE (AYT) - Many systems provide a function which provides the user with some visible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of the unanticipated (by the user) length of a computation, an unusually heavy system load, etc. AYT is the standard representation for invoking this function.

ERASE CHARACTER (EC) - Many systems provide a function which deletes the last preceding undeleted character or "print position"* from the stream of data being supplied by the user. This function is typically used to edit keyboard input when typing mistakes are made. EC is the standard representation for invoking this function.

ERASE LINE (EL) - Many systems provide a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input. EL is the standard representation for invoking this function.

*Note: A "print position" may contain several characters which are the result of overstrikes, or of sequences such as <char1> BS <char2>...

The TELNET "Synch" Signal

Most time-sharing systems provide mechanisms which allow a terminal user to regain control of a "runaway" process; the IP and AO functions described above are examples of these mechanisms. Such systems, when used locally, have access to all of the signals supplied by the user, whether these are normal characters or special "out of band" signals such as those supplied by the teletype "BREAK" key or the IBM 2741 "ATTN" key. This is not necessarily true when terminals are connected to the system through the network; the network's flow control mechanisms may cause such a signal to be buffered elsewhere, for example in the user's Host.

To counter this problem, the TELNET "Synch" mechanism is introduced. A Synch signal consists of a Host/Host Protocol INS command, coupled with the TELNET command DATA MARK. The INS command, which is not subject to the flow control pertaining to the TELNET connections, is used to invoke special handling of the data stream by the process which receives it. In this mode, the data stream is immediately scanned for "interesting" signals as defined below, discarding intervening data. The TELNET command DATA MARK (DM) is the synchronizing mark in the data stream which indicates that any special signal has already occurred and the recipient can return to normal processing of the data stream. When a DM arrives before its associated INS, the recipient should not process the data stream further until the matching INS is received, in order to insure that the two ends of the connection remain synchronized. Also, implementers are warned that in some cases several Synch's may be sent in succession. In general, this will require a count of the INS's received so as to properly pair them with the associated DM's. "Interesting" signals are defined

to be: the TELNET standard representations of IP, AO, and AYT (but not EC or EL); the local analogs of these standard representations (if any); all other TELNET commands; other site-defined signals which can be acted on without delaying the scan of the data stream.

Since one effect of the SYNCH mechanism is the discarding of essentially all characters (except TELNET commands) between the sender of the Synch and its recipient, this mechanism is specified as the standard way to clear the data path when that is desired. For example, if a user at a terminal causes an AO to be transmitted, the server which receives the AO (if it provides that function at all) should return a Synch to the user.

Finally, just as the NCP-level INS command is needed at the TELNET level as an out-of-band signal, so other protocols which make use of TELNET may require a TELNET command which can be viewed as an out-of-band signal at a different level.

By convention the sequence [IP, Synch] is to be used as such a signal. For example, suppose that some other protocol, which uses TELNET, defines the character string STOP analogously to the TELNET command AO. Imagine that a user of this protocol wishes a server to process the STOP string, but the connection is blocked because the server is processing other commands. The user should instruct his system to:

1. Send the TELNET IP character;
2. Send the TELNET SYNC sequence, that is:
 - a. Send the TELNET Data Mark (DM);
 - b. Send the Host-Host Protocol INS;
3. Send the character string STOP; and
4. Send the other protocol's analog of the TELNET DM (if any).

The user (or process acting on his behalf) must transmit the TELNET SYNCH sequence of step 2 above to ENSURE that the TELNET IP gets through to the server's TELNET interpreter.

The NVT Printer and Keyboard

The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 USASCII graphics (codes 32 through 126). Of the 33 USASCII control codes (0 through 31 and 127), and the 128 uncovered codes (128 through 255), the following have specified meaning to the NVT printer:

Name	Code	Meaning
NULL (NUL)	0	A no operation.
Line Feed (LF)	10	Moves the printer to the next print line, keeping the same horizontal position.
Carriage Return (CR)	13	Moves the printer to the left margin of the current line.

In addition, the following codes shall have defined, but not required, effects on the NVT printer. Neither end of a TELNET connection may assume that the other party will take, or will have taken, any particular action upon receipt or transmission of these:

BELL (BEL)	7	Produces an audible or visible signal (which does NOT move the print head).
Back Space (BS)	8	Moves the print head one character position toward the left margin.
Horizontal Tab (HT)	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Vertical Tab (VT)	11	Moves the printer to the next vertical tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Form Feed (FF)	12	Moves the printer to the top of the next page, keeping the same horizontal position.

All remaining codes do not cause the NVT printer to take any action.

The sequence "CR LF", as defined, will cause the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). However, many systems and terminals do not treat CR and LF independently, and will have to go to some effort to simulate their effect. (For example, some terminals do not have a CR independent of the LF, but on such terminals it may be possible to simulate a CR by backspacing.) Therefore, the sequence "CR LF" must be treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" must be used where a carriage return alone is actually desired; and the CR character must be avoided in other contexts. This rule gives assurance to systems which must decide whether to perform a "new line" function or a multiple-backspace that the TELNET stream contains a character following a CR that will allow a rational decision.

The NVT keyboard has keys, or key combinations, or key sequences, for generating all 128 USASCII codes. Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

In addition to these codes, the NVT keyboard shall be capable of generating the following additional codes which, except as noted, have defined, but not required, meanings. The actual code assignments for these "characters" are in the TELNET Command section, because they are viewed as being, in some sense, generic and should be available even when the data stream is interpreted as being some other character set.

Synch

This key allows the user to clear his data path to the other party. The activation of this key causes a DM (see command section) to be sent in the data stream and an INS to be sent on the control link. The pair DM-INS is to have required meaning as defined previously.

Break (BRK)

This code is provided because it is a signal outside the USASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym for the IP standard representation.

Interrupt Process (IP)

Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use TELNET.

Abort Output (AO)

Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a Synch to the user.

Are You There (AYT)

Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

Erase Character (EC)

The recipient should delete the last preceding undeleted character or "print position" from the data stream.

Erase Line (EL)

The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the TELNET connection.

The spirit of these "extra" keys, and also the printer format effectors, is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local". Just as the NVT data byte 104 should be mapped into whatever the local code for "uppercase D" is, so the EC character should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 174 is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. Similarly for format effectors: if the terminal actually DOES have a "Vertical tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.

TELNET COMMAND STRUCTURE

All TELNET commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option

referenced. This format was chosen so that as more comprehensive use of the "data space" is made--by negotiations from the basic NVT, of course--collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and inefficiency, of "escaping" the data bytes into the stream. With the current set-up, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

The following are the defined TELNET commands. Note that these codes and code sequences have the indicated meaning ONLY when immediately preceded by an IAC.

Name	Code	Meaning
SE	240	End of subnegotiation parameters
NOP	241	No operation
Data Mark	242	The data stream portion of a Synch. This should always be accompanied by an INS on the control link
Break	243	NVT character BRK
Interrupt Process	244	The function IP
Abort Output	245	The function AO
Are You There	246	The function AYT
Erase Character	247	The function EC
Erase Line	248	The function EL
Go Ahead	249	The GA signal
SB (option code)	250	Indicates that what follows is subnegotiation of the indicated option
WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option

TELNET Protocol Specification
NIC 18639 (Aug. 1973)

WON'T (option code)	252	Indicates the refusal to perform or continue performing, the indicated option
DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option
DON'T (option code)	254	Indicates the demand that the other party stop performing, or confirmation that you are no longer expecting the other party to perform the indicated option
IAC	255	Data byte 255

TELNET Options Specifications

The intent of providing for options in the TELNET Protocol is to permit sites to obtain more elegant solutions to the problems of communication between dissimilar devices than is possible within the framework provided by the Network Virtual Terminal (NVT). It should be possible for sites to invent, test, or discard options at will, even though the negotiation method permits the direct use of only 256 option codes. Nevertheless, it is envisioned that options which prove to be generally useful will eventually be supported by many sites; therefore it is desirable that options should be carefully documented and well publicized. In addition, it is necessary to insure that a single option code is not used for several different options.

This document specifies a method of option code assignment and standards for documentation of options. The individual responsible for assignment of option codes may waive the requirement for complete documentation for some cases of experimentation, but in general documentation will be required prior to code assignment. Options will be publicized by including their documentation in the TELNET section of Current Network Protocols (NIC #7104); inventors of options may, of course, publicize them in other ways as well.

Option codes will be assigned by:

Jonathan B. Postel
Stanford Research Institute
Augmentation Research Center
333 Ravenswood Avenue
Menlo Park, California 94025
(415) 326-6200 ext. 3718

SNDMSG mailbox = POSTEL@BBNB

NLS Sendmail Ident = JBP

Documentation of options should contain at least the following sections:

Section 1 - Command name (and option code).

Section 2 - Command meanings (definitions).

The meaning of each possible TELNET command relevant to this option should be described. Note that for complex options, where "subnegotiation" is required, there may be a large number of possible commands. The concept of "subnegotiation" is described in more detail below.

Section 3 - Default specification.

The default assumptions for sites which do not choose to implement, or use, the option must be described.

Section 4 - Motivation.

A detailed explanation of the motivation for inventing a particular option, or for choosing a particular form for the option, is extremely helpful to others who are not faced (or don't realize that they are faced) by the problem that the option is designed to solve.

Section 5 - Description (or Implementation Rules).

Merely defining the command meanings and providing a statement of motivation are not always sufficient to insure that two implementations of an option will be able to communicate. Therefore, a more complete description should be furnished in most cases. This description might take the form of text, a sample implementation, hints to implementers, etc.

A Note on "Subnegotiation"

Some options will require more information to be passed between sites than a single option code. For example, any option which requires a parameter is such a case. The strategy to be used consists of two steps: first, both parties agree to "discuss" the parameter(s) and, second, the "discussion" takes place.

The first step, agreeing to discuss the parameters, takes place in the normal manner; one party proposes use of the option by sending a DO (or WILL) followed by the option code, and the other party accepts by returning a WILL (or DO) followed by the option code. Once both parties have agreed to use the option, subnegotiation takes place by using the command SB, followed by the option code, followed by the parameter(s), followed by the command SE. Each party is presumed to be able to parse the parameter(s), since each has indicated that the option is supported (via the initial exchange of WILL and DO). On the other hand, the receiver may locate the end of a parameter string by searching for the SE command (i.e., the string IAC SE), even if the receiver is unable to parse the parameters. Of course, either party may refuse to pursue further subnegotiation at any time by sending a WON'T or DON'T to the other party.

Thus, for option "ABC", which requires subnegotiation, the formats of the TELNET commands are:

IAC WILL ABC

Offer to use option ABC (or favorable acknowledgment of other party's request)

IAC DO ABC

Request for other party to use option ABC (or favorable acknowledgment of other party's offer)

IAC SB ABC <parameters> IAC SE

One step of subnegotiation, used by either party.

Designers of options requiring "subnegotiation" must take great care to avoid unending loops in the subnegotiation process. For example, if each party can accept any value of a parameter, and both parties suggest parameters with different values, then one is likely to have an infinite oscillation of "acknowledgments" (where each receiver believes it is only acknowledging the new proposals of the other). Finally, if parameters in an option "subnegotiation" include a byte with a value of 255, it IS necessary to double this byte in accordance with the general TELNET rules.

TELNET Binary Transmission Option

1. Command name and code.

TRANSMIT-BINARY 0

2. Command meanings.

IAC WILL TRANSMIT-BINARY

The sender of this command REQUESTS permission to begin transmitting, or confirms that it will now begin transmitting, characters which are to be interpreted as 8 bits of binary data by the receiver of the data.

IAC WON'T TRANSMIT-BINARY

If the connection is already being operated in binary transmission mode,* the sender of this command DEMANDS to begin transmitting data characters which are to be interpreted as standard NVT ASCII characters by the receiver of the data. If the connection is not already being operated in binary transmission mode, the sender of this command REFUSES to begin transmitting characters which are to be interpreted as binary characters by the receiver of the data (i.e., the sender of the data demands to continue transmitting characters in its present mode).

IAC DO TRANSMIT-BINARY

The sender of this command REQUESTS that the sender of the data start transmitting, or confirms that the sender of data is expected to transmit, characters which are to be interpreted as 8 bits of binary data (i.e., by the party sending this command.)

IAC DON'T TRANSMIT-BINARY

If the connection is already being operated in binary transmission mode,* the sender of this command DEMANDS that the sender of the data start transmitting characters which are to be interpreted as standard NVT ASCII characters by the receiver of the data (i.e., the party sending this command). If the connection is not already being operated in binary transmission mode, the sender of this command DEMANDS that the sender of data continue transmitting characters which are to be interpreted in the present mode.

*A connection is being operated in binary transmission mode only when one party has requested it and the other has acknowledged it.

3. Default

WON'T TRANSMIT-BINARY

DON'T TRANSMIT-BINARY

i.e., won't switch to binary mode (if not already in it) or switching back to NVT ASCII mode (if presently in binary mode).

4. Motivation for the option.

It is sometimes useful to have available a binary transmission path within TELNET without having to utilize one of the more efficient, higher level protocols providing binary transmission (such as the File Transfer Protocol). The use of the IAC prefix within the basic TELNET protocol provides the option of binary transmission in a natural way, requiring only the addition of a mechanism by which the parties involved can agree to INTERPRET the characters transmitted over a TELNET connection as binary data.

5. Description of the option.

With the binary transmission option in effect, the receiver should interpret characters received from the transmitter which are not preceded with IAC as 8 bit binary data, with the exception of IAC followed by IAC which stands for the 8 bit binary data with the decimal value 255. IAC followed by an effective TELNET command (plus any additional characters required to complete the command) is still the command even with the binary transmission option in effect. IAC followed by a character which is not a defined TELNET command has the same meaning as IAC followed by NOP, although an IAC followed by an undefined command should not normally be sent in this mode.

6. Implementation suggestions.

It is foreseen that implementations of the binary transmission option will choose to refuse some other options (such as the EBCDIC transmission option) while the binary transmission option is in effect. However, if a pair of Hosts can understand being in binary transmission mode simultaneous with being in, for example, echo mode, then it is all right if they negotiate that combination. Some options in combination with the binary transmission option look very useful, such as negotiate line length (i.e., buffer length).

It should be mentioned that the meanings of WON'T and DON'T are dependent upon whether the connection is presently being operated in binary mode or not. Consider a connection operating in, say, EBCDIC mode which involves a system which has chosen not to implement any

knowledge of the binary command. If this system were to receive a DO TRANSMIT-BINARY, it would not recognize the TRANSMIT-BINARY option and therefore would return a WON'T TRANSMIT-BINARY. If the default for the WON'T TRANSMIT-BINARY were always NVT ASCII, the sender of the DO TRANSMIT-BINARY would expect the recipient to have switched to NVT ASCII, whereas the receiver of the DO TRANSMIT-BINARY would not make this interpretation. Thus, we have the rule that when a connection is not presently operating in binary mode, the default (i.e., the interpretation of WON'T and DON'T) is to continue operating in the current mode, whether that is NVT ASCII, EBCDIC, or some other mode. This rule, however, is not applied once a connection is operating in a binary mode (as agreed to by both ends); this would require each end of the connection to maintain a stack, containing all of the encoding-method transitions which had previously occurred on the connection, in order to properly interpret a WON'T or DON'T. Thus, a WON'T or DON'T received after the connection is operating in binary mode causes the encoding method to revert to NVT ASCII.

It should be remembered that a TELNET connection is normally a PAIR of connections, one going each way; operating in binary transmission mode must be negotiated separately for each of the pair of connections, if that is desired.

Implementation of the binary transmission option, as implementations of all other TELNET options, must follow the loop preventing rules given in the General Considerations section of the TELNET Protocol Specification.

Consider now some issues of binary transmission both to and from both a process and a terminal:

a) Binary transmission from a terminal.

The implementer of the binary transmission option should consider how (or whether) a terminal transmitting over a TELNET connection with binary transmission in effect is allowed to generate all eight bit characters, ignoring parity considerations, etc., on input from the terminal.

b) Binary transmission to a process.

The implementer of the binary transmission option should consider how (or whether) all characters are passed to a process receiving over a connection with binary transmission in effect. As an example of the possible problem, TENEX intercepts certain characters (e.g., ETX, the Teletype

TELNET Binary Transmission Option
NIC 15389 (Aug. 1973)

control-C) at monitor level and does not pass them to the process.

c) Binary transmission from a process.

The implementer of the binary transmission option should consider how (or whether) a process transmitting over a connection with binary transmission in effect is allowed to send all eight bit characters with no characters intercepted by the monitor and changed to other characters. An example of such a conversion may be found in the TENEX system where certain non-printing characters are normally converted to a Circumflex (uparrow) followed by a printing character.

d) binary transmission to a terminal.

The implementer of the binary transmission option should consider how (or whether) all characters received over a connection with binary transmission in effect are sent to a local terminal. At issue may be the addition of timing characters normally inserted locally, parity calculations, and any normal code conversion.

TELNET Echo Option

1. Command name and code.

ECHO 1

2. Command meanings.

IAC WILL ECHO

The sender of this command REQUESTS to begin, or confirms that it will now begin, echoing data characters it receives over the TELNET connection back to the sender of the data characters.

IAC WON'T ECHO

The sender of this command DEMANDS to stop, or refuses to start, echoing the data characters it receives over the TELNET connection back to the sender of the data characters.

IAC DO ECHO

The sender of this command REQUESTS that the receiver of this command begin echoing, or confirms the receiver of this command is expected to echo, data characters it receives over the TELNET connection back to the sender.

IAC DON'T ECHO

The sender of this command DEMANDS the receiver of this command stop, or not start, echoing data characters it receives over the TELNET connection.

3. Default.

WON'T ECHO

DON'T ECHO

i.e., no echoing (which may or may not imply local echoing).

4. Motivation for the option.

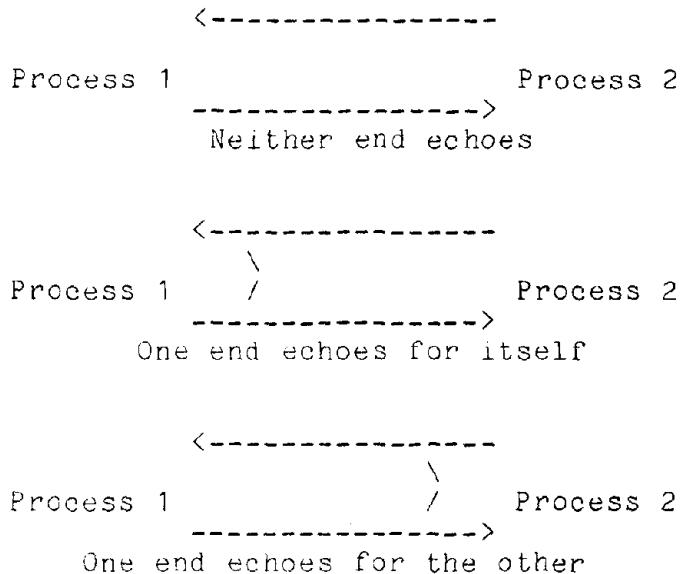
The NVT has a printer and a keyboard which are nominally interconnected so that "echoes" need never traverse the network; that is to say, the NVT nominally operates in a mode where characters typed on the keyboard are (by some means) locally turned around and printed on the printer. In highly interactive situations it is

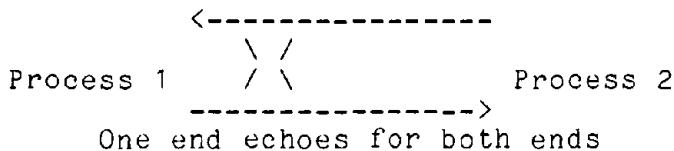
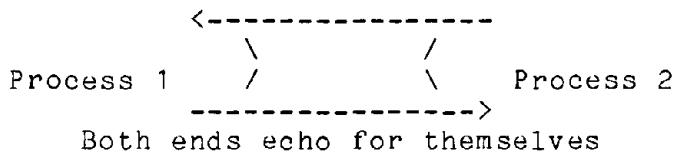
appropriate for the remote process (command language interpreter, etc.) to which the characters are being sent to control the way they are echoed on the printer. In order to support such interactive situations, it is necessary that there be a TELNET option to allow the parties at the two ends of the TELNET connection to agree that characters typed on an NVT keyboard are to be echoed by the party at the other end of the TELNET connection.

5. Description of the option.

When the echoing option is in effect, the party at the end performing the echoing is expected to transmit (echo) data characters it receives back to the sender of the data characters. The option does not require that the characters echoed be exactly the characters received (for example, a number of systems echo the ASCII ESC character with something other than the ESC character). When the echoing option is not in effect, the receiver of data characters should not echo them back to the sender; this, of course, does not prevent the receiver from responding to data characters received.

The normal TELNET connection uses two coupled, simplex connections, one each direction; and neither, either, or both of these simplex connections may be operating simultaneously in echo mode. There are five reasonable modes of operation for echoing on a connection pair:





This option provides the capability to decide on whether or not either end will echo for the other. It does not, however, provide any control over whether or not an end echos for itself; this decision must be left to the sole discretion of the systems at each end (although they may use information regarding the state of "remote" echoing negotiations in making this decision).

It should be noted that if BOTH sites enter the mode of echoing characters transmitted by the other site, then any character transmitted in either direction will be "echoed" back and forth indefinitely. Therefore, care should be taken in each implementation that if one site is echoing, echoing is not permitted to be turned on at the other.

As discussed in the TELNET Protocol Specification, both parties to a full duplex pair of TELNET connections initially assume each simplex connection is being operated in the default mode which is non-echo. If either party desires himself to echo characters to the other party or for the other party to echo characters to him, that party gives the appropriate command (WILL ECHO or DO ECHO) and waits (and hopes) for acceptance of the option. If the request to operate the connection in echo mode is refused, then the connection continues to operate in non-echo mode. If the request to operate the connection in echo mode is accepted, the connection is operated in echo mode. After a connection has been changed to echo mode, either party may demand that it revert to non-echo mode by giving the appropriate DON'T ECHO or WON'T ECHO command (which the other party must confirm thereby allowing the connection to operate in non-echo mode). Just as each of the pair of simplex TELNET connections may be put in remote echoing mode independently, each of the pair of simplex TELNET connections must be removed from remote echoing mode separately.

Implementations of the echo option, as implementations of all other TELNET options, must follow the loop preventing rules given in the

General Considerations section of the TELNET Protocol Specification. Also, so switches between echo and non-echo mode can be made with minimal confusion (momentary double echoing, etc.), switches in mode of operation should be made at times precisely coordinated with the reception and transmission of echo requests and demands. For instance, if one party responds to a DO ECHO with a WILL ECHO, all data characters received after the DO ECHO should be echoed and the WILL ECHO should immediately precede the first of the echoed characters.

The echoing option alone will normally not be sufficient to effect what is commonly understood to be remote computer echoing of characters typed on a terminal keyboard--the SUPPRESS-GO AHEAD option will normally have to be invoked in conjunction with the ECHO option to effect character-at-a-time remote echoing.

6. A Sample Implementation of the Option.

It may be useful to present a sample implementation--the TIP development group suggests the following:

For each user terminal the TIP would keep three state bits: whether the terminal echoes for itself (no ECHO always) or not (ECHO mode possible), whether the (human) user prefers to operate in ECHO mode or in non-ECHO mode, and whether the connection from this terminal to the server is in ECHO or non-ECHO mode. We call these three bits P(physical), D(esired), and A(ctual).

When a terminal dials up the TIP the P-bit is set appropriately, the D-bit is set equal to it, and the A-bit is set to non-ECHO. The P- and A-bits may be manually reset by direct commands if the user so desires; for instance, a user in Hawaii on a 'full-duplex' terminal might know that whatever the preference of a mainland server, because of satellite delay his terminal had better not operate in ECHO mode - he would direct the TIP to change his D-bit from ECHO to non-ECHO.

When a connection is opened from the TIP terminal to a server, the TIP would send the server a DO ECHO command if the MIN (with non-ECHO less than ECHO) of the P- and D-bits is different from the A-bit. If a WON'T ECHO or WILL ECHO arrives from the server, the TIP will set the A-bit to the MIN of the received request, the P-bit, and the D-bit. If this changes the state of the A-bit, the TIP will send off the appropriate acknowledgment; if it does not, then the TIP will send off the appropriate refusal if not changing meant that it had to deny the request (i.e., the MIN of the P-and D-bits was less than the received A-request). If while a connection is open, the TIP terminal user changes either the P- or D-bit, the TIP will repeat the above tests and send off a DO ECHO or DON'T ECHO, if necessary. When the

connection is closed, the TIP would reset the A-bit to indicate no remote echoing.

While the TIP's implementation would not involve DO ECHO or DON'T ECHO commands being sent to the server except when the connection is opened or the user explicitly changes his echoing mode, bigger Hosts might invoke such mode switches quite frequently. For instance, if JOSS, a line at a time system, were running, the server might attempt to put the user in local echo mode by sending the WON'T ECHO command to the user; but while DDT was running, the server might attempt to invoke remote echoing for the user by sending the WILL ECHO command to the user. Furthermore, while the TIP will never send a WILL ECHO command and will only send a WON'T ECHO to refuse a server sent DO ECHO command, a server Host will often send the WILL and WON'T ECHO commands.

TELNET Reconnection Option

1. Command name and code

RCP 2 (prepare to reconnect)

2. Command meanings.

IAC DO RCP

The sender of this command requests the receiver of the command to be prepared to break the TELNET connection with the sender of the command and to re-establish the TELNET connection with some other party (to be specified later).

IAC WILL RCP

The receiver of this command agrees to break its TELNET connection to the sender of the DO RCP command and to re-establish the connection with the party to be specified by the sender of the DO RCP command.

IAC WON'T RCP

The receiver of this command refuses to take part in a reconnection.

IAC DON'T RCP

The sender of this command demands the cancellation of its previous DO RCP command.

IAC SB RCP RCS <host> <socket>

The sender of this command instructs the receiver of the command to transfer this TELNET connection to the place specified by <host> <socket>. The code for RCS is 0.

IAC SB RCP RCW <host> <socket>

The sender of this command instructs the receiver of the command to break the TELNET connection and to await a new TELNET connection from the place specified by <host> <socket>. The code for RCW is 1.

Preceding page blank

3. Default.

WON'T RCP

i.e., no reconnection is allowed.

4. Motivation for the option.

There are situations in which it is desirable to move one or both ends of a communication path from one Host to another.

A. Consider the case of an executive program which TIP users could use to get network status information, send messages, link to other users, etc. Due to the TIP's limited resources the executive program would probably not run on the TIP itself but rather would run on one or more larger Hosts who would be willing to share some of their resources with the TIP (see Figure 1).

The TIP user could access the executive by typing a command such as "@EXEC"; the TIP should then ICP to Host1's executive port. After obtaining the latest network news and perhaps sending a few messages, the user would be ready to log into Host2 (in general not the same as Host1) and do some work. At that point he would like to tell the executive program that he is ready to use Host2 and have the executive hand him off to Host2. To do this the executive program would first interact with Host2, telling it to expect a call from the TIP, and then would instruct the TIP to reconnect to Host2. When the user logs off Host2 he could be passed back to the executive at Host1 preparatory to doing more elsewhere. The reconnection activity would be invisible to the TIP user.

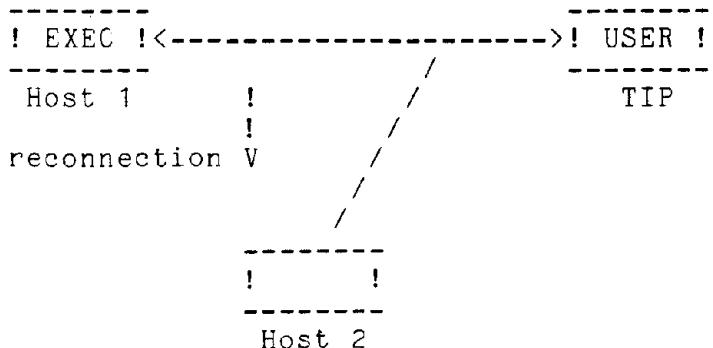


FIGURE 1

B. Imagine a scenario in which a user could use the same name and password (and perhaps account) to log into any server on the network. For reasons of security and economy it would be undesirable to have every name and password stored at every site. A user wanting to use a Host that doesn't have his name or password locally would connect to it and attempt to log in as usual (see Figure 2). The Host, discovering that it doesn't know the user, would hand him off to a network authentication service which can determine whether the user is who he claims to be. If the user passes the authentication test he can be handed back to Host which can then provide him service.

If the user doesn't trust the Host and is afraid that it might read his password rather than pass him off to the Authenticator he could connect directly to the authentication service. After authentication, the Authenticator can pass him off to the Host.

The idea is that the shuffling of the user back and forth between Host and Authenticator should be invisible to the user.

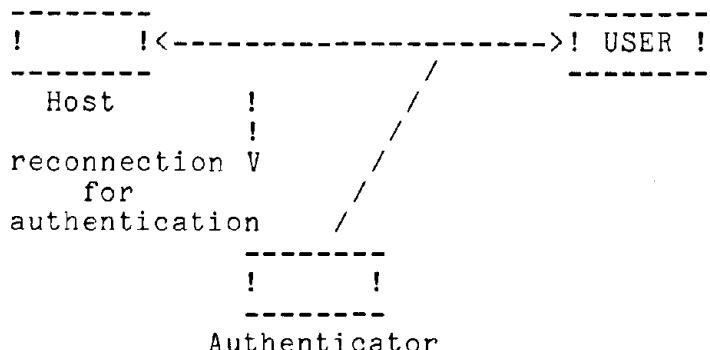


FIGURE 2a

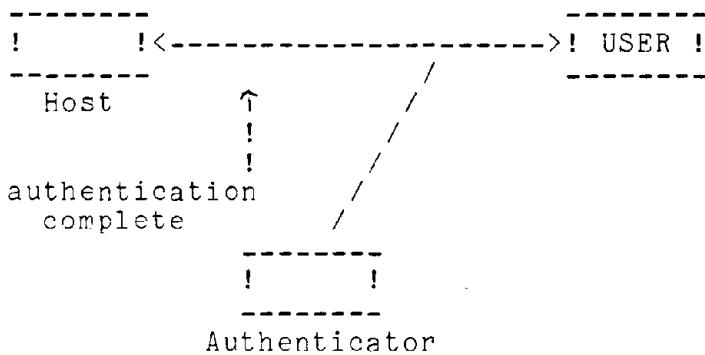


FIGURE 2b

C. The McROSS air traffic simulation system (see 1972 SJCC paper by Thomas) already supports reconnection. It permits an on-going simulation to reconfigure itself by allowing parts to move from computer to computer. For example, in a simulation of air traffic in the Northeast the program fragment simulating the New York Enroute air space could move from Host2 to Host5 (see figure 3). As part of the reconfiguration process the New York Terminal area simulator and Boston Enroute area simulators break their connections with the New York Enroute simulator at Host2 and reconnect to it at Host5.

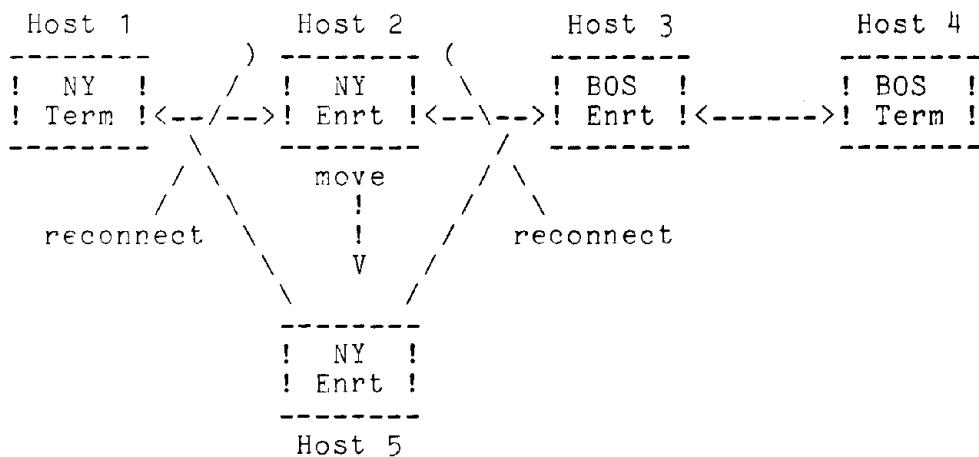


FIGURE 3

5. An abstract description of a reconnection mechanism.

The reconnection mechanism includes four (abstract) commands:

Reconnect Request: RRQ <path>
Reconnect OK: ROK <path>
Reconnect No: RNO <path>
Reconnect Do: RDO <path> <destination>

where <path> is a communication path to be redirected to <destination>.

Assume that H1 wants to move its end of communication path A-C from itself to port D at H3 (Figure 4).

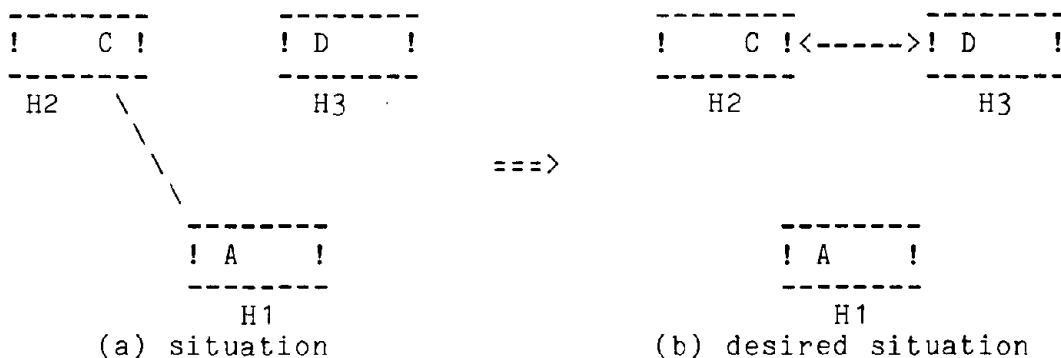


FIGURE 4

The reconnection proceeds by steps:

- a. H1 arranges for the reconnection by sending RRQ to H2:
H1->H2: RRQ (path A-C)
- b. H2 agrees to reconnect and acknowledges with ROK:
H2->H1: ROK (path C-A)
- c. H1 notes that H2 has agreed to reconnect and instructs H2 to perform the reconnection;
H1->H2: RDO (path A-C) (Host H3, PortD)

d. H1 breaks paths A-C.

H2 breaks path C-A and initiates path C-D.

In order for the reconnection to succeed H1 must, of course, have arranged for H3's cooperation. One way H1 could do this would be to establish the path B-D and then proceed through the reconnection protocol exchange with H3 concurrently with its exchange with H2 (See Figure 5):

H1->H3: RRQ (path B-D)
H3->H1: ROK (path D-B)
H1->H3: RDO (path B-D) (Host H2, Port C)

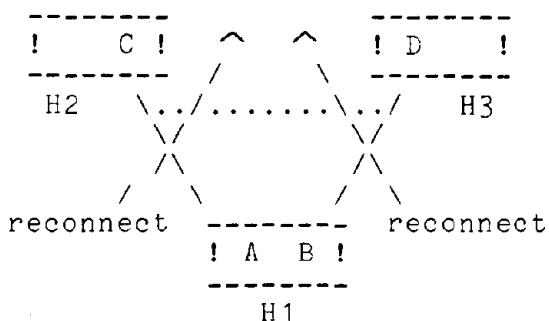


FIGURE 5

Either of the parties may use the RNO command to refuse or abort reconnection. H2 could respond to H1's RRQ with RNO; H1 can abort the reconnection by responding to ROK with RNO rather than RDO.

It is easy to insure that messages in transit are not lost during the reconnection. Receipt of the ROK message by H1 is taken to mean that no further messages are coming from H2; similarly receipt of RDO from H1 by H2 is taken to mean that no further messages are coming from H1.

To complete the specification of the reconnection mechanism consider the situation in which two "adjacent" entities initiate reconnections:

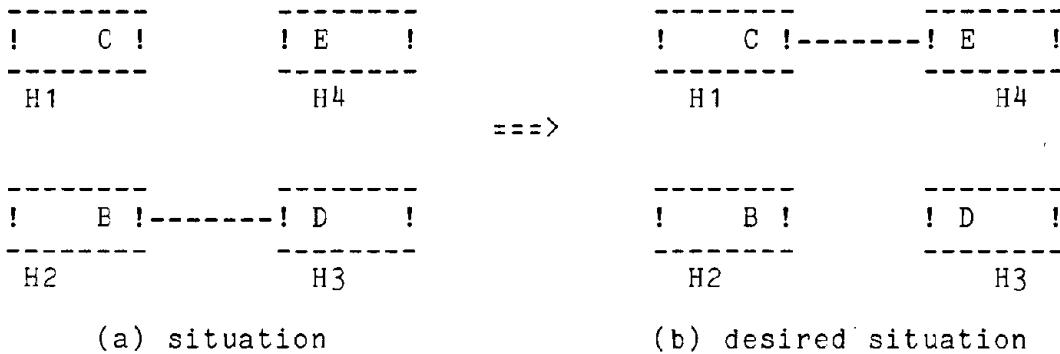


FIGURE 6

H2 and H3 "simultaneously" try to arrange for reconnection:

H2->H3: RRQ (path B-D)

H3->H2: RRQ (path D-B)

Thus, H2 sees an RRQ from H3 rather than an ROK or RNO in response to its RRQ to H3. This "race" situation can be resolved by having the reconnections proceed in series rather than in parallel: first one entity (say H2) performs its reconnect and then the other (H3) performs its reconnect. There are several means that could be used to decide which gets to go first. Perhaps the simplest is to base the decision on sockets and site addresses: the entity for which the 40 bit number formed by concatenating the 32 bit socket number with the 8 bit site address is largest gets to go first. Using this mechanism the rule is the following:

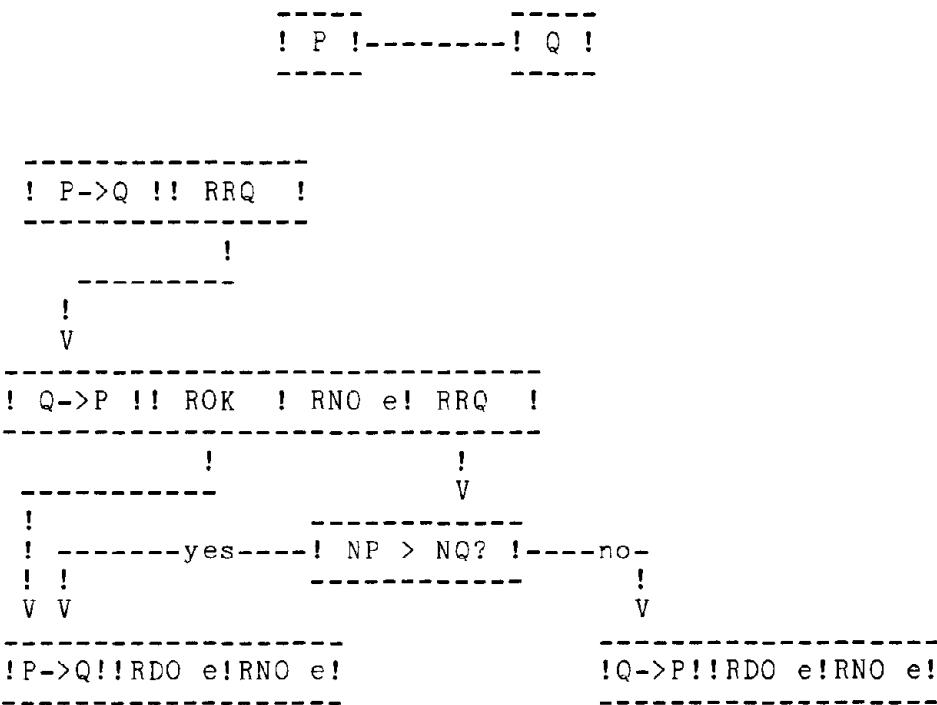
If H2 receives an RRQ from H3 in response to an RRQ of its own:

(let NH2, NH3 = the 40 bit numbers corresponding to H2 and H3)

- a. if $NH2 > NH3$ then both H2 and H3 interpret H3's RRQ as an ROK in response to H2's RRQ.
- b. if $NH2 < NH3$ then both interpret H3's RRQ as an RNO in response to H2's RRQ. This would be the only case in which it makes sense to "ignore" the refusal and try again - of course, waiting until completion of the first reconnect before doing so.

Once an ordering has been determined the reconnection proceeds as though there was no conflict.

The following diagram describes the legal protocol command/response exchange sequences for a reconnection initiated by P:



NP and NQ are the 40 bit numbers for P and Q; e indicates end of sequence.

6. A description of the option.

The reconnection mechanism described abstractly in the previous section can be effected as a TELNET option by use of the command RCP. Using this command and the TELNET DO, DON'T, WILL, WON'T, and SB prefixes, the four commands used in the previous abstract description become

```
RRQ => DO RCP  
ROK => WILL RCP  
RNO => WON'T RCP ;for responses to DO RCP  
DON'T RCP ;for responses to WILL RCP  
;i.e. used to cancel an RCP.
```

RDO <host> <socket> => SB RCP RCS <host> <socket>

A fifth command is also introduced

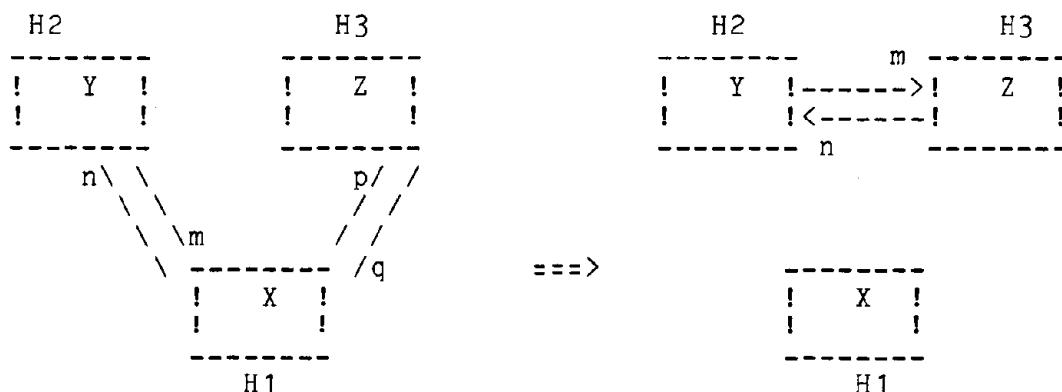
RWT <host> <socket> => SB RCP RCW <host> <socket>

The first three commands require no parameters since they refer to the connections they are received on. For RDO and RWT, <host> is an 8 bit (= 1 TELNET character) Host address and <socket> is a 32 bit (= 4 TELNET characters) number that specifies a TELNET receive socket at the specified Host (the associated transmit socket is always one higher than the receive socket).

A pending reconnection can be activated with either RDO or RWT. The response to either is to first break the TELNET connection with the sender and then reopen the TELNET connection to the Host and sockets specified. For RDO, the connection is to be reopened by sending two RFC's; for RWT, by waiting for two RFC's.

The RWT command is introduced to avoid requiring Hosts to queue RFC's.

As an example, the reconnection



could be accomplished as follows:

X->Y:	RRQ	(=IAC DO RCP)
X->Z:	RRQ	(=IAC DO RCP)
Y->X:	ROK	(=IAC WILL RCP)
Z->X:	ROK	(=IAC WILL RCP)
X->Y:	RWT H3 P	(=IAC SB RCP RCW H3 P)
X closes connections to Y		
Y closes connections to X		

Y waits for STR and RTS from H3
X->Z: RDO H2 N (=IAC SB RCP RCS H2 N)
X closes connections to Z
Z closes connections to X
Z sends STR and RTS to H2 which Y answers with matching RTS and
STR to compete reconnection

The RCS and RCW sub-commands should never be sent until a DO RCP has been acknowledged by a WILL RCP. Thus a Host not choosing to implement the reconnection option does not have to know what RCP means--all the Host need do in response to DO RCP is to transmit WON'T RCP. The WILL RCP and WON'T RCP commands should never be volunteered. If an unsolicited WILL RCP is ever received, a DON'T RCP should be fired back, which should be answered by a WON'T RCP command. If an unsolicited WON'T RCP command is received, it should be treated as a No-operation.

7. A word about security.

It should be clear that the decision to accept or reject a particular reconnection request is the responsibility of the entity (person at the terminal or process) using the connection. In many cases the entity may chose to delegate that responsibility to its TELNET (e.g., Example A, Section 4). However, the interface a Host provides to the reconnection mechanism would best include means for local entities to exercise control over response to remotely intitiated reconnection requests. For example, a user-TELNET might support several modes of operation with respect to remotely initiated reconnections:

1. transparent: all requested reconnections are to be performed in a way that is invisible to the user;
2. visible: all requested reconnections are to be performed and the user is to be informed whenever a reconnection occurs;
3. confirmation: the user is to be informed of each reconnection request which he may accept or reject;
4. rejection: all requested reconnects are to be rejected.

TELNET Suppress Go Ahead Option

1. Command name and code.

SUPPRESS-GO-AHEAD 3

2. Command meanings.

IAC WILL SUPPRESS-GO-AHEAD

The sender of this command requests permission to begin suppressing transmission of the TELNET GO AHEAD (GA) character when transmitting data characters, or the sender of this command confirms it will now begin suppressing transmission of GAs with transmitted data characters.

IAC WON'T SUPPRESS-GO-AHEAD

The sender of this command demands to begin transmitting, or to continue transmitting, the GA character when transmitting data characters.

IAC DO SUPPRESS-GO-AHEAD

The sender of this command requests that the sender of data start suppressing GA when transmitting data, or the sender of this command confirms that the sender of data is expected to suppress transmission of GAs.

IAC DON'T SUPPRESS-GO-AHEAD

The sender of this command demands that the receiver of the command start or continue transmitting GAs when transmitting data.

3. Default.

WON'T SUPPRESS-GG-AHEAD

DON'T SUPPRESS-GO-AHEAD

i.e., transmit GAs.

4. Motivation for the option.

While the NVT nominally follows a half duplex protocol complete with a GO AHEAD signal, there is no reason why a full duplex connection between a full duplex terminal and a Host optimized to handle full duplex terminals should be burdened with the GO AHEAD signal.

Therefore it is desirable to have a TELNET option with which parties involved can agree that one or the other or both should suppress transmission of GO AHEADS.

5. Description of the option.

When the SUPPRESS-GO-AHEAD option is in effect on the connection between a sender of data and the receiver of the data, the sender need not transmit GAs.

It seems probable that the parties to the TELNET connection will suppress GO AHEAD in both directions of the TELNET connection if GO AHEAD is suppressed at all; but, nonetheless, it must be suppressed in both directions independently.

With the SUPPRESS-GO-AHEAD option in effect, the IAC GA command should be treated as a NCP if received, although IAC GA should not normally be sent in this mode.

6. Implementation considerations.

As the SUPPRESS-GO-AHEAD option is sort of the opposite of a line at a time mode, the sender of data which is suppressing GO AHEADs should attempt to actually transmit characters as soon as possible (i.e., with minimal buffering) consistent with any other agreements which are in effect (e.g., approximate transmit message size agreements).

In many TELNET implementations it will be desirable to couple the SUPPRESS-GO-AHEAD option to the echo option so that when the echo option is in effect, the SUPPRESS-GG-AHEAD option is in effect simultaneously: both of these options will normally have to be in effect simultaneously to effect what is commonly understood to be character at a time echoing by the remote computer.

TELNET Message Size Negotiation Option
NIC 15393 (Aug. 1973)

TELNET Approximate Message Size Negotiation Option

1. Command name and code.

NAMS 4

(Negotiate Approximate Message Size)

2. Command meanings.

IAC WILL NAMS

The sender of this command requests, or agrees, to negotiate the approximate size for messages of data characters it sends.

IAC WON'T NAMS

The sender of this command refuses to negotiate the approximate size for messages of data characters it sends.

IAC DO NAMS

The sender of this command requests the receiver of this command to negotiate the approximate size for messages of data characters transmitted by the command receiver.

IAC DON'T NAMS

The sender of this command refuses to negotiate the approximate size for messages of data characters transmitted by the command receiver.

IAC SB NAMS DR <16 bit value>

The sender of this command requests the receiver of this command to set its approximate message size for data the command receiver transmits to the value specified in the 16 bit parameter, a data character count. The code for DR (Data Receiver) is 0.

IAC SB NAMS DS <16 bit value>

The sender of this command requests or agrees to set its approximate message size for data it transmits to the value specified in the 16 bit parameter, a data character count. The code for DS (Data Sender) is 1.

TELNET Message Size Negotiation Option
NIC 15393 (Aug. 1973)

3. Default

WON'T NAMS

DON'T NAMS

i.e., no attempt will be made to agree on a message size.

4. Motivation for the option.

The TELNET protocol does not specify how many characters the transmitter of data should attempt to pack into messages it sends. However, 1) some receivers may prefer received messages to generally have some minimum size, for example, to lessen the burden of processing input interrupts; 2) some receivers may prefer received data messages to generally have some maximum size, for example, because the maximum data message size could be used in conjunction with the Host/Host protocol message and bit allocates to more efficiently utilize input buffer space; 3) some transmitters may have maximum sizes for transmitted data messages, information which could be used in conjunction with the Host/Host protocol message and bit allocates to more efficiently utilize the receiver's input buffer space; and 4) some transmitters may desire to transmit some minimum size message, for example, to lessen the burden of processing output interrupts.

Therefore, it is desirable to have some mechanism whereby the parties involved can attempt to agree on the approximate size of messages transmitted over the connection. (It might be even more powerful to be able to negotiate approximate or even exact upper and lower bounds on message size. However, fixed bounds would sometimes be hard to manage and sometimes even in conflict with Host/Host protocol allocates; and specifying both upper and lower bounds, even approximately, seems overly complicated considering the expected payoff.)

5. Description of the option.

With the option which specifies the approximate size of messages transmitted over the connection, the transmitter attempts to send messages of the specified size unless some other constraint (for instance, an end of line) requires the message to be sent sooner, or characters for transmission arrive so fast that the message has to be bigger than the specified size. The option is to be used strictly to improve the STATISTICS (e.g., timing and buffering) of message reception and transmission -- the option does NOT specify any absolutes.

With this option not in effect, message size is completely (even statistically) undefined as per the NVT specification.

TELNET Message Size Negotiation Option
NIC 15393 (Aug. 1973)

Once the data transmitter and receiver have agreed to negotiate the approximate message size, they must actually do this negotiation. This is done using the DS and DR SB commands. The transmitter of data messages may give the SB NAMS DS command and the receiver may give the SB NAMS DR command. The rules for negotiation of the actual approximate message size are as follows:

- a) Either party may at any time send a SB command specifying a value less than any previously sent or received and immediately assume that that value has been agreed upon.
- b) If either party receives a SB command, the party should assume the value specified in the received command is in effect if the party has not previously sent a SB command specifying a lower value.
- c) Before any SB command is sent, the approximate message size is undefined.
- d) At any time either party may quit the whole thing by sending a DON'T or WON'T NAMS command which must be acknowledged and the approximate message length becomes undefined.
- e) An approximate message size value may not be less than one.

As the receiver and transmitter may have conflicting requirements for the approximate message size, neither should be cavalier about requesting a specified approximate message size, each "bending over backward" to let the other party (who should be presumed to have a greater need) specify the approximate message size.

Host/Host protocol allocate considerations, of course, always dominate negotiated message size considerations.

Request for Comments: 651
D. Crocker (UCLA-NMC)
original file: <[ISI]<DCROCKER>STATUS-OPTION-REVISION.RNO

Revised TELNET Status Option

1. Command name and code

STATUS 5

2. Command meanings

As described in the NAOL and NAOP option specifications, this option applies to a simplex connection.

IAC DO STATUS

Sender of DO wishes to be able to send requests for status-of-options information, or confirms that he is willing to send such requests.

IAC WILL STATUS

Sender of WILL wishes or agrees to send status information, spontaneously or in response to future requests.

IAC DON'T STATUS

Sender refuses to carry on any further discussion of the current status of options.

IAC WON'T STATUS

Sender refuses to carry on any further discussion of the current status of options.

IAC SB STATUS SEND IAC SE

Sender requests receiver to transmit his (the receiver's) perception of the current status of TELNET options. The code for SEND is 1. (See below.)

IAC SB STATUS IS ... IAC SE

Sender is stating his perception of the current status of TELNET options. The code for IS is 0. (See below.)

Preceding page blank

3. Default

DON'T STATUS/WON'T STATUS. That is, the current status of options will not be discussed.

4. Motivation for the option

This option allows a user/process to verify the current status of TELNET options (e.g., echoing) as viewed by the person/process on the other end of the TELNET connection. Simply renegotiating options could lead to the nonterminating request loop problem discussed in (NIC #16237). The changes to the option, described in this paper, allow STATUS to fit into the normal structure of TELNET options, by deferring the actual transfer of status information to the SB command. Additionally, the numbers of bytes that must be sent to describe the state of the options has been considerably reduced.

5. Description of the option

WILL/DO are now used only to obtain and grant permission for future discussion. The actual exchange of status information occurs within option subcommands (IAC SB STATUS...).

Once the two hosts have exchanged a WILL and a DO, the sender of the WILL STATUS is free to transmit status information, spontaneously or in response to a request from the sender of the DO. At worst, this may lead to transmitting the information twice. Only the sender of the DO may send requests (IAC SB STATUS SEND IAC SE) and only the sender of the WILL may transmit actual status information (within an IAC SB STATUS IS ... IAC SE command).

IS has the subcommands WILL, DO and SB. They are used EXACTLY as used during the actual negotiation of TELNET options, except that SB is terminated with SE, rather than IAC SE. Transmission of SE, as a regular data byte, is accomplished by doubling the byte (SE SE). Options that are not explicitly described are assumed to be in their default states. A single IAC SB STATUS IS ... IAC SE describes the condition of ALL options.

The following is an example of use of the option:

Host1: IAC DO STATUS

Host2: IAC WILL STATUS

(Host2 is now free to send status information at any time. Solicitations from Host1 are NOT necessary. This should not produce any dangerous race conditions. At worst, two IS's will be sent.)

Revised TELNET Status Option
RFC 651, NIC 31154 (Oct. 25, 1974)

Host1 (perhaps): IAC SB STATUS SEND IAC SE

Host2 (the following stream is broken into multiple lines only for readability. No carriage returns are implied.):

IAC SB STATUS IS
WILL ECHO
DO SUPPRESS-GO-AHEAD
WILL STATUS
DO STATUS
WILL RCTE
SB RCTE <11><1><24> SE
DO NAOL
SB NAOL DS <66> SE
IAC SE

Explanation of Host2's perceptions: It is responsible for echoing back the data characters it receives over the TELNET connection; it will not send Go-Ahead signals; it will both issue and request Status information; it will send instruction for controlling the other side's terminal printer; it will discuss the line width for data it is sending.

TELNET Timing Mark Option

1. Command name and code.

TIMING-MARK b

2. Command meanings.

IAC DO TIMING-MARK

The sender of this command REQUESTS that the receiver of this command return a WILL TIMING-MARK in the data stream at the "appropriate place" as defined in section 4 below.

IAC WILL TIMING-MARK

The sender of this command ASSURES the receiver of this command that it is inserted in the data stream at the "appropriate place" to insure synchronization with a DO TIMING-MARK transmitted by the receiver of this command.

IAC WON'T TIMING-MARK

The sender of this command REFUSES to insure that this command is inserted in the data stream at the "appropriate place" to insure synchronization.

IAC DON'T TIMING-MARK

The sender of this command notifies the receiver of this command that a WILL TIMING-MARK (previously transmitted by the receiver of this command) has been IGNORED.

3. Default

WON'T TIMING-MARK, DON'T TIMING-MARK

i.e., no explicit attempt is made to synchronize the activities at the two ends of the TELNET connection.

4. Motivation for the option

It is sometimes useful for a user or process at one end of a TELNET connection to be sure that previously transmitted data has been completely processed, printed, discarded, or otherwise disposed of. This option provides a mechanism for doing so. In addition, even if the option request (DO TIMING-MARK) is refused (by WON'T TIMING-MARK) the requester is at least assured that the refuser has received (if not processed) all previous data.

Preceding page blank

TELNET Timing Mark Option
NIC 16238

As an example of a particular application, imagine a TELNET connection between a physically full duplex terminal and a "full duplex" server system which permits the user to "type ahead" while the server is processing previous user input. Suppose that both sides have agreed to Suppress GA and that the server has agreed to provide echos. The server now discovers a command which it cannot parse, perhaps because of a user typing error. It would like to throw away all of the user's "type-ahead" (since failure of the parsing of one command is likely to lead to incorrect results if subsequent commands are executed), send the user an error message, and resume interpretation of commands which the user typed after seeing the error message. If the user were local, the system would be able to discard buffered input; but input may be buffered in the user's Host or elsewhere. Therefore, the server might send a DO TIMING-MARK and hope to receive a WILL TIMING-MARK from the user at the "appropriate place" in the data stream.

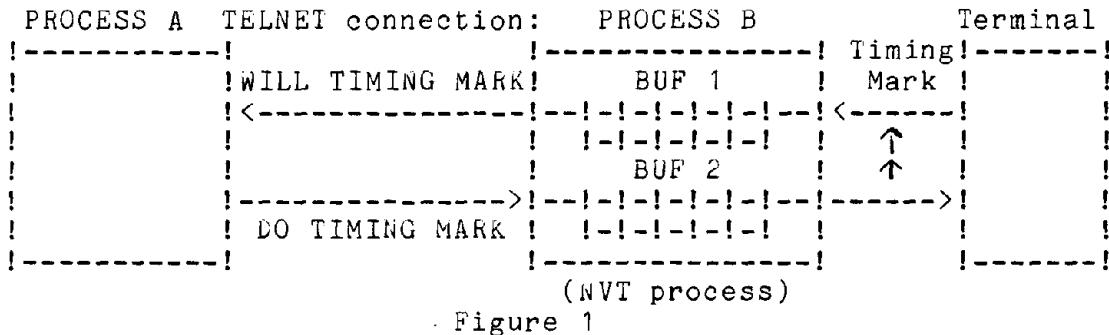
The "appropriate place", therefore (in absence of other information) is clearly just before the first character which the user typed after seeing the error message. That is, it should appear that the timing mark was "printed" on the user's terminal and that, in response, the user typed an answering timing mark.

Next, suppose that the user in the example above realized that he had misspelled a command, realized that the server would send a DO TIMING-MARK, and wanted to start "typing ahead" again without waiting for this to occur. He might then instruct his own system to send a WILL TIMING-MARK to the server and then begin "typing ahead" again. (Implementers should remember that the user's own system must remember that it sent the WILL TIMING-MARK so as to discard the DO/DON'T TIMING-MARK when it eventually arrives.) Thus, in this case the "appropriate place" for the insertion of the WILL TIMING-MARK is the place defined by the user.

It should be noted, in both of the examples above, that it is the responsibility of the system which transmits the DO TIMING-MARK to discard any unwanted characters; the WILL TIMING-MARK only provides help in deciding which characters are "unwanted".

5. Description of the option

Suppose that Process A of Figure 1 wishes to



synchronize with B. The DO TIMING-MARK is sent from A to B. B can refuse by replying WON'T TIMING-MARK, or agree by permitting the timing mark to flow through his "outgoing" buffer, BUF2. Then, instead of delivering it to the terminal, B will enter the mark into his "incoming" buffer BUF1, to flow through toward A. When the mark has propagated through B's incoming buffer, B returns the WILL TIMING-MARK over the TELNET connection to A.

When A receives the WILL TIMING-MARK, he knows that all the information he sent to B before sending the timing mark has been delivered, and all the information sent from B to A before turnaround of the timing mark has been delivered.

Three typical applications are:

- A. Measure round-trip delay between a process and a terminal or another process.
- B. Resynchronizing an interaction as described in section 4 above. A is a process interpreting commands forwarded from a terminal by B. When A sees an illegal command it:
 - i. Sends <carriage return>, <line feed>, <question mark>
 - ii. Sends DO TIMING-MARK
 - iii. Sends an error message
 - iv. Starts reading input and throwing it away until it receives a WILL TIMING-MARK
 - v. Resumes interpretation of input.

TELNET Timing Mark Option
NIC 16238

This achieves the effect of flushing all "type ahead" after the erroneous command, up to the point when the user actually saw the question mark.

C. The dual of B above. The terminal user wants to throw away unwanted output from A.

- i. B sends DO TIMING-MARK, followed by some new command.
- ii. B starts reading output from A and throwing it away until it receives WILL TIMING-MARK.
- iii. B resumes forwarding A's output to the terminal.

This achieves the effect of flushing all output from A, up to the point where A saw the timing mark, but not output generated in response to the following command.

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

David C. Crocker (UCLA-NMC)
Jonathan B. Postel (UCLA-NMC)
Obsoletes NIC 18492

Remote Controlled Transmission and Echoing TELNET Option

1. Command name and code:

RCTE 7

2. Command meanings:

IAC WILL RCTE

The sender of this command REQUESTS or AGREES to use the RCTE option, and will send instructions for controlling the other side's terminal printer.

IAC WON'T RCTE

The sender of this option REFUSES to send instructions for controlling the other side's terminal printer.

IAC DO RCTE

The sender REQUEST or AGREES to have the other side (sender of WILL RCTE) issue commands which will control his (sender of the DO) output to the terminal printer.

IAC DON'T RCTE

The sender of this command REFUSES to allow the other side to control his (sender of DON'T) terminal printer.

IAC SB RCTE <cmd> [BC1 BC2] [TC1 TC2] IAC SE

where:

<cmd> is one 8-bit byte having the following flags (bits are counted from the right):

Bit	Meaning
7	Unused
6	Unused
5	Unused
4	Unused
3	Unused
2	Unused
1	Unused
0	Unused

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

- 0 0 = Ignore all other bits in this byte and repeat the last <cmd> that was sent. Equals a 'continue what you have been doing'.
- 1 = Perform actions as indicated by other bits in this byte.
- 1 0 = Print (echo) Break character
- 1 = Skip (don't echo) Break character
- 2 0 = Print (echo) text up to Break character
- 1 = Skip (don't echo) text up to Break character
- 3 0 = Continue using same classes of Break characters.
- 1 = The two 8-bit bytes following this byte contain flags for the new Break classes.
- 4 0 = Continue using same classes of Transmit characters.
- 1 = Reset Transmit classes according to the two bytes following 1) the Break classes bytes, if the Break classes are also being reset, or 2) this byte, if the Break classes are NOT also being reset.

Value (decimal) of the <cmd> byte and its meaning:

- 0 = Continue what you have been doing
- 1 = Print (echo) up to AND INCLUDING Break character
- 3 = Print up to Break character and SKIP (don't echo) Break character
- 5 = Skip text (don't echo) up to Break character, but PRINT Break character
- 7 = Skip up to and including Break character

Add one of the previous non-zero values to one of the following values, to get the total decimal value for the byte (Note that Classes may not be reset without also resetting the printing action; so an odd number is guaranteed):

- 8 = Set Break classes (using the next two bytes [BC1 BC2])
- 16 = Set Transmission classes (using the next two bytes [TC1 TC2])
- 24 = Set Break classes (using the next two bytes [BC1

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

BC2]) and the Transmission classes (using the two bytes after that [TC1 TC2]).

Sub-commands (IAC SB RCTE...) are only sent by the Controlling Host and, in addition to other functions, functionally replace the Go-Ahead (IAC GA) Telnet option. RCTE also functionally replaces the Echo (IAC ECHO) Telnet option.

3. Default:

WON'T RCTE -- DON'T RCTE

Neither host asserts special control over the other host's terminal printer.

4. Motivation for the option:

RFC's 1, 5 and 51 discuss Network and process efficiency and smoothness.

RFC 357, by John Davidson, introduces the problem of echoing delay that occurs when a remote user accesses a full-duplex host, thru a satellite link. In order to save the many thousands of miles of transit time for each echoed character, while still permitting full server responsiveness and clean terminal output, an echo control similar to that used by some time-sharing systems is suggested for the entire Network.

In effect, the option described in this document involves making a using host carefully regulate the local terminal printer according to explicit instructions from the foreign (serving) host.

An important additional issue is efficient Network transmission. Implementation of the Davidson Echoing Scheme will eliminate almost all server-to-user echoing.

The option described in this document also requests using hosts to buffer a terminal's input to the serving host until it forms a useful unit (with "useful unit" delimited by Break or Transmission characters as described below). Therefore, fewer messages are sent on the user-to-server path.

N.B.: This option is only intended for use with full-duplex hosts. The Go-Ahead Telnet feature is completely adequate for half-duplex server hosts. Also, RCTE should be used in place of the ECHO TELNET option.

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

5. Explicit description of control mechanism:

A. Overview of User Terminal Printing Action & Control

- (1) Agree to use RCTE option
- (2) User holds echo printing until instructed by server to do otherwise
- (3) Server may send output to terminal printer.
- (4) Network output is printed up to an RCTE command
- (5) Server sends IAC SB RCTE <cmd> IAC SE
- (6) User acts upon the <cmd> command up to a Break character or until receipt of output from the server host.
- (7) Go to (2)

Note: Output from the server host may occur at any time, in which case, the flow of control switches to (2) and then proceeds to (3), (4), etc.

B. Explanation:

- (1) Both hosts agree to use the RCTE option. After that, the using host (IAC DO RCTE) merely acts upon the Controlling (serving) host's commands and does not issue any RCTE commands unless and until it (using host) decides to stop allowing use of the option (by sending IAC DON'T RCTE).
- (2) Using host begins synchronization between the serving host and itself by suspending terminal echo printing until directed to do otherwise by the controlling host, thru an IAC SB RCTE <cmd> IAC SE.
- (3) The server may send output to the terminal printer, either in response to input from the user (in which case it is already synchronized with the terminal input) or spontaneously. In the latter case, flow of control automatically switches to (2) and continues from there. Output from the server is defined as completed when step (5) occurs. That is, text from the Server to the terminal printer MUST end with an RCTE SB command.
- (4) Any output from the server is printed on the terminal IMMEDIATELY. Again note that the end of such output is defined to be the occurrence of an IAC SB RCTE <cmd> IAC SE command.
- (5) Server sends an RCTE command. The command may redefine Break and Transmission classes, Action to be performed on Break characters, and action to be performed on text. Each of these

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

independent functions is controlled by separate bits in the <cmd> byte.

- a. A Transmission character is one which RECOMMENDS that the Using Host transmit all text accumulated up to and including its occurrence. (For Net efficiency, Using hosts are DISCOURAGED (but not prohibited) from sending before the occurrence of a Transmission character, as defined at the moment the character is typed).

If the Transmission Classes bit (Bit 4) is on, the two bytes following the two Break Classes bytes (or immediately following the <cmd> byte, if the Break Classes bit is not on) will indicate what classes are to be enabled.

If the Bit is OFF, the Transmission classes remain unchanged. When the RCTE option is first initiated, NO CLASSES are in effect. That is, no character will be considered a Transmission character. (As if both TC1 and TC2 are zero.)

- b. A Break character REQUIRES that the Using host transmit all text accumulated up to and including its occurrence and also causes the Using host to stop its print/discard action upon the User's input text, until directed to do otherwise by another IAC SB RCTE <cmd> IAC SE command from the Serving host. Break characters therefore define printing units. "Break character" as used in this document does NOT mean Telnet Break character.

If the Break Classes bit (Bit 3) is on, the two bytes following <cmd> will indicate what classes are to be enabled. There are currently nine (9) classes defined, with room for expansion.

If the bit is OFF, the Break classes remain unchanged. When the RCTE option is initiated, CLASSES 4, 5, and 9 are to be in effect. That is, Format Effectors, Non-format effector Control Characters and DEL, and Punctuation characters are to be Break characters.

- c. The list of character classes, used to define Break and Transmission classes are listed at the end of this document, in the "Tables" Section.
- d. Because Break characters are special, the print/discard action that should be performed upon them is not always the same as should be performed upon the rest of the input text.

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

For example, while typing a filename to TENEX, I want the text of the filename to be printed (echoed); but I do not want the <escape> (if I use the name completion feature) to be printed.

If Bit 1 is ON The Break character is NOT to be printed.

- e. A separate bit (Bit 2) signals whether or not the text itself should be printed (echoed) to the terminal. If Bit 2 = 0, then the text IS to be printed.
- f. Yet another bit (Bit 0 - right-most bit) signals whether or not any of the other bits of the command should be checked. If this bit is OFF, then the command should be interpreted to mean "continue whatever echoing strategy you have been following, using the same Break and Transmission classes."

This is particularly useful for the <cmd> command that follows spontaneously generated output from the Serving host (such as "System Going Down") which needs to signal End-of-Message, but does not usually want to reset any other conditions.

- (6) Input from the terminal is (hopefully) buffered into units ending with a Transmission or Break character; and echoing of input text is suspended after the occurrence of a Break Character and until receipt of a Break Reset command from the Serving host. The most recent RCTE Break reset command determines the Break actions.
- (7) When a Break character is typed, the cycle of control is complete and action re-commences at (2). Action also automatically switches to (2) upon receipt of any text from the Server host.

C. Notes, Comments, Etc.:

- (1) Even-Numbered Commands, greater than zero, are in error, since they will have the low-order bit off. The command should be interpreted as equal to zero, which means that any Classes Reset bytes ([TC1 TC2] [BC1 BC2]) will be in error. (The IAC SE, at the end of the command, eliminates any parsing problems due to this error.)
- (2) Serving hosts will generally instruct Using hosts NOT to echo Break Characters, even though it might be alright to echo most Break characters. For example, <cr> is usually a safe character to echo but <esc> is not. TENEX Exec is willing to accept either, during filename specification. Therefore, the

using host must be instructed NOT to echo ANY Break Characters.

This is generally a tolerable problem, since the serving host has to send an RCTE command at this point, anyhow. Adding the Break character to the message (so that it appears to be echoed) will not cause any extra Network traffic.

- (3) The RCTE Option entails a rather large overhead. In a true character-at-a-time situation, this overhead is not justified. But on the average, it should result in significant savings, both in Network traffic and Host wake-ups.
- (4) Buffering Problems and Transmission vs. Printing Constraints:

There are NO mandatory transmission constraints. The Using host is allowed to send a character a time, though this would be a waste of RCTE. The Transmission Classes commands are GUIDELINES, so deviating from them, as when the User's buffer gets full, is allowed.

Additionally, the Using host may send a Break Class character, without knowing that it is one (as with type-ahead).

The problem with buffering occurs when printing on the user's terminal must be suspended, after the user has typed a currently valid Break Character and until a Break Reset command is received from the serving host. During this time, the user may be typing merrily along. The text being typed may be SENT, but may not yet be PRINTED.

The more standard problem of filling the transmission buffer, while awaiting an ALLOC from the Serving host, may also occur, but this problem is well known to implementors and in no way special to RCTE.

In any case, when the buffer does fill and further text typed by the user will be lost, the user should be notified.

- (5) The Serving and Using hosts must carefully synchronize Break Class Reset commands with the transmission of Break characters. Except at the beginning of an interaction, the Serving host MAY ONLY send a Break Reset command in response to the Using host's having sent a Break character as defined at that time. This should establish a one-to-one correspondence between them. (A <cmd> value of zero, in this context, is interpreted as a Break Classes reset to the same

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

class(es) as before.) The Reset command may be preceded by terminal output.

- (6) Text should be buffered by the Using host until the user types a character which belongs to the transmission class in force at THE MOMENT THE CHARACTER IS TYPED.
- (7) Transmission Class Reset commands may be sent by the Serving host at ANY TIME. If they are frequently sent separate from Break Class Reset commands, it will probably be better to exit from RCTE and enter regular character at a time transmission.
- (8) It is not immediately clear what the Using host should do with currently buffered text, when a Transmission Classes Reset command is received. The buffering is according to the previous Transmission Classes scheme.

The Using host clearly should NOT simply wait until a Transmission character (according to the new scheme) is typed.

Either the buffered text should be rescanned, under the new scheme;

Or the buffered text should simply be sent as a group. This is the simpler approach, and probably quite adequate.

- (9) It is possible to define NO BREAK CHARACTERS except TELNET commands (IAC ...). This might actually be useful, as in the case of transmitting on carriage-return, with the Using host echoing (a controlled half-duplex).

Having the serving host send a Telnet command will allow the server to know when he may reset the Break classes, but the mechanism is awkward and probably should be avoided.

D. Sample Interaction:

"S:" is sent from Serving (WILL RCTE) host to Using host.

"U:" is sent from Using (DO RCTE) host to Serving host.

"T:" is entered by the terminal user.

"P:" is printed on the terminal.

Text surrounded by square brackets ([]) is commentary.

Text surrounded by angle brackets (<>) is to be taken as a single unit. E.g., carriage return is <cr>, and the decimal value 27 is represented <27>.

The following interaction shows a Logon to a Tenex, initiation of the DED editor, insertion of some text and the return to the Exec level.

An attempt has been made to give some flavor of the asynchrony of Network I/O and the user's terminal input. Many other possible combinations, using the same set of actions listed below, could be devised. The actual order of events will depend upon network and hosts' load and the user's typing speed.

A Telnet connection has already been opened, but the TENEX prompt has not yet been issued. The hosts first discuss using the RCTE option:

S: <IAC><WILL><RCTE>

U: <IAC><DO><RCTE>

S: TENEX 1.31.18, TENEX EXEC 1.50.2 <cr><lf>@
<IAC><SB><RCTE><11><1><24><IAC><SE>

[Print the Herald and echo input text upto a Break Character, but do not echo the Break Character. Classes 4 (Format Effectors), 5 (Non-format effector Controls and), and 9 (<space>) act as Break Characters.]

P: TENEX 1.31.18, TENEX EXEC 1.50.2 <cr><lf>@

T: LOGIN ARPA <cr>

P: LOGIN

U: LOGIN <space>

U: ARPA <cr>

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

S: <space><IAC><SB><RCTE><0><IAC><SE>
P: <space>ARPA
S: <cr><lf> (PASSWORD): <IAC><SB><RCTE><7><IAC><SE>
P: <cr><lf> (PASSWORD):
T: WASHINGTON 1000<cr>
[The password "WASHINGTON" is not echoed. Printing of
"1000<cr>" is withheld]
U: WASHINGTON <space>
U: 1000<cr>
S: <space><IAC><SB><RCTE><3><IAC><SE>
S: <cr><lf> JOB 17 ON TTY41 7-JUN-73 14:13 <cr><lf>@
<IAC><SB><RCTE><0><IAC><SE>
P: <space> 1000
[Printing is slow at this point; so the account number
is not printed as soon as the server's command for it is
received.]
P: <cr><lf> JOB 17 ON TTY41 7-JUN-73 14:13 <cr><lf>@
T: DED <esc><cr>
P: DED
U: DED<esc>
S: .SAV;1 <IAC><SB><RCTE><0><IAC><SE>
P: .SAV;1
U: <cr>
S: <cr><lf><lf> Ded 3/14/73 DRO,KRK <cr><lf>:
<IAC><SB><RCTE><15><1><255><IAC><SE>
[The program is started and the DED prompt ":" is sent.
At the command level, DED responds to every character.]
P: <cr><lf><lf> DED 3/14/73 DRO,KRK <cr><lf>:

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

T: IThis is a test line.<cr> This is another test line.<^Z>
Q

["I" means Insert Text. The text follows, terminated by a Control-Z. The "Q" instructs DED to Quit.]

U: I

U: This is a test line.<cr>

S: I<cr><lf>* <IAC><SB><RCTE><11><0><24><IAC><SE>

[DED prompts the user, during text input, with an asterisk at the beginning of every line.]

P: I<cr><lf> *This is a test line.

S: <cr><lf>* <IAC><SB><RCTE><0><IAC><SE>

P: <cr><lf>* This is another test line.

U: This is another test line.<^Z>

U: Q

[Note that the "Q" will not immediately be printed on the terminal, since it is a Break character.]

S: ^Z<cr><lf>: <IAC><SB><RCTE><15><1><255><IAC><SE>

[The returned "^Z" is two characters, not the ASCII Control-Z.]

S: Q<cr><lf>@ <IAC><SB><RCTE><11><1><24><IAC><SE>

P: Q<cr><lf> @

And the user is returned to the Exec level.

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

E. Tables:

(1) <cmd> is one 8-bit byte having the following flags (bits are counted from the right):

Bit Meaning

- | | |
|---|---|
| 0 | 0 = Ignore all other bits in this byte and repeat the last <cmd> that was sent. Equals a 'continue what you have been doing'. |
| 1 | = Perform actions as indicated by other bits in this byte. |
| 1 | 0 = Print (echo) Break character
1 = Skip (don't echo) Break character |
| 2 | 0 = Print (echo) text up to Break character
1 = Skip (don't echo) text up to Break character |
| 3 | 0 = Continue using same classes of Break characters.
1 = The two 8-bit bytes following this byte contain flags for the new Break classes. |
| 4 | 0 = Continue using same classes of Transmit characters
1 = Reset Transmit classes according two the two bytes following 1) the Break classes bytes, if the Break classes are also being reset, or 2) this byte, if the Break classes are NOT also being reset. |

Byte value (decimal) and its meaning:

0 = Continue what you have been doing

Even numbers greater than zero (i.e., numbers with the right-most bit off) are in error and should be interpreted as equal to zero. When the <cmd> is an even number greater than zero, Classes bytes TC1 & TC2 and/or BC1 & BC2 MUST NOT BE SENT.

1 = Print (echo) up to AND INCLUDING Break character

3 = Print up to Break character and SKIP (don't echo) Break character

5 = Skip text (don't echo) up to Break character, but PRINT Break character

7 = Skip up to and including Break character

Remote Controlled Transmission & Echoing TELNET Option
NIC 19859 (Nov. 1, 1973)

Add one of the previous non-zero values to one of the following values, to get the total decimal value for the byte (Note that Classes may not be reset, without also resetting the printing action; so an odd number is guaranteed):

8 = Set Break classes (using the next two bytes [BC1 BC2])
16 = Set Transmission classes (using the next two bytes [TC1 TC2])
24 = Set Break classes (using the next two bytes [BC1 BC2]) and the Transmission classes (using the two bytes after that [TC1 TC2]).

(2) Classes for Break and Transmission (The right-most bit of the second byte (TC2 or BC2) represents Class 1; the left-most bit of the first byte (TC1 or BC1) represents the currently undefined Class 16):

- 1: Upper-Case Letter (A-Z)
- 2: Lower-case letters (a-z)
- 3: Numbers (0-9)
- 4: Format Effectors (<BS> <CR> <LF> <FF> <HT> <VT>)
- 5: Non-format effector Control Characters, and <ESC>
- 6: . , ; : ? !
- 7: { [(< >)] }
- 8: ' " / \ % @ \$ & # + - * = ^ _ | ~
- 9: <Space>

And Telnet commands (IAC . . .) are ALWAYS to have the effect of a Break character.

Jon Postel (UCLA-NMC)
Dave Crocker (UCLA-NMC)

TELNET Output Line Width Option

Command name and code.

NAOL 8 (Negotiate About Output Line-width)

Command meanings

In the following, we are discussing a simplex connection, one half of a full duplex TELNET connection. On the simplex connection under discussion, by definition data passes from the data sender to the data receiver. If we consider the example of a computer transmitting data over a connection to a terminal where the data is printed, then the computer is the data sender and the terminal is the data receiver. Continuing to use this example, the NAOL option could be used to negotiate the line width to be used when printing lines from the computer on the terminal. To negotiate line width on the other half of the TELNET connection the parties involved reverse their data sender and data receiver roles; this can be done unambiguously as the sender of a DO or DON'T NAOL command can only be the data sender, thus defining the half of the TELNET connection under discussion, and the sender of a WILL or WON'T NAOL command can only be the data receiver.

IAC DO NAOL

The data sender requests or agrees to negotiate about output line width with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output line width considerations.

IAC DON'T NAOL

The data sender refused to negotiate about output line width with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOL

The data receiver requests or agrees to negotiate about output line width with the data sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output line width considerations.

TELNET Output Line Width Option
NIC 20196 (Nov. 13, 1973)

IAC WON'T NAOL

The data receiver refuses to negotiate about output line width, or demands a return to the unnegotiated default mode.

IAC SB NAOL DS <8 bit value> IAC SE

The data sender specifies, with the 8 bit value, which party should handle output line width considerations and how. The code for DS is 1.

IAC SB NAOL DR <8 bit value> IAC SE

The data receiver specifies, with the 8 bit value, which party should handle output line width considerations and how. The code for DR is 0.

Default

DON'T NAOL
WON'T NAOL

In the default absence of negotiation concerning which party, data sender or data receiver, is handling output line width considerations, neither party is required to handle line width consideration and neither party is prohibited from handling line width consideration but it is appropriate if at least the data receiver handles line width considerations albeit primitively.

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about the output line width:

- (a) the sender may wish the receiver to use its local knowledge of the printer width to properly handle the line width;
- (b) the receiver may wish the sender to use its local knowledge of the data being sent to properly handle the line width;
- (c) the sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of the line width; and
- (d) the receiver may wish to use its local knowledge of the printer width to instruct the sender in the proper handling of the line width.

An example of proper handling of the line length is for the receiver to "fold" lines sent by the sender so that the lines fit on the printer page. Another example of proper handling of the line length might be not folding lines even though they overflow the printer page, as that is what the user desires.

Description of the Option

The data sender and data receiver use the 8 bit value along with the LS and DR SB commands as follows:

8 bit value	Meaning
0	command sender suggests he alone will handle output line-width considerations for the connection.
1 to 253	command sender suggests other party alone should handle output line-width considerations but suggests line width should be value given, in characters.
254	command sender suggests other party alone should handle output line-width considerations but suggests line width should be considered infinity.
255	command sender suggests other party alone should handle output line-width considerations and suggests nothing about how it should be done.

The guiding rules are that

- (1) if neither data receiver or data sender wants to handle output line-width considerations, the data receiver must do it, and
- (2) if both data receiver or data sender want to handle output line-width considerations, the data sender gets to do it.

The reasoning for the former rule is that if neither want to do it, then the default in the NAOL option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver makes.

TELNET Output Line Width Option
NIC 20196 (Nov. 13, 1973)

Some sample negotiations are:

no subnegotiations

data receiver handles output
line-width considerations

IAC SB NAOL DS 132 IAC SE
IAC SB NAOL DR 0 IAC SE

data sender suggests data
receiver handle output line-width
consideration data with suggested
line width of 132; receiver
agrees.

IAC SB NAOL DR 255 IAC SE
IAC SB NAOL DS 0 IAC SE

data receiver suggests data
sender handle line-width
considerations; sender refuses.

IAC SB NAOL DS 0 IAC SE
IAC SB NAOL DR 72 IAC SE

data sender wants to handle
line-width considerations;
receiver agrees but notifies the
sender the line printer only has
72 columns.

As with all option negotiation, neither party should suggest a state
already in effect except to refuse to negotiate; changes should be
acknowledged; and once refused, an option should not be resuggested
until "something changes" (e.g., another process starts).

At any time either party can disable further negotiation by giving
the appropriate WON'T NAOL or DON'T NAOL command.

TELNET Output Page Size Option
NIC 20197 (Nov. 13, 1973)

Jon Postel (UCLA-NIC)
Dave Crocker (UCLA-NMC)

TELNET Output Page Size Option

Command name and code.

NAOP 9 (Negotiate About Output Page-size)

(By page size we mean number of lines per page.)

Command meanings.

In the following, we are discussing a simplex connection, one half of a full duplex TELNET connection. On the simplex connection under discussion, by definition data passes from the data sender to the data receiver. If we consider the example of a computer transmitting data over a connection to a terminal where the data is printed, then the computer is the data sender and the terminal is the data receiver. Continuing to use this example, the NAOP option could be used to negotiate the page size to be used when printing pages from the computer on the terminal. To negotiate page size on the other half of the TELNET connection the parties involved reverse their data sender and data receiver roles; this can be done unambiguously as the sender of a DO or DON'T NAOP command can only be the data sender, thus defining the half of the TELNET connection under discussion, and the sender of a WILL or WON'T NAOP command can only be the data receiver.

IAC DO NAOP

The data sender requests or agrees to negotiate about output page size with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output page size considerations.

IAC DON'T NAOP

The data sender refuses to negotiate about output page size with the data receiver, or demands a return to the unnegotiated default mode.

TELNET Output Page Size Option
NIC 20197 (Nov. 13, 1973)

IAC WILL NAOP

The data receiver requests or agrees to negotiate about output page size with the data sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output page size considerations.

IAC WON'T NAOP

The data receiver refuses to negotiate about output page size, or demands a return to the unnegotiated default mode.

IAC SB NAOP DS <8 bit value> IAC SE

The data sender specifies, with the 8 bit value, which party should handle output page size considerations and how. The code for DS is 1.

IAC SB NAOP DR <8 bit value> IAC SE

The data receiver specifies, with the 8 bit value, which party should handle output page size considerations and how. The code for DR is 0.

Default

DON'T NAOP
WON'T NAOP

In the default absence of negotiation concerning which party, data sender or data receiver, is handling output page size considerations, neither party is required to handle page size consideration and neither party is prohibited from handling page size consideration, but it is appropriate if at least the data receiver handles page size considerations albeit primitively.

Motivation for the Option

There appear to be four cases in which it is useful for the party at one end of a TELNET connection to communicate with the other party about the output page size:

- (a) the sender may wish the receiver to use its local knowledge of the printer page size to properly handle the page size;

- (b) the receiver may wish the sender to use its local knowledge to the data being sent to properly handle the page size;
- (c) the sender may wish to use its local knowledge of the data being sent to instruct the receiver in the proper handling of the page size; and
- (d) the receiver may wish to use its local knowledge of the printer size to instruct the sender in the proper handling of the page size.

An example of proper handling of the page size is for the receiver to hold off further output until instructed to continue when the lines being printed are about to overflow the scope face.

Description of the Option.

The data sender and data receiver use the 8 bit value along with the DS and DR SB commands as follows.

8 bit value	Meaning
0	command sender suggests he alone will handle output page-size considerations for the connection.
1 to 253	command sender suggests other party alone should handle output page-size considerations but suggests page size should be value given, in lines.
254	command sender suggests other party alone should handle output page size considerations but suggests page size should be considered infinity.
255	command sender suggests other party alone should handle output page size considerations and suggests nothing about how it should be done.

The guiding rules are that

- (1) if neither data receiver or data sender wants to handle output page size considerations, the data receiver must do it, and
- (2) if both data receiver or data sender want to handle output page size, the data sender gets to do it.

TELNET Output Page Size Option
NIC 20197 (Nov. 13, 1973)

The reasoning for the former rule is that if neither want to do it, then the default in the NAOP option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver makes.

Some sample negotiations are:

no subnegotiations

data receiver handles output page size considerations

IAC SB NAOP DS 66 IAC SE
IAC SB NAOP DR 0 IAC SE

data sender suggests data receiver handle output page size consideration data with suggested page size of 66 lines; receiver agrees.

IAC SB NAOP DR 255 IAC SE
IAC SB NAOP DS 0 IAC SE

data receiver suggests data sender handle page size consideration; sender refuses.

IAC SB NAOP DS 0 IAC SE
IAC SB NAOP DR 30 IAC SE

data sender wants to handle page size considerations; receiver agrees but notifies the sender the scope only has 30 IAC SE lines.

As with all option negotiation, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g.. another process starts).

At any time either party can disable further negotiation by giving the appropriate WON'T NAOP or DON'T NAOP command.

TELNET Output Carriage-Return Disposition Option
RFC 652, NIC 31155 (Oct. 25, 1974)

Request for Comments: 652
D. Crocker (UCLA-NMC)
Original file: [ISI]<DCROCKER>NAOCRD.TXT

TELNET Output Carriage-Return Disposition Option

1. Command name and code

NAOCRD 10 (Negotiate About Output Carriage-Return Disposition)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP TELNET options.

IAC DO NAOCRD

The data sender requests or agrees to negotiate about output carriage-return character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output carriage-returns.

IAC DON'T NAOCRD

The data sender refuses to negotiate about output carriage-return disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOCRD

The data receiver requests or agrees to negotiate about output carriage-return disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output carriage-returns.

IAC WON'T NAOCRD

The data receiver refuses to negotiate about output carriage-return disposition, or demands a return to the unnegotiated default mode.

IAC SB NAOCRD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle carriage-returns and what their disposition should be. The code for DS is 1.

TELNET Output Carriage-Return Disposition Option
RFC 652, NIC 31155 (Oct. 25, 1974)

IAC SB NAOCRD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle carriage-returns and what their disposition should be. The code for DR is 0.

3. Default

DON'T NAOCRD/WON'T NAOCRD.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output carriage-returns, neither party is required to handle carriage-returns and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles carriage-returns, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOP TELNET option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value along with the NAOCRD SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle carriage-returns, for the connection.
1 to 250	Command sender suggests that the other party alone should handle carriage-returns, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character. (See qualification, below.)
251	Not allowed, in order to be compatible with related TELNET options.
252	Command sender suggests that the other party alone handle carriage-returns, but suggests that they be discarded.
253	Not allowed, in order to be compatible with related TELNET options.

TELNET Output Carriage-Return Disposition Option
RFC 652, NIC 31155 (Oct. 25, 1974)

- 254 Command sender suggests that the other party alone should handle carriage-returns but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. (See qualification, below.) Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
- 255 Command sender suggests that the other party alone should handle carriage-returns and suggests nothing about how it should be done.

The guiding rules are that:

- a. if neither data receiver nor data sender wants to handle carriage-returns, the data receiver must do it, and
- b. if both data receiver and data sender want to handle carriage-returns, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOCRD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

Note that carriage-return delays, controlled by the data sender, must consist of NUL characters inserted immediately after the character in question. This is necessary due to the assynchrony of network transmissions. Due to the TELNET end-of-line convention, with carriage-returns followed by a linefeed, any NULs that would otherwise be placed after the carriage-return must be placed after the linefeed, regardless of any modifications that may additionally be made to the line feed (see NAOLFD TELNET option).

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOCRD or DON'T NAOCRD command.

Request for Comments: 653
D. Crocker (UCLA-NMC)
Original file: [ISI]<DCROCKER>NAOHTS.TXT

TELNET Output Horizontal Tabstops Option

1. Command name and code

NAOHTS 11 (Negotiate About Output Horizontal Tabstops)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP TELNET options.

IAC DO NAOHTS

The data sender requests or agrees to negotiate about output horizontal tabstops with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output horizontal tabstops.

IAC DON'T NAOHTS

The data sender refuses to negotiate about output horizontal tabstops with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOHTS

The data receiver requests or agrees to negotiate about output horizontal tabstops with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output horizontal tabstops.

IAC WON'T NAOHTS

The data receiver refuses to negotiate about output horizontal tabstops, or demands a return to the unnegotiated default mode.

IAC SB NAOHTS DS <8-bit value> ... <8-bit value> IAC SE

The data sender specifies, with the 8-bit value(s), which party should handle output horizontal tabstop considerations and what the stops should be. The code for DS is 1.

Preceding page blank

TELNET Output Horizontal Tabstops Option
RFC 653, NIC 31156 (Oct. 25, 1974)

IAC SB NAUHTS DR <8-bit value> ... <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value(s), which party should handle output horizontal tabstop considerations and what the stops should be. The code for DR is 0.

3. Default

DON'T NAUHTS/WON'T NAUHTS.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output horizontal tabstops, neither party is required to handle them and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles horizontal tabstops, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOP TELNET option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value(s) along with the DS and DR SB subcommands as follows (multiple 8-bit values are allowed only if each is greater than zero and less than 251):

c-bit value	Meaning
0	Command sender suggests that he alone will handle tabstops, for the connection.
1 to 250	Command sender suggests that the other party alone should handle tabstop considerations, but suggests that the indicated value(s) be used. The value(s) are the column numbers, relative to the physical left side of the printer page or terminal screen, that are to be set.
251 to 254	Not allowed, in order to be compatible with related TELNET options.
255	Command sender suggests that the other party alone should handle output tabstops and suggests nothing about how it should be done.

The guiding rules are that:

- a. if neither data receiver nor data sender wants to handle output horizontal tabstops, the data receiver must do it, and
- b. if both data receiver and data sender want to handle output horizontal tabstops, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOHTS option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOHTS or DON'T NAOHTS command.

TELNET Output Horizontal Tab Disposition Option
RFC 654, NIC 31157 (Oct. 25, 1974)

Request for Comments: 654
D. Crocker (UCLA-NMC)
Original file: [ISI]<DCROCKER>NAOHTD.TXT

TELNET Output Horizontal Tab Disposition Option

1. Command name and code

NAOHTD 12

(Negotiate About Output Horizontal Tab Disposition)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP TELNET options.

IAC DO NAOHTD

The data sender requests or agrees to negotiate about output horizontal tab character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output horizontal tab character considerations.

IAC DON'T NAOHTD

The data sender refuses to negotiate about output horizontal tab characters with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOHTD

The data receiver requests or agrees to negotiate about output horizontal tab characters with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output horizontal tab character considerations.

IAC WON'T NAOHTD

The data receiver refuses to negotiate about output horizontal tab characters, or demands a return to the unnegotiated default mode.

TELNET Output Horizontal Tab Disposition Option
RFC 654, NIC 31157 (Oct. 25, 1974)

IAC SB NAOHTD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle output horizontal tab characters and what their disposition should be. The code for DS is 1.

IAC SB NAOHTD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle output horizontal tab characters and what their disposition should be. The code for DR is 0.

3. Default

DON'T NAOHTD/WON'T NAOHTD.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output horizontal tab character considerations, neither party is required to handle horizontal tab characters and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles horizontal tab character considerations, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOP TELNET option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value along with the DS and Dh SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle horizontal tab characters, for the connection.
1 to 250	Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle horizontal tabs, but suggests that each occurrence of the character be replaced by a space

TELNET Output Horizontal Tab Disposition Option
RFC 654, NIC 31157 (Oct. 25, 1974)

- 252 Command sender suggests that the other party alone handle horizontal tabs, but suggests that they be discarded.
- 253 Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that tabbing be simulated.
- 254 Command sender suggests that the other party alone should handle horizontal tab characters, but suggests that waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
- 255 Command sender suggests that the other party alone should handle output horizontal tabs and suggests nothing about how it should be done.

The guiding rules are that:

- a. if neither data receiver nor data sender wants to handle output horizontal tab characters, the data receiver must do it, and
- b. if both data receiver and data sender wants to handle output horizontal tab characters, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOHTD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

Simulation is defined as the replacement of the horizontal tab character by enough spaces to move the printer head (or line-pointer) to the next horizontal tab stop.

Note that delays, controlled by the data sender, must consist of NUL characters inserted immediately after the horizontal tab character. This is necessary due to the assynchrony of network transmissions.

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOHTD or DON'T NAOHTD command.

TELNET Output Formfeed Disposition Option
RFC 655, NIC 31158 (Oct. 25, 1974)

request for Comments: 655
D. Crocker (UCLA-NMC)
Original file: [ISI]<DCROCKER>NAOFFD.TXT

TELNET Output Formfeed Disposition Option

1. Command name and code

NAOFFD 13

(Negotiate About Output Formfeed Disposition)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP TELNET Options specifications.

IAC DO NAOFFD

The data sender requests or agrees to negotiate about output formfeed disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output formfeeds.

IAC DON'T NAOFFD

The data sender refuses to negotiate about output formfeed disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOFFD

The data receiver requests or agrees to negotiate about output formfeed disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output formfeeds.

IAC WON'T NAOFFD

The data receiver refuses to negotiate about output formfeed disposition, or demands a return to the unnegotiated default mode.

Preceding page blank

TELNET Output Formfeed Disposition Option
RFC 655, NIC 31158 (Oct. 25, 1974)

IAC SB NAOFFD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle formfeeds and what their disposition should be. The code for DS is 1.

IAC SB NAOFFD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle formfeeds and what their disposition should be. The code for DR is 0.

3. Default

DON'T NAOFFD/WON'T NAOFFD.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output formfeeds, neither party is required to handle formfeeds and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles formfeed considerations, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOFFD TELNET option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle formfeeds, for the connection.
1 to 250	Command sender suggests that the other party alone should handle formfeeds, but suggests that the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle formfeeds, but suggests that each occurrence of the character be replaced by carriage-return/line-feed.

TELNET Output Formfeed Disposition Option
RFC 655, NIC 31158 (Oct. 25, 1974)

- 252 Command sender suggests that the other party alone handle formfeeds, but suggests that they be discarded.
- 253 Command sender suggests that the other party alone should handle formfeeds, but suggests that formfeeds be simulated.
- 254 Command sender suggests that the other party alone should handle output formfeeds but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
- 255 Command sender suggests that the other party alone should handle output formfeeds and suggests nothing about how it should be done.

The guiding rules are that:

- a. if neither data receiver nor data sender wants to handle output formfeeds, the data receiver must do it, and
- b. if both data receiver and data sender want to handle output formfeeds, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOFFD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

Simulation is defined as the replacement of the formfeed character by enough line-feeds (only) to advance the paper (or line-pointer) to the top of the next page (or to the top of the terminal screen).

Note that delays, controlled by the data sender, must consist of NUL characters inserted immediately after the formfeed character. This is necessary due to the assynchrony of network transmission.

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAUFFD or DON'T NAOFFD command.

TELNET Output Vertical Tabstops Option
RFC 656, NIC 31159 (Oct. 25, 1974)

Request for Comments: 656
D. Crocker (UCLA-NMC)
Original file: [ISI]<DCROCKER>NAOVTS.TXT

TELNET Output Vertical Tabstops Option

1. Command name and code

NAOVTS 14

(Negotiate About Vertical Tabstops)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP TELNET Options specifications.

IAC DO NAOVTS

The data sender requests or agrees to negotiate about output vertical tabstops with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output vertical tabstop considerations.

IAC DON'T NAOVTS

The data sender refuses to negotiate about output vertical tabstops with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOVTS

The data receiver requests or agrees to negotiate about output vertical tabstops with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output vertical tabstop considerations.

IAC WON'T NAOVTS

The data receiver refuses to negotiate about output vertical tabstops, or demands a return to the unnegotiated default mode.

IAC SB NAOVTS DS <8-bit value> ... <8-bit value> IAC SE

The data sender specifies, with the 8-bit value(s), which party

TELNET Output Vertical Tabstops Option
RFC 656, NIC 31159 (Oct. 25, 1974)

should handle output vertical tabstop considerations and what the stops should be. The code for DS is 1.

IAC SB NAOVTS DR <8-bit value> ... <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value(s), which party should handle output vertical tabstop considerations and what the stops should be. The code for DR is 0.

3. Default

DON'T NAOVTS/WON'T NAOVTS.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output vertical tabstop considerations, neither party is required to handle vertical tabstops and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles vertical tabstop considerations, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOVTS TELNET option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value(s) along with the DS and DR SB commands as follows (multiple 8-bit values are allowed only if each is greater than zero and less than 251):

8-bit value	Meaning
0	Command sender suggests that he alone will handle the vertical tabstops, for the connection.
1 to 250	Command sender suggests that the other party alone should handle the stops, but suggests that the indicated value(s) be used. Each value is the line number, relative to the top of the printer page or terminal screen, that is to be set as a vertical tabstop.
251 to 254	Not allowed, in order to be compatible with related TELNET options.
255	Command sender suggests that the other party alone should handle output vertical tabstops and suggests nothing about how it should be done.

The guiding rules are that:

- a. if neither data receiver nor data sender wants to handle output vertical tabstops, the data receiver must do it, and
- b. if both data receiver and data sender want to handle output vertical tabstops, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOVTS option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make. This is necessary due to the assynchrony of network transmissions.

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOVTS or DON'T NAOVTS command.

TELNET Output Vertical Tab Disposition Option
RFC 657, NIC 31160 (Oct. 25, 1974)

Request for Comments: 657
D. Crocker (UCLA-NMC)
Original file: [ISI]<DCROCKER>NAOVTD.TXT

TELNET Output Vertical Tab Disposition Option

1. Command name and code

NAOVTD 15

(Negotiate About Output Vertical Tab Disposition)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP TELNET Options specifications.

IAC DO NAOVTD

The data sender requests or agrees to negotiate about output vertical tab character disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output vertical tab character considerations.

IAC DON'T NAOVTD

The data sender refuses to negotiate about output vertical tab character disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOVTD

The data receiver requests or agrees to negotiate about output vertical tab character disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output vertical tab character considerations.

IAC WON'T NAOVTD

The data receiver refuses to negotiate about output vertical tab character disposition, or demands a return to the unnegotiated default mode.

TELNET Output Vertical Tab Disposition Option
RFC 657, NIC 31160 (Oct. 25, 1974)

IAC SB NAOVTD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle output vertical tab characters and what their disposition should be. The code for DS is 1.

IAC SB NAOVID DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle output vertical tab characters and what their disposition should be. The code for DR is 0.

3. Default

DON'T NAOVTD/WON'T NAOVTD.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output vertical tab character considerations, neither party is required to handle vertical tab characters and neither party is prohibited from handling them; but it is appropriate if at least the data receiver handles vertical tab character considerations, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOVTD TELNET option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value along with the DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle vertical tab characters, for the connection.
1 to 250	Command sender suggests that the other party alone should handle tab characters, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character.
251	Command sender suggests that the other party alone handle vertical tabs, but suggests that each occurrence of the character be replaced by carriage-return/linefeed.

TELNET Output Vertical Tab Disposition Option
RFC 657, NIC 31160 (Oct. 25, 1974)

- 252 Command sender suggests that the other party alone handle vertical tabs, but suggests that they be discarded.
- 253 Command sender suggests that the other party alone should handle tab characters, but suggests that tabbing be simulated.
- 254 Command sender suggests that the other party alone should handle the output disposition but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.
- 255 Command sender suggests that the other party alone should handle the output disposition and suggests nothing about how it should be done.

The guiding rules are that:

- a. if neither data receiver nor data sender wants to handle the output vertical tab characters, the data receiver must do it, and
- b. if both data receiver and data sender want to handle the output vertical tab characters, the data sender gets to do it.

The reasoning for the former rule is that if neither want to do it, then the default in the NAOVTD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

Simulation is defined as the replacement of the character by enough line-feeds (only) to advance the paper (or line-pointer) to the next vertical tab stop.

Note that delays, controlled by the data sender, must consist of NUL characters, inserted immediately after the line-feed character. This is necessary due to the assynchrony of network transmissions.

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

TELNET Output Vertical Tab Disposition Option
RFC 657, NIC 31160 (Oct. 25, 1974)

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOVTD or DON'T NAOVTD command.

TELNET Output Linefeed Disposition
RFC 658, NIC 31161 (Oct. 25, 1974)

Request for Comments: 658
D. Crocker (UCLA-NMC)
Original file: [ISI]<DCROCKER>NAOLFD.TXT

TELNET Output Linefeed Disposition

1. Command name and code

NAOLFD 1b

(Negotiate About Output Linefeed Disposition)

2. Command meanings

In the following, we are discussing a simplex connection, as described in the NAOL and NAOP TELNET Options.

IAC DO NAOLFD

The data sender requests or agrees to negotiate about output linefeed disposition with the data receiver. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver is assumed to be handling output linefeed considerations.

IAC DON'T NAOLFD

The data sender refuses to negotiate about output linefeed disposition with the data receiver, or demands a return to the unnegotiated default mode.

IAC WILL NAOLFD

The data receiver requests or agrees to negotiate about output linefeed disposition with the sender. In the case where agreement has been reached and in the absence of further subnegotiations, the data receiver alone is assumed to be handling output linefeed considerations.

IAC WON'T NAOLFD

The data receiver refuses to negotiate about output linefeed disposition, or demands a return to the unnegotiated default mode.

TELNET Output Linefeed Disposition
RFC 658, NIC 31161 (Oct. 25, 1974)

IAC SB NAOLFD DS <8-bit value> IAC SE

The data sender specifies, with the 8-bit value, which party should handle output linefeeds and what their disposition should be. The code for DS is 1.

IAC SB NAOLFD DR <8-bit value> IAC SE

The data receiver specifies, with the 8-bit value, which party should handle output linefeeds and what their disposition should be. The code for DR is 0.

3. Default

DON'T NAOLFD/WON'T NAOLFD.

In the default absence of negotiations concerning which party, data sender or data receiver, is handling output linefeed considerations, neither party is required nor prohibited from handling linefeeds; but it is appropriate if at least the data receiver handles them, albeit primitively.

4. Motivation for the Option

Please refer to section 4 of the NAOL and of the NAOLFD TELNET option descriptions.

5. Description of the Option

The data sender and the data receiver use the 8-bit value along with DS and DR SB commands as follows:

8-bit value	Meaning
0	Command sender suggests that he alone will handle linefeeds, for the connection.
1 to 250	Command sender suggests that the other party alone should handle linefeeds, but suggests that a delay of the indicated value be used. The value is the number of character-times to wait or number of NULs to insert in the data stream before sending the next data character. (See qualifications, below.)
251	Not allowed, in order to be compatible with related TELNET options.
252	Command sender suggests that the other party alone

handle linefeeds, but suggests that they be discarded.

253 Command sender suggests that the other party alone should handle linefeeds, but suggests that linefeeds be simulated.

254 Command sender suggests that the other party alone should handle output linefeeds but suggests waiting for a character to be transmitted (on the other simplex connection) before sending more data. (See qualifications, below.) Note that, due to the assynchrony of the two simplex connections, phase problems can occur with this option.

255 Command sender suggests that the other party alone should handle output linefeeds and suggests nothing about how it should be done.

The guiding rules are that:

- a. if neither data receiver nor data sender wants to handle output linefeeds, the data receiver must do it, and
- b. if both data receiver and data sender want to handle output linefeed disposition, the data sender gets to do it.

The reasoning for the former rule is that if neither wants to do it, then the default in the NAOLFD option dominates. If both want to do it, the sender, who is presumed to have special knowledge about the data, should be allowed to do it, taking into account any suggestions the receiver may make.

Simulation is defined as the replacement of the linefeed character by new-line (see following) and enough blanks to move the print head (or line pointer) to the same lateral position it occupied just prior receiving the linefeed. To avoid infinite recursion, such simulation is allowed only for linefeed characters that are not immediately preceded by carriage-returns (that is, part of a TELNET new-line combination). It is assumed that linefeed simulation will be necessary for printers that do not have a separate linefeed (like the IBM 2741); in this case, end-of-line character padding can be specified through NAOCRD. Any padding ($0 < \text{8-bit-value} < 251$) of linefeed characters is to be done for ALL linefeed characters.

Note that delays, controlled by the data sender, must consist of NUL characters inserted immediately after the character. This is necessary due to the assynchrony of network transmissions. Additionally, due to the presence of the TELNET end-of-line

TELNET Output Linefeed Disposition
RFC 658, NIC 31161 (Oct. 25, 1974)

convention, it may be necessary to add carriage-return padding or delay after the associated linefeed (see NAOCRD TELNET option).

As with all option negotiations, neither party should suggest a state already in effect except to refuse to negotiate; changes should be acknowledged; and once refused, an option should not be resuggested until "something changes" (e.g., another process starts).

At any time, either party can disable further negotiation by giving the appropriate WON'T NAOLFD or DON'T NAOLFD command.

TELNET Extended ASCII Option

1. Command Name and code.

EXTEND-ASCII 17

2. Command Meanings.

IAC WILL EXTEND-ASCII

The sender of this command requests permission to begin transmitting, or confirms that it may now begin transmitting extended ASCII, where additional 'control' bits are added to normal ASCII, which are treated specially by certain programs on the host computer.

IAC WON'T EXTEND-ASCII

If the connection is already being operated in extended ASCII mode, the sender of this command demands that the receiver begin transmitting data characters in standard NVT ASCII. If the connection is not already being operated in extended ASCII mode, The sender of this command refuses to begin transmitting extended ASCII.

IAC DO EXTEND-ASCII

The sender of this command requests that the receiver begin transmitting, or confirms that the receiver of this command is allowed to begin transmitting extended ASCII.

IAC DON'T EXTEND-ASCII

The sender of this command demands that the receiver of this command stop or not start transmitting data in extended ASCII mode.

IAC SB EXTASC

<high order bits (bits 15-8)> <low order bits (bits 7-0)>

IAC SE

This command transmits an extended ASCII character in the form of two 8-bit bytes. Each 8-bit byte contains 8 data bits.

3. Default

DON'T EXTEND-ASCII

WON'T EXTEND-ASCII

i.e. only use standard NVT ASCII

4. Motivation.

Several sites on the net, for example, SU-AI and MII-AI, use keyboards which use almost all 128 characters as printable characters, and use one or more additional bits as 'control' bits as command modifiers or to separate textual input from command input to programs. Without these additional bits, several characters cannot be entered as text because they are used for control purposes, such as the greek letter 'beta' which on a TELNET connection is Control-C and is used for stopping ones job. In addition there are several commonly used programs at these sites require these additional bits to be run effectively. Hence it is necessary to provide some means of sending characters larger than 8 bits wide.

5. Description of the option.

This option is to allow the transmission of extended ASCII.

Experience has shown that most of the time, 7-bit ASCII is typed, with an occasional 'control' character used. Hence, it is expected normal NVT ASCII would be used for 7-bit ASCII and that extended- ASCII be sent as an escape character sequence.

The exact meaning of these additional bits depends on the user program. At SU-AI and at MII-AI, the first two bits beyond the normal 7-bit ASCII are passed on to the user program and are denoted as follows.

Bit 8 (or 200 octal) is the CONTROL bit

Bit 9 (or 400 octal) is the META bit

(Note that 'CONTROL' is used in a non-standard way here; that is, it usually refers to codes 0-37 in NWG ASCII. CONTROL and META are echoed by prefixing the normal character with 013 (integral symbol) for CONTROL and 014 (plus-minus) for META. If both are present, it is known as CONTROL-META and echoed as 013 014 7-bit character.)

b. Description of Stanford Extended ASCII Characters

In this section, the extended graphic character set used at SU-AI is described for reference, although this specific character set is not required as part of the extended ASCII Telnet option. Characters described as 'hidden' are alternate graphic interpretations of codes normally used as format effectors, used by certain typesetting programs.

Code	Graphic represented
000	null (hidden vertically centered dot)
001	downward arrow
002	alpha (all Greek letters are lowercase)
003	beta
004	logical and (caret)
005	logical not (dash with downward extension)
006	epsilon
007	pi
010	lambda
011	tab (hidden gamma)
012	linefeed (hidden delta)
013	vertical tab (hidden integral)
014	formfeed (hidden plus-minus)
015	carriage return (hidden circled-plus)
016	infinity
017	del (partial differential)
020	proper subset (right-opening horseshoe)
021	proper superset (left-opening horseshoe)
022	intersection (down-opening horseshoe)
023	union (up-opening horseshoe)
024	universal quantifier (upside-down A)
025	existential quantifier (backwards E)
026	circled-times
027	left-right double headed arrow
030	underbar
031	right pointing arrow

032	tilde
033	not-equal
034	less-than-or-equal
035	greater-than-or-equal
036	equivalence (column of 3 horizontal bars)
037	logical or (V shape)
040-135	as in standard ASCII
136	upward pointing arrow
137	left pointing arrow
140-174	as in standard ASCII
175	altmode (prints as lozenge)
176	right brace
177	rubout (hidden circumflex)

TELNET Extended-Options-List Option
NIC 16239

TELNET Extended-Options-List Option

1. Command name and code.

EXTENDED-OPTIONS-LIST (EXOPL) 255

2. Command meanings.

IAC DO EXOPL

The sender of this command REQUESTS that the receiver of this command begin negotiating, or confirms that the receiver of this command is expected to begin negotiating, TELNET options which are on the "Extended Options List."

IAC WILL EXOPL

The sender of this command requests permission to begin negotiating, or confirms that it will begin negotiating, TELNET options which are on the "Extended Options List."

IAC WON'T EXOPL

The sender of this command REFUSES to negotiate, or to continue negotiating, options on the "Extended Options List."

IAC DON'T EXOPL

The sender of this command DEMANDS that the receiver conduct no further negotiation of options on the "Extended Options List."

IAC SB EXOPL <subcommand>

The subcommand contains information required for the negotiation of an option of the "Extended Options List." The format of the subcommand is discussed in section 5 below.

3. Default.

WON'T EXOPL, DON'T EXOPL

i.e., negotiation of options on the "Extended Options List" is not permitted.

4. Motivation for the option.

Eventually, a 257th TELNET option will be needed and there is currently no way to support it. The proposed option will extend the option list for another 256 options in a manner which is easy to

TELNET Extended-Options-List Option
NIC 16239

implement. The option is proposed now, rather than later (probably much later), in order to reserve the option number (255).

5. An abstract description of the option.

The EXOPL option has five subcommand codes: WILL, WON'T, DO, DON'T, and SB. They have exactly the same meanings as the TELNET commands with the same names, and are used in exactly the same way. For consistency, these subcommand codes will have the same values as the TELNET command codes (250-254). Thus, the format for negotiating a specific option on the "Extended Options List" (once both parties have agreed to use it) is:

IAC SB EXOPL DO/DON'T/WILL/WON'T <option code>

Once both sides have agreed to use the specific option specified by <option code>, subnegotiation may be required. In this case the format to be used is:

IAC SB EXOPL SB <option code> <parameters>

USAS
X3.4-1968
Revision of
X3.4-1967

USA Standard Code for Information Interchange

Sponsor
Business Equipment Manufacturers Association

Approved October 10, 1968
United States of America Standards Institute

Contents

SECTION	PAGE
1. Scope	6
2. Standard Code	6
3. Character Representation and Code Identification	7
4. Legend	7
4.1 Control Characters.....	7
4.2 Graphic Characters	7
5. Definitions	8
5.1 General	8
5.2 Control Characters.....	8
5.3 Graphic Characters	9
6. General Considerations	9
Appendixes	
Appendix A Design Considerations for the Coded Character Set	11
A1. Introduction	11
A2. Considerations Affecting the Standard	11
A3. Set Size	11
A4. Set Structure	11
A5. Choice of Graphics	11
A6. Graphic Subset Structure.....	12
A7. Control Subset Content and Structure	13
Appendix B Notes on Application	13
B1. Introduction	13
B2. Character Substitutions	13
B3. Related Larger and Smaller Sets	14
B4. International Considerations	14
B5. Communications Considerations	14
Appendix C Original Criteria	14
C1. Introduction	14
C2. Criteria	15
Appendix D Terminology.....	15

Preceding page blank

(163)

USA Standard Code for Information Interchange

1. Scope

This coded character set is to be used for the general interchange of information among information processing systems, communication systems, and associated equipment.

2. Standard Code

B i t s	b ₇ → b ₆ → b ₅ →				0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1	
	b ₄	b ₃	b ₂	b ₁	COLUMN	ROW	0	1	2	3	4	5	6
	0	0	0	0	0	NUL	DLE	SP	0	•	P	‘	p
	0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
	0	0	1	0	2	STX	DC2	"	2	B	R	b	r
	0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
	0	1	0	0	4	EOT	DC4	\$	4	D	T	d	t
	0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
	0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
	0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
	1	0	0	0	8	BS	CAN	(8	H	X	h	x
	1	0	0	1	9	HT	EM)	9	I	Y	i	y
	1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
	1	0	1	1	11	VT	ESC	+	;	K	[k	{
	1	1	0	0	12	FF	FS	,	<	L	\	l	l
	1	1	0	1	13	CR	GS	-	=	M]	m	}
	1	1	1	0	14	SO	RS	.	>	N	~	n	~
	1	1	1	1	15	SI	US	/	?	O	—	o	DEL

3. Character Representation and Code Identification

The standard 7-bit character representation, with b_7 the high-order bit and b_1 the low-order bit, is shown below:

EXAMPLE: The bit representation for the character "K," positioned in column 4, row 11, is

b_7	b_6	b_5	b_4	b_3	b_2	b_1
1	0	0	1	0	1	1

The code table position for the character "K" may also be represented by the notation "column 4, row 11" or alternatively as "4/11." The decimal equivalent of the binary number formed by bits b_7 , b_6 , and b_5 , collectively, forms the column number, and the decimal equivalent of the binary number formed by bits b_4 , b_3 , b_2 , and b_1 , collectively, forms the row number.

The standard code may be identified by the use of the notation ASCII or USASCII.

The notation ASCII (pronounced as'-key) or USASCII (pronounced you-sas'-key) should ordinarily be taken to mean the code prescribed by the latest issue of the standard. To explicitly designate a particular (perhaps prior) issue, the last two digits of the year of issue may be appended, as, "ASCII 63" or "USASCII 63".

4. Legend

4.1 Control Characters

NUL	Null	DLE	Data Link Escape (CC)
SOH	Start of Heading (CC)	DC1	Device Control 1
STX	Start of Text (CC)	DC2	Device Control 2
ETX	End of Text (CC)	DC3	Device Control 3
EOT	End of Transmission (CC)	DC4	Device Control 4 (Stop)
ENQ	Enquiry (CC)	NAK	Negative Acknowledge (CC)
ACK	Acknowledge (CC)	SYN	Synchronous Idle (CC)
BEL	Bell (audible or attention signal)	ETB	End of Transmission Block (CC)
BS	Backspace (FE)	CAN	Cancel
HT	Horizontal Tabulation (punched card skip) (FE)	EM	End of Medium
LF	Line Feed (FE)	SUB	Substitute
VT	Vertical Tabulation (FE)	ESC	Escape
FF	Form Feed (FE)	FS	File Separator (IS)
CR	Carriage Return (FE)	GS	Group Separator (IS)
SO	Shift Out	RS	Record Separator (IS)
SI	Shift In	US	Unit Separator (IS)
		DEL	Delete ¹

NOTE: (CC) Communication Control
(FE) Format Effector
(IS) Information Separator

¹In the strict sense, DEL is not a control character. (See 5.2.)

<u>Column/Row</u>	<u>Symbol</u>	<u>Name</u>
2/0	SP	Space (Normally Non-Printing)
2/1	!	Exclamation Point
2/2	"	Quotation Marks (Diaeresis ²)
2/3	#	Number Sign ^{3,4}
2/4	\$	Dollar Sign
2/5	%	Percent
2/6	&	Ampersand
2/7	'	Apostrophe (Closing Single Quotation Mark; Acute Accent ²)
2/8	(Opening Parenthesis
2/9)	Closing Parenthesis
2/10	*	Asterisk
2/11	+	Plus
2/12	,	Comma (Cedilla ²)
2/13	-	Hyphen (Minus)
2/14	.	Period (Decimal Point)
2/15	/	Slant
3/10	:	Colon
3/11	;	Semicolon
3/12	<	Less Than
3/13	=	Equals
3/14	>	Greater Than
3/15	?	Question Mark
4/0	@	Commercial At ³
5/11	[Opening Bracket ³
5/12	\	Reverse Slant ³
5/13]	Closing Bracket ³
5/14	^	Circumflex ^{2,3}
5/15	—	Underline
6/0	~	Grave Accent ^{2,3} (Opening Single Quotation Mark)
7/11	{	Opening Brace ³
7/12		Vertical Line ³
7/13	}	Closing Brace ³
7/14	~	Overline ³ (Tilde ² ; General Accent ²)

²The use of the symbols in 2/2, 2/7, 2/12, 5/14, 6/0, and 7/14 as diacritical marks is described in Appendix A, A5.2.

³These characters should not be used in international interchange without determining that there is agreement between sender and recipient. (See Appendix B4.)

⁴In applications where there is no requirement for the symbol #, the symbol £ may be used in position 2/3.

5. Definitions

5.1 General

(CC) Communication Control: A functional character intended to control or facilitate transmission of information over communication networks.

(FE) Format Effector: A functional character which controls the layout or positioning of information in printing or display devices.

(IS) Information Separator: A character which is used to separate and qualify information in a logical sense. There is a group of four such characters, which are to be used in a hierarchical order.

5.2 Control Characters

NUL (Null): The all-zeros character which may serve to accomplish time fill and media fill.

SOH (Start of Heading): A communication control character used at the beginning of a sequence of characters which constitute a machine-sensible address or routing information. Such a sequence is referred to as the "heading." An STX character has the effect of terminating a heading.

STX (Start of Text): A communication control character which precedes a sequence of characters that is to be treated as an entity and entirely transmitted through to the ultimate destination. Such a sequence is referred to as "text." STX may be used to terminate a sequence of characters started by SOH.

ETX (End of Text): A communication control character used to terminate a sequence of characters started with STX and transmitted as an entity.

EOT (End of Transmission): A communication control character used to indicate the conclusion of a transmission, which may have contained one or more texts and any associated headings.

ENQ (Enquiry): A communication control character used in data communication systems as a request for a response from a remote station. It may be used as a "Who Are You" (WRU) to obtain identification, or may be used to obtain station status, or both.

ACK (Acknowledge): A communication control character transmitted by a receiver as an affirmative response to a sender.

BEL (Bell): A character for use when there is a need to call for human attention. It may control alarm or attention devices.

BS (Backspace): A format effector which controls the movement of the printing position one printing space backward on the same printing line. (Applicable also to display devices.)

HT (Horizontal Tabulation): A format effector which controls the movement of the printing position to the next in a series of predetermined positions along the printing line. (Applicable also to display devices and the skip function on punched cards.)

LF (Line Feed): A format effector which controls the movement of the printing position to the next printing line. (Applicable also to display devices.) Where appropriate, this character may have the meaning "New Line" (NL), a format effector which controls the movement of the printing point to the first printing position on the next printing line. Use of this convention requires agreement between sender and recipient of data.

VT (Vertical Tabulation): A format effector which controls the movement of the printing position to the next in a series of predetermined printing lines. (Applicable also to display devices.)

FF (Form Feed): A format effector which controls the movement of the printing position to the first predetermined printing line on the next form or page. (Applicable also to display devices.)

CR (Carriage Return): A format effector which controls the movement of the printing position to the first printing position on the same printing line. (Applicable also to display devices.)

SO (Shift Out): A control character indicating that the code combinations which follow shall be interpreted as outside of the character set of the standard code table until a Shift In character is reached.

SI (Shift In): A control character indicating that the code combinations which follow shall be interpreted according to the standard code table.

DLE (Data Link Escape): A communication control character which will change the meaning of a limited number of contiguously following characters. It is used exclusively to provide supplementary controls in data communication networks.

DC1, DC2, DC3, DC4 (Device Controls): Characters for the control of ancillary devices associated with data processing or telecommunication systems, more especially switching devices "on" or "off." (If a single "stop" control is required to interrupt or turn off ancillary devices, DC4 is the preferred assignment.)

NAK (Negative Acknowledge): A communication control character transmitted by a receiver as a negative response to the sender.

SYN (Synchronous Idle): A communication control character used by a synchronous transmission system in the absence of any other character to provide a signal from which synchronism may be achieved or retained.

ETB (End of Transmission Block): A communication control character used to indicate the end of a block of data for communication purposes. ETB is used for blocking data where the block structure is not necessarily related to the processing format.

CAN (Cancel): A control character used to indicate that the data with which it is sent is in error or is to be disregarded.

EM (End of Medium): A control character associated with the sent data which may be used to identify the physical end of the medium, or the end of the used, or wanted, portion of information recorded on a medium.

(The position of this character does not necessarily correspond to the physical end of the medium.)

SUB (Substitute): A character that may be substituted for a character which is determined to be invalid or in error.

ESC (Escape): A control character intended to provide code extension (supplementary characters) in general information interchange. The Escape character itself is a prefix affecting the interpretation of a limited number of contiguously following characters.

FS (File Separator), GS (Group Separator), RS (Record Separator), and US (Unit Separator): These information separators may be used within data in optional fashion, except that their hierarchical relationship shall be: FS is the most inclusive, then GS, then RS, and US is least inclusive. (The content and length of a File, Group, Record, or Unit are not specified.)

DEL (Delete): This character is used primarily to "erase" or "obliterate" erroneous or unwanted characters in perforated tape. (In the strict sense, DEL is not a control character.)

5.3 Graphic Characters

SP (Space): A normally non-printing graphic character used to separate words. It is also a format effector which controls the movement of the printing position, one printing position forward. (Applicable also to display devices.)

6. General Considerations

6.1 This standard does not define the means by which the coded set is to be recorded in any physical medium, nor does it include any redundancy or define techniques for error control. Further, this standard does not define data communication character structure, data communication formats, code extension techniques, or graphic representation of control characters.

6.2 Deviations from the standard may create serious difficulties in information interchange and should be used only with full cognizance of the parties involved.

6.3 The relative sequence of any two characters, when used as a basis for collation, is defined by their binary values.

6.4 No specific meaning is prescribed for any of the graphics in the code table except that which is understood by the users. Furthermore, this standard does not specify a type style for the printing or display of the various graphic characters. In specific applications, it may be desirable to employ distinctive styling of individual graphics to facilitate their use for specific purposes as, for example, to stylize the graphics in code positions 2/1 and 5/14 into those frequently associated with logical OR (1) and logical NOT (\neg), respectively.

6.5 The appendixes to this standard contain additional information on the design and use of this code.

Appendices

(These Appendixes are not a part of USA Standard Code for Information Interchange, X3.4-1968, but are included to facilitate its use.)

Appendix A Design Considerations for the Coded Character Set

A1. Introduction

The standard coded character set is intended for the interchange of information among information processing systems, communication systems, and associated equipment.

A2. Considerations Affecting the Code

There were many considerations that determined the set size, set structure, character selection, and character placement of the code. Among these were (not listed in order of priority):

- (1) Need for adequate number of graphic symbols
- (2) Need for adequate number of device controls, format effectors, communication controls, and information separators
- (3) Desire for a non-ambiguous code, i.e., one in which every code combination has a unique interpretation
- (4) Physical requirements of media and facilities
- (5) Error control considerations
- (6) Special interpretation of the all-zeros and all-ones characters
- (7) Ease in the identification of classes of characters
- (8) Data manipulation requirements
- (9) Collating conventions
 - (a) Logical
 - (b) Historical
- (10) Keyboard conventions
 - (a) Logical
 - (b) Historical
- (11) Other set sizes
- (12) International considerations
- (13) Programming languages
- (14) Existing coded character sets.

A3. Set Size

A 7-bit set is the minimum size that will meet the requirements for graphics and controls in applications involving general information interchange.

A4. Set Structure

A4.1 In discussing the set structure it is convenient to divide the set into 8 columns and 16 rows as indicated in this standard.

A4.2 It was considered essential to have a dense subset which contained only graphics. For ease of identification this graphic subset was placed in six contiguous columns.

A4.3 The first two columns were chosen for the controls for the following three reasons:

(1) The character NUL by its definition has the location 0/0 in the code table. NUL is broadly considered a control character.

(2) The representations in column 7 were felt to be most susceptible to simulation by a particular class of transmission error—one which occurs during an idle condition on asynchronous systems.

(3) To permit the considerations of graphic subset structure described in A6 to be satisfied, the two columns of controls had to be adjacent.

A4.4 The character set was structured to enable the easy identification of classes of graphics and controls.

A5. Choice of Graphics

A5.1 Included in the set are the numerals 0 through 9, upper and lower cases of the alphabetic letters A through Z, and those punctuation, mathematical, and business symbols considered most useful. The set includes a number of characters commonly encountered in programming languages. In particular, all the COBOL and FORTRAN graphics are included.

A5.2 In order to permit the representation of languages other than English, one diacritical (or accent) mark has been included, and provision has been made for the use of five punctuation symbols alternately as diacritical marks. The pairing of these punctuation symbols with their corresponding diacritical marks was done to facilitate the design of a typeface which would be acceptable for both uses.

These arrangements are:

Col/Row	Code Table Symbol	Punctuation	Use	Diacritical
2/2	"	Quotation Marks		Diaeresis
2/7	'	Apostrophe		Acute Accent
2/12	,	Comma		Cedilla
5/14	~	(None)		Circumflex
6/0	`	Opening Single		
7/14	~	Quotation Mark	Grave Accent	
		Overline	Tilde*	

*May also be used for other accents not specifically provided.

A5.3 The character *overline* is shown as it is in the code table to suggest a means of avoiding confusion with *underline*, and to reflect its use as *tilde*. The character *vertical line* is shown as it is in the code table to avoid confusion with the solid vertical bar frequently used as a logical operator, which may be found in some systems as a graphic stylization of *exclamation point*.

A6. Graphic Subset Structure

A6.1 The basic structure of the dense graphic subset was influenced by logical collating considerations, the requirements of simply related 6-bit sets, and the needs of typewriter-like devices. For information processing, it is desirable that the characters be arranged in such a way as to minimize both the operating time and the hardware components required for ordering and sequencing operations. This requires that the relative order of characters, within classes, be such that a simple comparison of the binary codes will result in information being ordered in a desired sequence.

A6.2 Conventional usage requires that SP (*space*) be ahead of any other symbol in a collatable set. This permits a name such as "JOHNS" to collate ahead of a name such as "JOHNSON." The requirement that punctuation symbols such as *comma* also collate ahead of the alphabet ("JOHNS, A" should also collate before "JOHNSON") establishes the special symbol locations, including SP, in the first column of the graphic subset.

A6.3 To simplify the design of typewriter-like devices, it is desirable that there be only a common 1-bit difference between characters to be paired on keytops. This, together with the requirements for a contiguous alphabet, and the collating requirements outlined above, resulted in the placement of the alphabet in the last four columns of the graphic subset and the placement of the numerals in the second column of the graphic subset.

A6.4 It is expected that devices having the capability of printing only 64 graphic symbols will continue to be important. It may be desirable to arrange these devices to print one symbol for the bit pattern of both upper and lower case of a given alphabetic letter. To facilitate this, there should be a single bit difference between the upper and lower case representations of any given letter. Combined with the requirement that a given case of the alphabet be contiguous, this dictated the assignment of the alphabet, as shown, in columns 4 through 7.

A6.5 To minimize ambiguity caused by the use of a 64-graphic device as described above, it is desirable, to the degree possible, that each character in column 6 or 7 differ little in significance from the corresponding character in column 4 or 5. In certain cases, this was not possible.

A6.6 The assignment of *reverse slant* and *vertical line*, the *brackets* and *braces*, and *circumflex* and *overline* were made with a view to the considerations of A6.5.

A6.7 The resultant structure of "specials" (S), "digits" (D), and "alphabetics" (A) does not conform to the most prevalent collating convention (S-A-D) because of other more demanding code requirements.

A6.8 The need for a simple transformation from the set sequence to the prevalent collating convention was recognized, and dictated the placement of some of the "specials" within the set. Specifically, those special symbols, viz., *ampersand* (&), *asterisk* (*), *comma* (,), *hyphen* (-), *period* (.), and *slant* (/), which are most often used as identifiers for ordering information and which normally collate ahead of both the alphabet and the numerals, were not placed in the column containing the numbers, so that the entire numeric column could be rotated via a relatively simple transformation to a position higher than the alphabet. The sequence of the aforementioned "specials" was also established to the extent practical to conform to the prevalent collating convention.

A6.9 The need for a useful 4-bit numeric subset also played a role in the placement of characters. Such a 4-bit subset, including the digits and the symbols *asterisk*, *plus* (+), *comma*, *hyphen*, *period*, and *slant*, can easily be derived from the code.

A6.10 Considerations of other domestic code sets, including the Department of Defense former standard 8-bit data transmission code ("Fieldata"-1961), as well as international requirements, played an important role in deliberations that resulted in the code. The selection and grouping of the symbols *dollar sign* (\$), *percent* (%), *ampersand* (&), *apostrophe* ('), *less than* (<), *equals* (=), and *greater than* (>) facilitate contraction to either a business or scientific 6-bit subset. The position of these symbols, and of the symbols *comma*, *hyphen*, *period*, and *slant*, facilitates achievement of commonly accepted pairing on a keyboard. The historic pairing of *question mark* and *slant* is preserved and the *less than* and *greater than* symbols, which have comparatively low usage, are paired with *period* and *comma* so that in dual-case keyboard devices where it is desired to have *period* and *comma* in both cases, the *less than* and *greater than* symbols are the ones displaced. Provision was made for the accommodation of alphabets containing more than 26 letters and for 6-bit contraction

by the location of low-usage characters in the area following the alphabet. In addition, the requirement for the digits 10 and 11 used in sterling monetary areas was considered in the placement of the *asterisk*, *plus*, *semicolon*, and *colon*, so that the 10 and 11 could be substituted for the *semicolon* and *colon*.

A7. Control Subset Content and Structure

A7.1 The control characters included in the set are those required for the control of terminal devices, input and output devices, format, or communication transmission and switching on a general enough basis to justify inclusion in a standard set.

A7.2 Many control characters may be considered to fall into the following categories:

- (1) Communication Controls
- (2) Format Effectors
- (3) Device Controls
- (4) Information Separators.

To the extent practical controls of each category were grouped in the code table. The structure chosen also facilitates the contraction of the set to a logically related 6-bit set.

A7.3 The information separators (FS, GS, RS, US) identify boundaries of various elements of information, but differ from punctuation in that they are primarily intended to be machine sensible. They were arranged in accordance with an expected hierarchical use, and the lower end of the hierarchy is contiguous in binary order with SP (space) which is sometimes used as a machine-sensible separator. Subject to this hierarchy the exact nature of their use within data is not specified.

A7.4 The character SYN (Synchronous Idle) was located so that its binary pattern in serial transmission was unambiguous as to character framing, and also to optimize certain other aspects of its communication usage.

A7.5 ACK (Acknowledge) and NAK (Negative Acknowledge) were located so as to gain the maximum practical protection against mutation of one into the other by transmission errors.

A7.6 The function "New Line" (NL) was associated with LF (rather than with CR or with a separate character) to provide the most useful combinations of functions through the use of only two character positions, and to allow the use of a common end-of-line format for both printers having separate CR-LF functions and those having a combined (i.e., NL) function. This sequence would be CR-LF, producing the same result on printers of both classes, and would be useful during conversion of a system from one method of operation to the other.

Appendix B Notes on Application

B1. Introduction

B1.1 The standard code was developed to provide for information interchange among information processing systems, communications systems, and associated equipment. In a system consisting of equipment with several local or native codes, maximum flexibility will be achieved if each of the native codes is translated to the standard whenever information interchange is desired.

B1.2 Within any particular equipment, system, or community of activity, it may be necessary to substitute characters. For example, some systems may require special graphic symbols and some devices may require special control codes. (Design efforts on the standard code included consideration of these types of adaptations.) So-called "secular sets" produced by such substitutions, although not conforming to this standard, may nonetheless be consonant with it if substitutions are made in accordance with the guidelines of B2.

B2. Character Substitutions

B2.1 Any character substitution will result in a coded character set which does not conform to this standard.

B2.2 It is recommended that when a character is substituted in the code table for a standard character, the standard character should not be reassigned elsewhere in the table. Such a substitute character, once assigned, should not be subsequently reassigned elsewhere.

B2.3 It is recommended that graphic substitutions be made only in the graphic area and control substitutions only in the control area. Any substitution involving a control should be made only with full cognizance of all possible operational effects.

B2.4 It should be noted that this standard specifies, for each position of the code table, the information represented by the character and not necessarily the precise action taken by the recipient when the character is received. In the case of graphics, considerable variation in the actual shape printed or displayed may be appropriate to different units, systems, or fields of application. In the case of controls, the action performed is dependent upon the use for which the particular system is intended, the application to which it is being put, and a number of conventions established by the user or designer—some system-wide and some unique to a particular unit.

B2.5 Typical examples of diversity in execution not necessarily contrary to this standard are:

(1) A number of graphic symbols, other than that used in the code table, are used for *ampersand* in various type styles; still other symbols may be more appropriate to electronic display devices. The use of such alternate symbols does not in itself constitute deviation from the standard as long as *ampersand* is the concept associated with the character. Note that this does not necessarily restrict the use of such an alternate symbol design to mean "and"; in any type style *ampersand* may, of course, be used with arbitrary meaning.

(2) A card punch in one application may "skip" when the character HT (Horizontal Tabulation: used as skip) is presented to it; in another application the character HT may be recorded in the card without further action.

B3. Related Larger and Smaller Sets

Consideration has been given to the relationship between the standard set and sets of other sizes. A number of straightforward logical transformations are possible which result in a variety of sets related to the standard. None of the transformed sets are recognized by this standard.

B4. International Considerations

This standard conforms to the anticipated recommendations of the International Organization for Standardization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT)* for a 7-bit code. It includes all the character assignments specified by those bodies for international standardization. Their recommendations, however, permit national standardization by the various countries in seven code table positions. Also, the characters in three additional positions have been designated as being replaceable by national characters in only those countries having an extraordinary requirement in this regard.

*An international body which establishes standards and conventions for international telecommunications, especially where the public telegraph and telephone services are governmentally owned and operated. Their recommendations are often embodied in the regulations applying to such services.

The seven national usage positions and their assignments in this standard are shown in the following, as well as the three "supplementary" positions, which are shown in parentheses:

Column	Row	Character (U.S.)
4/0		@
5/11		[
5/12		\
5/13]
(5/14)		~
(6/0)		'
7/11		{
7/12		:
7/13		}
(7/14)		~

In international interchange of information these 10 characters should not be used except where it is determined that there is agreement between sender and recipient.

In addition, in other countries, the number sign (#) (in position 2/3) may be replaced by "£".

B5. Communications Considerations

Certain control characters are designated as "communication controls." They are:

SOH	(Start of Heading)
STX	(Start of Text)
ETX	(End of Text)
EOT	(End of Transmission)
ENQ	(Enquiry)
ACK	(Acknowledge)
DLE	(Data Link Escape)
NAK	(Negative Acknowledge)
SYN	(Synchronous Idle)
ETB	(End of Transmission Block)

These may be used by communication systems for their internal signaling, or for the exchange of information relating to the control of the communication system between that system and its end terminals. Some such systems may impose restrictions on the use of these communication control characters by the end terminals. For example, the use of some of them may be completely prohibited while others may be restricted to use in conformity with the formats and procedures required by the communication system for its operation.

Appendix C Original Criteria

C1. Introduction

C1.1 This Appendix contains the original criteria upon which the design of the code was based. Not all criteria have been entirely satisfied. Some are conflicting, and

the characteristics of the set represent accepted compromises of these divergent criteria.

C1.2 The criteria were drawn from communication, processing, and media recording aspects of information interchange.

C2. Criteria

- C2.1** Every character of the code set shall be represented by the same number of bits (i.e., binary digits).
- C2.2** The standard set shall be so structured as to facilitate derivation of logically related larger or smaller sets.
- C2.3** In a code of n bits, all possible 2^n patterns of ones and zeros will be permitted and considered valid.
- C2.4** The number of bits, n , shall be sufficient to provide for the alphabetic and numeric characters, commonly used punctuation marks, and other special symbols, along with those control characters required for interchange of information.
- C2.5** The numerals 0 through 9 shall be included within a 4-bit subset.
- C2.6** The numerals 0 through 9 shall be so represented that the four low-order bits shall be the binary-coded decimal form of the particular numeral that the code represents. In the selection of the two characters immediately succeeding the numeral 9, consideration shall be given to their replacement by the graphics 10 and 11 to facilitate the adoption of the code in the sterling monetary area.
- C2.7** The interspersion of control characters among the graphic characters shall be avoided. The characters devoted to controls shall be easily separable from those devoted to graphics.
- C2.8** Within the standard set, each character shall stand by itself and not depend on surrounding characters for interpretation.
- C2.9** An entire case of the alphabet (A through Z) shall be included within a 5-bit subset. Consideration shall be given to the need for more than 26 characters in some alphabets.
- C2.10** The letters of each case of the alphabet shall be assigned, in conventional order (A through Z), to successive, increasing binary representations.
- C2.11** Suitable control characters required for communication and information processing shall be included.
- C2.12** Escape functions that provide for departures from the standard set shall be incorporated.
- C2.13** A simple binary comparison shall be sufficient to determine the order within each class of characters. (In this regard, the special graphics, the numerals, and the alphabet are each defined as distinct classes.) Simple binary rules do not necessarily apply between classes when ordering information.
- C2.14** Space (i.e., the space between words) must collate ahead of all other graphics.
- C2.15** Special symbols used in the ordering of information must collate ahead of both the alphabet and the numerals.
- C2.16** Insofar as possible, the special symbols shall be grouped according to their functions; for example, punctuation and mathematical symbols. Further, the set shall be so organized that the simplest possible test shall be adequate to distinguish and identify the basic alphabetic, numeric, and special symbol subsets.
- C2.17** Special symbols shall be placed in the set so as to simplify their generation by typewriters and similar keyboard devices. This criterion means, in effect, that the codes for pairs of characters that normally appear on the same keytops on a typewriter shall differ only in a common single-bit position.
- C2.18** The set shall contain the graphic characters of the principal programming languages.
- C2.19** The codes for all control characters shall contain a common, easily recognizable, bit pattern.
- C2.20** The Null (000...) and Delete (111...) characters shall be provided.

Appendix D

Terminology

This Appendix is intended to clarify the sense in which certain terms are used.

Bit: Contraction of "binary digit."

Bit Pattern: The binary representation of a character.

Character: A member of a coded character set; the binary representation of such a member and its graphic symbol or control function.

Code: A system of discrete representation of a set of symbols and functions.

FILE TRANSFER PROTOCOL

Preceding page blank

File Transfer Protocol
(Aug. 12, 1973)
RFC 542 NIC 17759

Nancy J. Neigus
Bolt Beranek and Newman, Inc.
Cambridge, Mass.

See Also: RFCs 354, 454, 495

File Transfer Protocol for the ARPA Network

Preceding page blank

PREFACE

This document is the result of several months discussion via RFC (relevant numbers are 430, 448, 454, 463, 468, 478, 480), followed by a meeting of the FTP committee at BBN on March 16, followed by further communication among committee members. There are a considerable number of changes for the last "official" version, see RFCs 354, 385, but the gross structure remains the same. The places to look for differences are (1) in the definitions of types and modes, (2) in the specification of the data connection and data sockets, (3) in the command-reply sequences, (4) in the functions dependent on the TELNET protocol (FTP has been altered to correspond to the new TELNET spec). The model has been clarified and enlarged to allow inter-server file transfer, and several new commands have been added to accommodate more specialized (or site-specific) functions. It is my belief that this new specification reflects the views expressed by the committee at the above-mentioned meeting and in subsequent conversations.

The large number of incompatibilities would complicate a phased implementation schedule, such as is in effect for the TELNET protocol. Therefore we have assigned a new socket, decimal 21, as a temporary logger socket for the new version and a change-over date of 1 February 1974. Until that date the old (354, 385) version of FTP will be available on Socket 3 and the new version (attached) should be implemented on Socket 21. On 1 February the new version will shift to Socket 3 and the old disappear from view.

The File Transfer protocol should be considered stable at least until February, though one should feel free to propose further changes via RFC. (Implementation of new commands on an experimental basis is encouraged and should also be reported by RFC.) In addition, members of the FTP committee may be contacted directly about changes. Based on attendance at the March 16 meeting, they are:

Abhay Bhushan MIT-DMCG
Bob Braden UCLA-CCN
Bob Bressler BBN-NET
Bob Clements BBN-TENEX
John Day ILL-ANTS
Peter Deutsch PARC-MAXC
Wayne Hathaway AMES-67
Mike Kudlick SRI-ARC
Alex McKenzie BBN-NET
Bob Merryman UCSD-CC
Nancy Neigus BBN-NET
Mike Padlipsky MIT-Multics
Jim Pepin USC-44
Ken Pogran MIT-Multics
Jon Postel UCLA-NMC

File Transfer Protocol
(Aug. 12, 1973)
RFC 542 NIC 17759

Milton Reese FNWC
Brad Reussow HARV-10
Marc Seriff MIT-DMCG
Ed Taft HARV-10
Bob Thomas BBN-TENEX
Ric Werme CMU-10
Jim White SRI-ARC

I would especially like to thank Bob Braden, Ken Pogran, Wayne Hathaway, Jon Postel, Ed Taft and Alex McKenzie for their help in preparing this document.

NJN/jm

FILE TRANSFER PROTOCOL

INTRODUCTION

The File Transfer Protocol (FTP) is a protocol for file transfer between Hosts (including Terminal Interface Message Processors (TIPs)) on the ARPA Computer Network (ARPANET). The primary function of FTP is to transfer files efficiently and reliably among Hosts and to allow the convenient use of remote file storage capabilities.

The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among Hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs.

The attempt in this specification is to satisfy the diverse needs of users of maxi-Hosts, mini-Hosts, TIPs, and the Datacomputer, with a simple, and easily implemented protocol design.

This paper assumes knowledge of the following protocols described in NIC #7104:

The Host-Host Protocol

The Initial Connection Protocol

The TELNET Protocol

DISCUSSION

In this section, the terminology and the FTP model are discussed. The terms defined in this section are only those that have special significance in FTP.

TERMINOLOGY

ASCII

The USASCII character set as defined in NIC #7104. In FTP, ASCII characters are defined to be the lower half of an eight-bit code set (i.e., the most significant bit is zero).

access controls

Access controls define users' access privileges to the use of a system, and to the files in that system. Access controls are necessary to prevent unauthorized or accidental use of files.

It is the prerogative of a server-FTP process to provide access controls.

byte size

The byte size specified for the transfer of data. The data connection is opened with this byte size. The data connection byte size is not necessarily the byte size in which data is to be stored in a system, nor the logical byte size for interpretation of the structure of the data.

data connection

A simplex connection over which data is transferred, in a specified byte size, mode and type. The data transferred may be a part of a file, an entire file or a number of files. The path may be between a server-DTP and a user-DTP, or between two server-DTPs.

data socket

The passive data transfer process "listens" on the data socket for an RFC from the active transfer process (server) in order to open the data connection. The server has fixed data sockets; the passive process may or may not.

EOF

The end-of-file condition that defines the end of a file being transferred.

EOR

The end-of-record condition that defines the end of a record being transferred.

error recovery

A procedure that allows a user to recover from certain errors such as failure of either Host system or transfer process. In FTP, error recovery may involve restarting a file transfer at a given checkpoint.

FTP commands

A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.

file

An ordered set of computer data (including programs), of arbitrary length, uniquely identified by a pathname.

mode

The mode in which data is to be transferred via the data connection. The mode defines the data format during transfer including EOR and EOF. The transfer modes defined in FTP are described in the Section on Transmission Modes.

NVT

The Network Virtual Terminal as defined in the ARPANET TELNET Protocol.

NVFS

The Network Virtual File System. A concept which defines a standard network file system with standard commands and pathname conventions. FTP only partially embraces the NVFS concept at this time.

pathname

Pathname is defined to be the character string which must be input to a file system by a user in order to identify a file. Pathname normally contains device and/or directory names, and file name specification. FTP does not yet specify a standard pathname convention. Each user must follow the file naming conventions of the file systems he wishes to use.

record

A sequential file may be structured as a number of contiguous parts called records. Record structures are supported by FTP but a file need not have record structure.

reply

A reply is an acknowledgment (positive or negative) sent from server to user via the TELNET connections in response to FTP commands. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

server-DTP

The data transfer process, in its normal "active" state, establishes the data connection by RFC to the "listening" data socket, sets up parameters for transfer and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for, rather than initiate, an RFC on the data socket.

server-FTP process

A process or set of processes which perform the function of file transfer in cooperation with a user-FTP process and, possibly, another server. The functions consist of a protocol interpreter (PI) and a data transfer process (DTP).

server-PI

The protocol interpreter "listens" on Socket 3 for an ICP from a user-PI and establishes a TELNET communication connection. It receives standard FTP commands from the user-PI, sends replies, and governs the server-DTP.

TELNET connections

The full-duplex communication path between a user-PI and a server-PI. The TELNET connections are established via the standard ARPANET Initial Connection Protocol (ICP).

type

The data representation type used for data transfer and storage. Type implies certain transformations between the time of data storage and data transfer. The representation types defined in FTP are described in the Section on Establishing Data Connections.

user

A human being or a process on behalf of a human being wishing to obtain file transfer service. The human user may interact directly with a server-FTP process, but use of a user-FTP process is preferred since the protocol design is weighted towards automata.

user-DTP

The data transfer process "listens" on the data socket for an RFC from a server-FTP process. If two servers are transferring data between them, the user-DTP is inactive.

user-FTP process

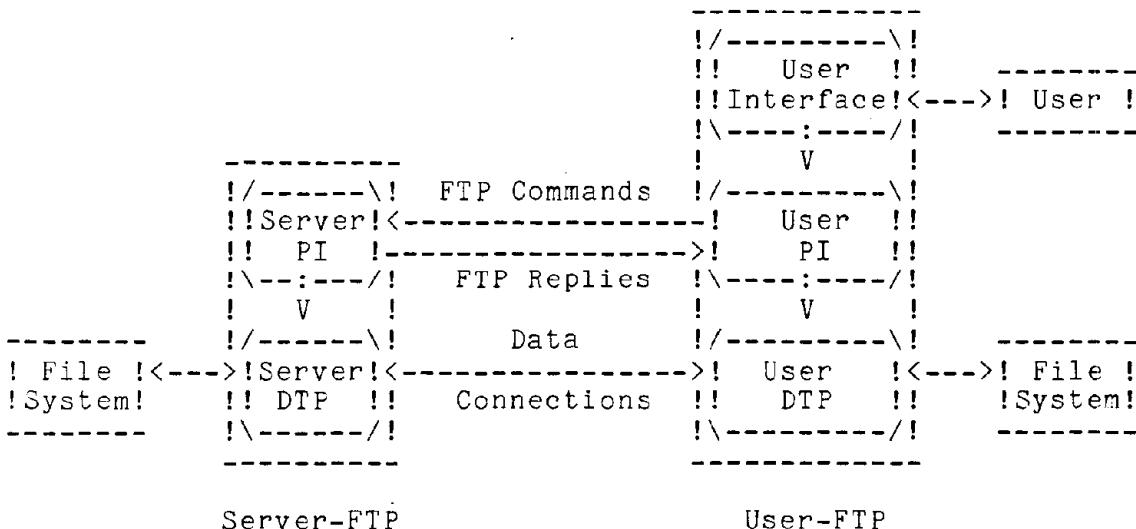
A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.

user-PI

The protocol interpreter initiates the ICP to the server-FTP process, initiates FTP commands, and governs the user-DTP if that process is part of the file transfer.

THE FTP MODEL

With the above definitions in mind, the following model (shown in Figure 1) may be diagrammed for an FTP service.



NOTES: 1. The data connection may be in either direction.
2. The data connection need not exist all of the time.

Figure 1 Model for FTP Use

In the model described in Figure 1, the user-protocol interpreter initiates the TELNET connections. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the TELNET connections. (The user may establish a direct TELNET connection to the server-FTP, from a TIP terminal for example, and generate standard FTP commands himself, by-passing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the TELNET connections in response to the commands.

The FTP commands specify the parameters for the data connection (data socket, byte size, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data socket, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data socket need not be in the same Host that initiates the FTP commands via the TELNET connections, but the user or his user-FTP process must ensure a "listen" on the specified data socket. It should also be noted that two data connections, one for send and the other for receive, may exist simultaneously.

In another situation a user might wish to transfer files between two Hosts, neither of which is his local Host. He sets up TELNET connections to the two servers and then arranges for a data connection between them. In this manner control information is passed to the user-PI but data is transferred between the server data transfer processes. Following is a model of this server-server interaction.

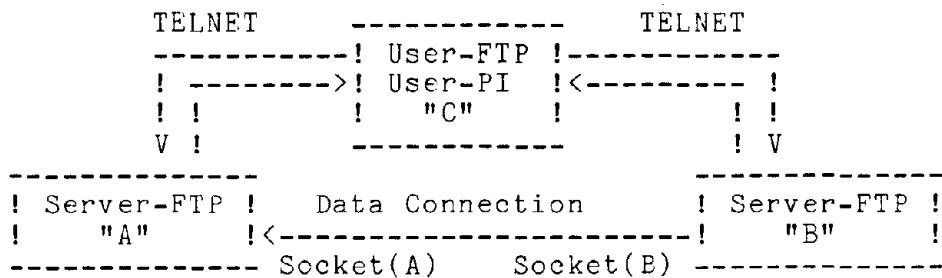


Figure 2

The protocol requires that the TELNET connections be open while data transfer is in progress. It is the responsibility of the user to request the closing of the TELNET connections when finished using the FTP service, while it is the server who takes the action. The server may abort data transfer if the TELNET connections are closed without command.

DATA TRANSFER FUNCTIONS

Files are transferred only via the data connection(s). The TELNET connection is used for the transfer of commands, which describe the functions to be performed, and the replies to these commands (see the Section on FTP Replies). Several commands are concerned with the transfer of data between Hosts. These data transfer commands include the BYTE, MODE, and SOCKet commands which specify how the bits of the data are to be transmitted, and the STRUcture and TYPE commands, which are used to define the way in which the data are to be represented. The transmission and representation are basically independent but "Stream" transmission mode is dependent on the file structure attribute and if "Compressed" transmission mode is used the nature of the filler byte depends on the representation type.

DATA REPRESENTATION AND STORAGE

Data is transferred from a storage device in the sending Host to a storage device in the receiving Host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has different data storage representations in different systems. PDP-10's generally store NVT-ASCII as five 7-bit ASCII characters, left-justified in a 36-bit word. 360's store NVT-ASCII as 8-bit EBCDIC codes. Multics stores NVT-ASCII as four 9-bit characters in a 36-bit word. It may be desirable to convert characters into the standard NVT-ASCII representation when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

A different problem in representation arises when transmitting binary data (not character codes) between Host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example, when transmitting 32-bit bytes from a 32-bit word-length system to a 36-bit word-length system, it may be desirable (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user should have the option of specifying data representation and transformation functions. It should be noted that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be

performed by the user directly or via the use of the Data Reconfiguration Service (DRS, RFC #138, NIC #6715). Additional representation types may be defined later if there is a demonstrable need.

Data representations are handled in FTP by a user specifying a representation type. This type may implicitly (as in ASCII or EBCDIC) or explicitly (as in Local byte) define a byte size for interpretation which is referred to as the "logical byte size." This has nothing to do with the byte size used for transmission over the data connection(s) (called the "transfer byte size") and the two should not be confused. For example, NVT-ASCII has a logical byte size of 8 bits but an ASCII file might be transferred using a transfer byte size of 32. If the type is Local byte, then the TYPE command has an obligatory second parameter specifying the logical byte size.

The types ASCII and EBCDIC also take a second (optional) parameter; this is to indicate what kind of vertical format control, if any, is associated with a file. The following data representation types are defined in FTP:

ASCII Format

This is the default type and must be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both Hosts would find the EBCDIC type more convenient.

The sender converts the data from his internal character representation to the standard 8-bit NVT-ASCII representation (see the TELNET specification). The receiver will convert the data from the standard form to his own internal form.

In accordance with the NVT standard, the <CRLF> sequence should be used, where necessary, to denote the end of a line of text. (See the discussion of file structure at the end of the Section on Data Representation and Storage).

Using the standard NVT-ASCII representation means that data must be interpreted as 8-bit bytes. If the BYTE command (see the Section on Transfer Parameter Commands) specifies a transfer byte size different from 8 bits, the 8-bit ASCII characters should be packed contiguously without regard for transfer byte boundaries.

The Format parameter for ASCII and EBCDIC types is discussed below.

EBCDIC Format

This type is intended for efficient transfer between Hosts which use EBCDIC for their internal character representation.

For transmission the data are represented as 8-bit EBCDIC characters. The character code is the only difference between the functional specifications of EBCDIC and ASCII types.

End-of-line (as opposed to end-of-record--see the discussion of structure) will probably be rarely used with EBCDIC type for purposes of denoting structure, but where it is necessary the <NL> character should be used.

A character file may be transferred to a Host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving Host must know how the vertical format control is represented. In the second case, it must be possible to store a file at a Host and then retrieve it later in exactly the same form. Finally, it ought to be possible to move a file from one Host to another and process the file at the second Host without undue trouble. A single ASCII or EBCDIC format does not satisfy all these conditions and so these types have a second parameter specifying one of the following three formats:

Non-print

This is the default format to be used if the second (format) parameter is omitted. Non-print format must be accepted by all FTP implementations.

The file need contain no vertical format information. If it is passed to a printer process, this process may assume standard values for spacing and margins.

Normally, this format will be used with files destined for processing or just storage.

TELNET Format Controls

The file contains ASCII/EBCDIC vertical format controls (i.e., <CR>, <LF>, <NL>, <VT>, <FF>) which the printer process will interpret appropriately. <CRLF>, in exactly this sequence, also denotes end-of-line.

Carriage Control (ASA)

The file contains ASA (FORTRAN) vertical format control characters. (See NWG/RFC #189 Appendix C and Communications cf

the ACM, Vol. 7, No. 10, 606 (Oct. 1964)). In a line or a record, formatted according to the ASA Standard, the first character is not to be printed. Instead it should be used to determine the vertical movement of the paper which should take place before the rest of the record is printed. The ASA Standard specifies the following control characters:

Character	Vertical Spacing
blank	Move paper up one line
0	Move paper up two lines
1	Move paper to top of next page
+	No movement, i.e., overprint

Clearly there must be some way for a printer process to distinguish the end of the structural entity. If a file has record structure (see below) this is no problem; records will be explicitly marked during transfer and storage. If the file has no record structure, the <CRLF> end-of-line sequence is used to separate printing lines, but these format effectors are overridden by the ASA controls.

Image

The data are sent as contiguous bits which, for transfer, are packed into transfer bytes of the size specified in the BYTE command. The receiving site must store the data as contiguous bits. The structure of the storage system might necessitate the padding of the file (or of each record, for a record-structured file) to some convenient boundary (byte, word or block). This padding, which must be all zeroes, may occur only at the end of the file (or at the end of each record) and there must be a way of identifying the padding bits so that they may be stripped off if the file is retrieved. The padding transformation should be well publicized to enable a user to process a file at the storage site.

Image type is intended for the efficient storage and retrieval of files and for the transfer of binary data. It is recommended that this type be accepted by all FTP implementations.

Local byte Byte size

The data is transferred in logical bytes of the size specified by the obligatory second parameter, Byte size. The value of Byte size must be a decimal integer; there is no default value. The logical byte size is not necessarily the same as the transfer byte size. If there is a difference in byte sizes,

then the logical bytes should be packed contiguously, disregarding transfer byte boundaries and with any necessary padding at the end.

When the data reaches the receiving Host it will be transformed in a manner dependent on the logical byte size and the particular Host. This transformation must be invertible (that is an identical file can be retrieved if the same parameters are used) and should be well publicized by the FTP implementors.

This type is intended for the transfer of structured data. For example, a user sending 36-bit floating-point numbers to a Host with a 32-bit word could send his data as Local byte with a logical byte size of 36. The receiving Host would then be expected to store the logical bytes so that they could be easily manipulated; in this example putting the 36-bit logical bytes into 64-bit double words should suffice.

A note of caution about parameters: a file must be stored and retrieved with the same parameters if the retrieved version is to be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

In addition to different representation types, FTP allows the structure of a file to be specified. Currently two file structures are recognized in FTP: file-structure, where there is no internal structure, and record-structure, where the file is made up of records. File-structure is the default, to be assumed if the STRUCTure command has not been used but both structures must be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations. The structure of a file will affect both the transfer mode of a file (see the Section on Transmission Modes) and the interpretation and storage of the file.

The "natural" structure of a file will depend on which Host stores the file. A source-code file will usually be stored on an IBM 360 in fixed length records but on a PDP-10 as a stream of characters partitioned into lines, for example by <CRLF>. If the transfer of files between such disparate sites is to be useful, there must be some way for one site to recognize the other's assumptions about the file.

With some sites being naturally file-oriented and others naturally record-oriented there may be problems if a file with one structure is sent to a Host oriented to the other. If a text file is sent with record-structure to a Host which is file oriented, then that Host should apply an internal transformation to the file based on the

record structure. Obviously this transformation should be useful but it must also be invertible so that an identical file may be retrieved using record structure.

In the case of a file being sent with file-structure to a record-oriented Host, there exists the question of what criteria the Host should use to divide the file into records which can be processed locally. If this division is necessary the FTP implementation should use the end-of-line sequence, <CRLF> for ASCII, or <NL> for EBCDIC text files, as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

ESTABLISHING DATA CONNECTIONS

The mechanics of transferring data consists of setting up the data connection to the appropriate sockets and choosing the parameters for transfer--byte size and mode. Both the user and the server-DTPs have default data sockets; these are the two sockets (for send and receive) immediately following the standard ICP TELNET socket ,i.e., (U+4) and (U+5) for the user-process and (S+2), (S+3) for the server. The use of default sockets will ensure the security of the data transfer, without requiring the socket information to be explicitly exchanged.

The byte size for the data connection is specified by the BYTE command, or, if left unspecified, defaults to 8-bit bytes. This byte size is relevant only for the actual transfer of the data; it has no bearing on representation of the data within a Host's file system. The protocol does not require servers to accept all possible byte sizes. Since the use of various byte sizes is intended for efficiency of transfer, servers may implement only those sizes for which their data transfer is efficient including the default byte size of 8 bits.

The passive data transfer process (this may be a user-DTP or a second server-DTP) shall "listen" on the data socket prior to sending a transfer request command. The FTP request command determines the direction of the data transfer and thus which data socket (odd or even) is to be used in establishing the connection. The server, upon receiving the transfer request, will initiate the data connection by RFC to the appropriate socket using the specified (or default) byte size. When the connection is opened, the data transfer begins between DTP's, and the server-PI sends a confirming reply to the user-PI.

It is possible for the user to specify an alternate data socket by use of the SOCK command. He might want a file dumped on a TIP line printer or retrieved from a third party Host. In the latter case the user-PI sets up TELNET connections with both server-PI's and sends

each a SOCK command indicating the fixed data sockets of the other. One server is then told (by an FTP command) to "listen" for an RFC which the other will initiate and finally both are sent the appropriate transfer commands. The exact sequence of commands and replies sent between the user-controller and the servers is defined in the Section on FTP Replies.

In general it is the server's responsibility to maintain the data connection--to initiate the RFC's and the closes. The exception to this is when the user-DTP is sending the data in a transfer mode that requires the connection to be closed to indicate EOF. The server MUST close the data connection under the following conditions:

1. The server has completed sending data in a transfer mode that requires a close to indicate EOF.
2. The server receives an ABORT command from the user.
3. The socket or byte size specification is changed by a command from the user.
4. The TELNET connections are closed legally or otherwise.
5. An irrecoverable error condition occurs.

Otherwise the close is a server option, the exercise of which he must indicate to the user-process by an appropriate reply.

TRANSMISSION MODES

The next consideration in transferring data is choosing the appropriate transmission mode. There are three modes: one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode the representation type determines the filler byte.

All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection. For files with record structure, all the end-of-record markers (EUR) are explicit, including the final one.

Note: In the rest of this section, byte means "transfer byte" except where explicitly stated otherwise.

The following transmission modes are defined in FTP:

Stream

The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed, in which case the transfer byte size must be at least 3 bits!

In a record structured file EOR and EOF will each be indicated by a two-byte control code of whatever byte size is used for the transfer. The first byte of the control code will be all ones, the escape character. The second byte will have the low order bit on and zeroes elsewhere for EOR and the second low order bit on for EOF; that is, the byte will have value 1 for EOR and value 2 for EOF. EOR and EOF may be indicated together on the last byte transmitted by turning both low order bits on, i.e., the value 3. If a byte of all ones was intended to be sent as data, it should be repeated in the second byte of the control code.

If the file does not have record structure, the EOF is indicated by the sending Host closing the data connection and all bytes are data bytes.

For the purpose of standardized transfer, the sending Host will translate his internal end of line or end of record denotation into the representation prescribed by the transfer mode and file structure, and the receiving Host will perform the inverse translation to his internal denotation. An IBM 360 record count field may not be recognized at another Host, so the end of record information may be transferred as a two byte control code in Stream mode or as a flagged bit in a Block or Compressed mode descriptor. End of line in an ASCII or EBCDIC file with no record structure should be indicated by <CRLF> or <NL>, respectively. Since these transformations imply extra work for some systems, identical systems transferring non-record structured text files might wish to use a binary representation and stream mode for the transfer.

Block

The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF) last block in the record (EOR), restart marker (see the Section on Error Recovery and Restart) or suspect data (i.e., the data

being transferred is suspected of errors and is not reliable). This last code is NOT intended for error control within FTP. It is motivated by the desire of sites exchanging certain types of data (e.g., seismic or weather data) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain portions are suspect). Record structures are allowed in this mode, and any representation type may be used. There is no restriction on the transfer byte size.

The header consists of the smallest integral number of bytes whose length is greater than or equal to 24 bits. Only the LEAST significant 24 bits (right-justified) of header shall have information; the remaining most significant bits are "don't care" bits. Of the 24 bits of header information, the 16 low order bits shall represent byte count, and the 8 high order bits shall represent descriptor codes as shown below.

Integral number of bytes greater than or equal to 24 bits

!	Don't care	!	Descriptor	!	Byte Count	!
!	0 to 231 bits	!	8 bits	!	16 bits	!

The descriptor codes are indicated by bit flags in the descriptor byte. Four codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

Code	Meaning
128	End of data block is EOR
64	End of data block is EOF
32	Suspected errors in data block
16	Data block is a restart marker

With this encoding more than one descriptor coded condition may exist for a particular block. As many bits as necessary may be flagged.

The restart marker is embedded in the data stream as an integral number of 8-bit bytes representing printable characters in the language being used over the TELNET connection (e.g., default--NVT-ASCII). These marker bytes are right-justified in the smallest integral number of transfer bytes greater than or equal to 8 bits. For example, if the

byte size is 7 bits, the restart marker byte would be one byte right-justified per two 7-bit bytes as shown below:

Two 7-bit bytes

```
-----  
!      ! Marker Char !  
!      ! 8 bits   !  
-----
```

If the transfer byte size is 16 or more bits, the maximum possible number of complete marker bytes should be packed, right-justified, into each transfer byte. The restart marker should begin in the first marker byte. If there are any unused marker bytes, these should be filled with the character <SP> (Space, in the appropriate language). <SP> must not be used WITHIN a restart marker. For example, to transmit a six-character marker with a 36-bit transfer byte size, the following three 36-bit bytes would be sent:

```
-----  
! Don't care !Descriptor! Byte count = 2 !  
! 12 bits   ! code = 16 !  
-----
```

```
-----  
!      ! Marker ! Marker ! Marker ! Marker !  
!      ! 8 bits ! 8 bits ! 8 bits ! 8 bits !  
-----
```

```
-----  
!      ! Marker ! Marker ! Space  ! Space  !  
!      ! 8 bits ! 8 bits ! 8 bits ! 8 bits !  
-----
```

Compressed

The file is transmitted as series of bytes of the size specified by the BYTE command. There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence. If the byte size is B bits and $n > 0$ bytes of regular data are sent, these n bytes are preceded by a byte with the left-most bit set to 0 and the right-most B-1 bits containing the number n.

1 B-1 B B
-----|-----|-----
Byte string: !0! n ! !d(1)!...!d(n)!
-----|-----|-----
 ↑ ↑
 !---n bytes---!
 of data

String of n data bytes d(1),..., d(n)
Count n must be positive

To compress a string of n replications of the data byte d, the following 2 bytes are sent:

2 B-2 B
-----|-----|-----
Replicated Byte: ! 1 0 ! n ! ! d !
-----|-----|-----

A string of n filler bytes can be compressed into a single byte, where the filler byte varies with the representation type. If the type is ASCII or EBCDIC the filler byte is <SP> (Space, ASCII code 32., EBCDIC code 64). If the transfer byte size is not 8, the expanded byte string should be filled with 8-bit <SP> characters in the manner described in the definition of ASCII representation type (see the Section on Data Representation and Storage). If the type is Image or Local byte the filler is a zero byte.

2 B-2
-----|-----
Filler String: ! 1 1 ! n !
-----|-----

The escape sequence is a double byte, the first of which is the escape byte (all zeroes) and the second of which contains descriptor codes as defined in Block mode. This implies that the byte size must be at least 8 bits, which is not much of a restriction for efficiency in this mode. The descriptor codes have the same meaning as in Block mode and apply to the succeeding string of bytes.

Compressed mode is useful for obtaining increased bandwidth on very large network transmissions at a little extra CPU cost. It is most efficient when the byte size chosen is that of the word size of the transmitting Host, and can be most effectively used to reduce the size of printer files such as those generated by RJE Hosts.

ERROR RECOVERY AND RESTART

There is no provision for detecting bits lost or scrambled in data transfer. This issue is perhaps handled best at the NCP level where it benefits most users. However, a restart procedure is provided to protect users from gross system failures (including failures of a Host, an FTP-process, or the IMP subnet).

The restart procedure is defined only for the block and compressed modes of data transfer. It requires the sender of data to insert a special marker code in the data stream with some marker information. The marker information has meaning only to the sender, but must consist of printable characters in the default or negotiated language of the TELNET connection. The marker could represent a bit-count, a record-count, or any other information by which a system may identify a data checkpoint. The receiver of data, if it implements the restart procedure, would then mark the corresponding position of this marker in the receiving system, and return this information to the user.

In the event of a system failure, the user can restart the data transfer by identifying the marker point with the FTP restart procedure. The following example illustrates the use of the restart procedure.

The sender of the data inserts an appropriate marker block in the data stream at a convenient point. The receiving Host marks the corresponding data point in its file system and conveys the last known sender and receiver marker information to the user, either directly or over the TELNET connection in a 251 reply (depending on who is the sender). In the event of a system failure, the user or controller process restarts the server at the last server marker by sending a restart command with server's marker code as its argument. The restart command is transmitted over the TELNET connection and is immediately followed by the command (such as RETR, STOR or LIST) which was being executed when the system failure occurred.

FILE TRANSFER FUNCTIONS

The communication channel from the user-PI to the server-PI is established by ICP from the user to a standard server socket. The user protocol interpreter is responsible for sending FTP commands and interpreting the replies received; the server-PI interprets commands, sends replies and directs its DTP to set up the data connection and transfer the data. If the second party to the data transfer (the passive transfer process) is the user-DTP then it is governed through the internal protocol of the user-FTP Host; if it is a second server-DTP then it is governed by its PI on command from the user-PI.

FTP COMMANDS

The File Transfer Protocol follows the specifications of the TELNET protocol for all communications over the TELNET connection - see NIC #7104. Since, in the future, the language used for TELNET communication may be a negotiated option, all references in the next two sections will be to the "TELNET language" and the corresponding "TELNET end of line code". Currently one may take these to mean NVT-ASCII and <CRLF>. No other specifications of the TELNET protocol will be cited.

FTP commands are "TELNET strings" terminated by the "TELNET end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and TELNET-EOL otherwise. The command codes and the semantics of commands are described in this section; the detailed syntax of commands is specified in the Section on Commands, the reply sequences are discussed in the Section on Sequencing of Commands and Replies, and scenarios illustrating the use of commands are provided in the Section on Typical FTP Scenarios.

FTP commands may be partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. Certain commands (such as ABOR, STAT, BYE) may be sent over the TELNET connections while a data transfer is in progress. Some servers may not be able to monitor the TELNET and data connections simultaneously, in which case some special action will be necessary to get the server's attention. The exact form of the "special action" is related to decisions currently under review by the TELNET committee; but the following ordered format is tentatively recommended:

1. User system inserts the TELNET "Interrupt Process" (IP) signal in the TELNET stream.
2. User system sends the TELNET "Synch" signal
3. User system inserts the command (e.g., ABOR) in the TELNET stream.
4. Server PI,, after receiving "IP", scans the TELNET stream for EXACTLY ONE FTP command.

(For other servers this may not be necessary but the actions listed above should have no unusual effect.)

ACCESS CONTROL COMMANDS

The following commands specify access control identifiers (command codes are shown in parentheses).

USER NAME (USER)

The argument field is a TELNET string identifying the user. The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the TELNET connections are made (some servers may require this). Additional identification information in the form of a password and/or an account command may also be required by some servers. Servers may allow a new USER command to be entered at any point in order to change the access control and/or accounting information. This has the effect of flushing any user, password, and account information already supplied and beginning the login sequence again. All transfer parameters are unchanged and any file transfer in progress is completed under the old account.

PASSWORD (PASS)

The argument field is a TELNET string identifying the user's password. This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress typeout. It appears that the server has no foolproof way to achieve this. It is therefore the responsibility of the user-FTP process to hide the sensitive password information.

ACCOUNT (ACCT)

The argument field is a TELNET string identifying the user's account. The command is not necessarily related to the USER command, as some sites may require an account for login and others only for specific access, such as storing files. In the latter case the command may arrive at any time. There are two reply codes to differentiate these cases for the automaton: when account information is required for login, the response to a successful PASSword command is reply code 331; then if a command other than ACCOUNT is sent, the server may remember it and return a 331 reply, prepared to act on the command after the account information is received; or he may flush the command and return a 433 reply asking for the account. On the other hand, if account information is NOT required for login, the reply to a successful PASSword command is 230; and if the

information is needed for a command issued later in the dialogue, the server should return a 331 or 433 reply depending on whether he stores (pending receipt of the ACCOUNT command) or discards the command, respectively.

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the TELNET connection is left open. This is identical to the state in which a user finds himself immediately after the ICP is completed and the TELNET connections are opened. A USER command may be expected to follow.

LOGOUT (BYE)

This command terminates a USER and if file transfer is not in progress, the server closes the TELNET connection. If file transfer is in progress, the connection will remain open for result response and the server will then close it. If the user-process is transferring files for several USERS but does not wish to close and then reopen connections for each, then the REIN command should be used instead of BYE.

An unexpected close on the TELNET connection will cause the server to take the effective action of an abort (ABOR) and a logout (BYE).

TRANSFER PARAMETER COMMANDS

All data transfer parameters have default values, and the commands specifying data transfer parameters are required only if the default parameter values are to be changed. The default value is the last specified value, or if no value has been specified, the standard default value as stated here. This implies that the server must "remember" the applicable default values. The commands may be in any order except that they must precede the FTP service request. The following commands specify data transfer parameters.

BYTE SIZE (BYTE)

The argument is a decimal integer (1 through 255) specifying the byte size for the data connection. The default byte size is 8 bits. A server may reject certain byte sizes that he has not implemented.

DATA SOCKET (SOCK)

The argument is a HOST-SOCKET specification for the data socket to be used in data connection. There may be two data sockets, one for transfer from the "active" DTP to the "passive" DTP and one for "passive" to "active". An odd socket number defines a send socket and an even socket number defines a receive socket. The default HOST is the user Host to which TELNET connections are made. The default data sockets are (U+4) and (U+5) where U is the socket number used in the TELNET ICP and the TELNET connections are on sockets (U+2) and (U+3). The server has fixed data sockets (S+2) and (S+3) as well, and under normal circumstances this command and its reply are not needed.

PASSIVE (PASV)

This command requests the server-DTP to "listen" on both of his data sockets and to wait for an RFC to arrive for one socket rather than initiate one upon receipt of a transfer command. It is assumed the server has already received a SOCK command to indicate the foreign socket from which the RFC will arrive to ensure the security of the transfer.

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single TELNET character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32.). The following codes are assigned for type:

A - ASCII	\	/	N - Non-print
	!	!	! -><- ! T - TELNET format effectors
E - EBCDIC	!	!	C - Carriage Control (ASA)
	/	\	
I - Image			
L # - Local byte			Bytesize

The default representation type is ASCII Non-print. If the Format parameter is changed, and later just the first argument is changed, Format then returns to the Non-print default.

FILE STRUCTURE (STRU)

The argument is a single TELNET character code specifying file structure described in the Section on Data Representation and Storage. The following codes are assigned for structure:

- F - File (no record structure)
- R - Record structure

The default structure is File (i.e., no records).

TRANSFER MODE (MODE)

The argument is a single TELNET character code specifying the data transfer modes described in the Section on Transmission Modes. The following codes are assigned for transfer modes:

- S - Stream
- B - Block
- C - Compressed

The default transfer mode is Stream.

FTP SERVICE COMMANDS

The FTP service commands define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the TELNET connection. The suggested default handling is to use the last specified device, directory or file name, or the standard default defined for local users. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command. The data, when transferred in response to FTP service commands, shall always be sent over the data connection, except for certain informative replies. The following commands specify FTP service requests:

RETRIEVE (RETR)

This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server- or user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

STORE (STOR)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

APPEND (with create) (APPE)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise the file specified in the pathname shall be created at the server site.

ALLOCATE (ALLO)

This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. The argument shall be a decimal integer representing the number of bytes (using the logical byte size) of storage to be reserved for the file. For files sent with record structure a maximum record size (in logical bytes) might also be necessary; this is indicated by a decimal integer in a second argument field of the command. This second argument is optional, but when present should be separated from the first by the three TELNET characters <SP> R <SP>. This command shall be followed by a STOR or APPEND command. The ALLO command should be treated as a NOOP (no operation) by those servers which do not require that the maximum size of the file be declared beforehand, and those servers interested in only the maximum record size should accept a dummy value in the first argument and ignore it.

RESTART (REST)

The argument field represents the server marker at which file transfer is to be restarted. This command does not cause file transfer but "spaces" over the file to the specified data checkpoint. This command shall be immediately followed by the appropriate FTP service command which shall cause file transfer to resume.

RENAME FROM (RNFR)

This command specifies the file which is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.

RENAME TO (RNTO)

This command specifies the new pathname of the file specified in the immediately preceding "rename from" command. Together the two commands cause a file to be renamed.

ABORT (ABOR)

This command indicates to the server to abort the previous FTP service command and any associated transfer of data. The abort command may require "special action", as discussed in the Section on FTP Commands, to force recognition by the server. No action is to be taken if the previous command has been completed (including data transfer). The TELNET connections are not to be closed by the server, but the data connection must be closed. An appropriate reply should be sent by the server in all cases.

DELETE (DELE)

This command causes the file specified in the pathname to be deleted at the server site. If an extra level of protection is desired (such as the query, "DO you really wish to delete?"), it should be provided by the user-FTP process.

LIST (LIST)

This command causes a list to be sent from the server to the passive DTP. If the pathname specifies a directory, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null argument implies the user's current working or default directory. The data transfer is over the data connection in type ASCII or type EBCDIC. (The user must ensure that the TYPE is appropriately ASCII or EBCDIC).

NAME-LIST (NLST)

This command causes a directory listing to be sent from server to user site. The pathname should specify a directory or other system-specific file group descriptor; a null argument implies the current directory. The server will return a stream of

names of files and no other information. The data will be transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by <CRLF> or <NL>. (Again the user must ensure that the TYPE is correct.)

SITE PARAMETERS (SITE)

This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

STATUS (STAT)

This command shall cause a status response to be sent over the TELNET connection in the form of a reply. The command may be sent during a file transfer (along with the TELNET IP and Synch signals--see the Section on FTP Commands) in which case the server will respond with the status of the operation in progress, or it may be sent between file transfers. In the latter case the command may have an argument field. If the argument is a pathname, the command is analogous to the "list" command except that data shall be transferred over the TELNET connection. If a partial pathname is given, the server may respond with a list of file names or attributes associated with that specification. If no argument is given, the server should return general status information about the server FTP process. This should include current values of all transfer parameters and the status of connections.

HELP (HELP)

This command shall cause the server to send helpful information regarding its implementation status over the TELNET connection to the user. The command may take an argument (e.g., any command name) and return more specific information as a response. The reply is type 0xx, general system status. It is suggested that HELP be allowed before entering a USER command. The server may use this reply to specify site-dependent parameters, e.g., in response to HELP SITE.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the server send a 200 reply.

MISCELLANEOUS COMMANDS

There are several functions that utilize the services of file transfer but go beyond it in scope. These are the Mail and Remote Job Entry functions. It is suggested that these become auxiliary protocols that can assume recognition of file transfer commands on the part of the server, i.e., they may depend on the core of FTP commands. The command sets specific to Mail and RJE will be given in separate documents.

Commands that are closely related to file transfer but not proven essential to the protocol may be implemented by servers on an experimental basis. The command name should begin with an X and may be listed in the HELP command. The official command set is expandable from these experiments; all experimental commands or proposals for expanding the official command set should be announced via RFC. An example of a current experimental command is:

Change Working Directory (XCWD)

This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

FTP REPLIES

The server sends FTP replies over the TELNET connection in response to user FTP commands. The FTP replies constitute the acknowledgment or completion code (including errors). The FTP-server replies are formatted for human or program interpretation. Single line replies consist of a leading three-digit numeric code followed by a space, followed by a one-line text explanation of the code. For replies that contain several lines of text, the first line will have a leading three-digit numeric code followed immediately by the character "-" (Hyphen, ASCII code 45), and possibly some text. All succeeding continuation lines except the last are constrained NOT to begin with three digits; the last line must repeat the numeric code of the first line and be followed immediately by a space. For example:

```
100-First Line
Continuation Line
Another Line
100 Last Line
```

It is possible to nest (but not overlap) a reply within a multi-line

reply. The same format for matched number-coded first and last lines holds.

The numeric codes are assigned by groups and for ease of interpretation by programs in a manner consistent with other protocols such as the RJE protocol. The three digits of the code are to be interpreted as follows:

1. The first digit specifies type of response as indicated below:

0xx These replies are purely informative and constitute neither a positive nor a negative acknowledgment.

1xx Informative replies to status inquiries. These constitute a positive acknowledgment to the status command.

2xx Positive acknowledgment of previous command or other successful action.

3xx Incomplete information. Activity cannot proceed without further specification and input.

4xx Unsuccessful reply. The request is correctly specified but the server is unsuccessful in correctly fulfilling it.

5xx Incorrect or illegal command. The command or its parameters were invalid or incomplete from a syntactic viewpoint, or the command is inconsistent with a previous command. The command in question has been completely ignored.

6xx-9xx Reserved for future expansion.

2. The second digit specifies the general category to which the response refers:

x00-x29 General purpose replies, not assignable to other categories.

x3x Primary access. Informative replies to the "log-on" attempt.

x4x Secondary access. The primary server is commenting on its ability to access a secondary service.

x5x FTP results.

x6x RJE results.

x7x Mail Portocol results.

x8x-x9x Reserved for future expansion.

3. The final digit specifies a particular message type. Since the code is designed for an automaton process to interpret, it is not necessary for every variation of a reply to have a unique number. Only the basic meaning of replies need have unique numbers. The text of a reply can explain the specific reason for that reply to a human user.

Each TELNET line delimited by a numeric code and the TELNET EOL (or group of text lines bounded by coded lines) that is sent by the server is intended to be a complete reply message. It should be noted that the text of replies is intended for a human user. Only the reply codes and in some instances the first line of text are intended for programs.

The assigned reply codes relating to FTP are:

- 000 Announcing FTP.
- 010 Message from system operator.
- 020 Executed delay.
- 030 Server availability information.
- 050 FTP commentary or user information.
- 100 System status reply.
- 110 System busy doing...
- 150 File status reply.
- 151 Directory listing reply.
- 200 Last command received correctly.
- 201 An ABORT has terminated activity, as requested.
- 202 Abort request ignored, no activity in progress.
- 230 User is "logged in". May proceed.
- 231 User is "logged out". Service terminated.
- 232 Logout command noted, will complete when transfer done.
- 233 User is "logged out". Parameters reinitialized.
- 250 FTP file transfer started correctly.
- 251 FTP Restart-marker reply.
 - Text is: MARK yyyy = mmmm
 - where 'yyyy' is user's data stream marker (yours)
 - and mmmm is server's equivalent marker (mine)
 - (Note the spaces between the markers and '=').
- 252 FTP transfer completed correctly.
- 253 Rename completed.
- 254 Delete completed.
- 257 Closing the data connection, transfer completed.
- 300 Connection greeting message, awaiting input.
- 301 Current command incomplete (no <CRLF> for long time).
- 330 Enter password.

331 Enter account (if account required as part of login sequence).
332 Login first, please.
400 This service not implemented.
401 This service not accepting users now, goodbye.
402 Command not implemented for requested value or action.
430 Log-on time or tries exceeded, goodbye.
431 Log-on unsuccessful. User and/or password invalid.
432 User not valid for this service.
433 Cannot transfer files without valid account. Enter account and resend command.
434 Log-out forced by operator action. Phone site.
435 Log-out forced by system problem.
436 Service shutting down, goodbye.
450 FTP: File not found.
451 FTP: File access denied to you.
452 FTP: File transfer incomplete, data connection closed.
453 FTP: File transfer incomplete, insufficient storage space.
454 FTP: Cannot connect to your data socket.
455 FTP: File system error not covered by other reply codes.
456 FTP: Name duplication; rename failed.
457 FTP: Transfer parameters in error.
500 Last command line completely unrecognized.
501 Syntax of last command is incorrect.
502 Last command incomplete, parameters missing.
503 Last command invalid (ignored), illegal parameter combination.
504 Last command invalid, action not possible at this time.
505 Last command conflicts illegally with previous command(s).
506 Last command not implemented by the server.
507 Catchall error reply.
550 Bad pathname specification (e.g., syntax error).

DECLARATIVE SPECIFICATIONS

MINIMUM IMPLEMENTATION

In order to make FTP workable without needless error messages, the following minimum implementation is required for servers:

```
TYPE - ASCII Non-print
MODE - Stream
STRUCTURE - File
        Record
BYTE - 8
COMMANDS - USER, BYE, SOCK,
            TYPE, BYTE, MODE, STRU,
            for the default values
            RETR, STOR,
            NOOP.
```

The initial default values for transfer parameters are:

```
TYPE - ASCII Non-print
BYTE - 8
MODE - Stream
STRU - File
```

All Hosts must accept the above as the standard defaults.

CONNECTIONS

The server protocol interpreter shall "listen" on Socket 3. The user or user protocol interpreter shall initiate the full-duplex TELNET connections performing the ARPANET standard initial connection protocol (ICP) to server Socket 3. Server- and user- processes should follow the conventions of the TELNET protocol as specified in NIC #7104. Servers are under no obligation to provide for editing of command lines and may specify that it be done in the user Host. The TELNET connections shall be closed by the server at the user's request after all transfers and replies are completed.

The user-DTP must "listen" on the specified data sockets (send and/or receive); these may be the default user sockets (U+4) and (U+5) or a socket specified in the SOCK command. The server shall initiate the data connection from his own fixed sockets (S+2) and (S+3) using the specified user data socket and byte size (default - 8 bits). The

direction of the transfer and the sockets used will be determined by the FTP service command.

When data is to be transferred between two servers, A and B (refer to Figure 2), the user-PI, C, sets up TELNET connections with both server-PI's. He then sends A's fixed sockets, S(A), to B in a SOCK command and B's to A; replies are returned. One of the servers, say A, is then sent a PASV command telling him to "listen" on his data sockets rather than initiate an RFC when he receives a transfer service command. When the user-PI receives an acknowledgment to the PASV command, he may send (in either order) the corresponding service commands to A and B. Server B initiates the RFC and the transfer proceeds. The command-reply sequence is listed below where the messages are vertically synchronous but horizontally asynchronous:

User-PI - Server A	User-PI - Server B
-----	-----
C->A : ICP	C->B : ICP
C->A : SOCK HOST-B, SKT-S(B)	C->B : SOCK HOST-A, SKT-S(A)
A->C : 200 Okay	B->C : 200 Okay
C->A : PASV	
A->C : 200 Okay	
C->A : STOR	C->B : RETR

The data connection shall be closed by the server under the conditions described in the Section on Establishing Data Connections. If the server wishes to close the connection after a transfer where it is not required, he should do so immediately after the file transfer is completed. He should not wait until after a new transfer command is received because the user-process will have already tested the data connection to see if it needs to do a "listen"; (recall that the user must "listen" on a closed data socket BEFORE sending the transfer request). To prevent a race condition here, the server sends a secondary reply (257) after closing the data connection (or if the connection is left open, a "file transfer completed" reply (252) and the user-PI should wait for one of these replies before issuing a new transfer command.

COMMANDS

The commands are TELNET character string transmitted over the TELNET connections as described in the Section on FTP Commands. The command functions and semantics are described in the Section on Access Control Commands, Transfer Parameter Commands, FTP Service Commands, and Miscellaneous Commands. The command syntax is specified here.

The commands begin with a command code followed by an argument field.

The command codes are four or fewer alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus any of the following may represent the retrieve command:

RETR Retr retr ReTr rETr

This also applies to any symbols representing parameter values, such as A or a for ASCII TYPE. The command codes and the argument fields are separated by one or more spaces.

The argument field consists of a variable length character string ending with the character sequence <CRLF> (Carriage Return, Linefeed) for NVT-ASCII representation; for other negotiated languages a different end of line character might be used. It should be noted that the server is to take NO action until the end of line code is received.

The syntax is specified below in NVT-ASCII. All characters in the argument field are ASCII characters including any ASCII represented decimal integers. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

The following are all the currently defined FTP commands:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <acctno> <CRLF>
REIN <CRLF>
BYE <CRLF>
BYTE <SP> <byte size> <CRLF>
SOCK <SP> <Host-socket> <CRLF>
PASV <CRLF>
TYPE <SP> <type code> <CRLF>
STRU <SP> <structure code> <CRLF>
MODE <SP> <mode code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
APPE <SP> <pathname> <CRLF>
ALLO <SP> <decimal integer> [<SP> R <SP> <decimal integer>] <CRLF>
REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNTO <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
SITE <SP> <string> <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
```

NOOP <CRLF>

The syntax of the above argument fields (using BNF notation where applicable) is:

```
<username> ::= <string>
<password> ::= <string>
<acctno> ::= <string>
<string> ::= <char>|<char><string>
<char> ::= any of the 128 ASCII characters except <CR> and <LF>
<marker> ::= <pr string>
<pr string> ::= <pr char>|<pr char><pr string>
<pr char> ::= any ASCII code 33. through 126., printable
    characters
<byte size> ::= any decimal integer 1 through 255
<Host-socket> ::= <socket>|<Host number>, <socket>
<Host-number> ::= a decimal integer specifying an ARPANET Host.
<socket> ::= decimal integer between 0 and (2**32)-1
<form code> ::= N|T|C
<type code> ::= A[<SP> <form code>]|E [<SP> <form code>]|I|
L <SP> <byte size>
<structure code> ::= F|R
<mode code> ::= S|B|C
<pathname> ::= <string>
```

SEQUENCING OF COMMANDS AND REPLIES

The communication between the user and server is intended to be an alternating dialogue. As such, the user issues an FTP command and the server responds with a prompt primary reply. The user should wait for this initial primary success or failure response before sending further commands.

Certain commands require a second reply for which the user should also wait. These replies may, for example, report on the progress or completion of file transfer or the closing of the data connection. They are secondary replies to file transfer commands.

The third class of replies are informational and spontaneous replies which may arrive at any time. The user-PI should be prepared to receive them. These replies are listed below as spontaneous.

One important group of spontaneous replies is the connection greetings. Under normal circumstances, a server will send a 300 reply, "awaiting input", when the ICP is completed. The user should wait for this greeting message before sending any commands. If the server is unable to accept input right away, he should send a 000 "announcing FTP" or a 020 "expected delay" reply immediately and a

300 reply when ready. The user will then know not to hang up if there is a delay.

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a server may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

COMMAND-REPLY CORRESPONDENCE TABLE

COMMAND	SUCCESS	FAILURE
USER	230,330	430-432,500-505,507
PASS	230,330	430-432,500-507
ACCT	230	430-432,500-507
REIN	232,233	401,436,500-507
Secondary Reply	300	
BYE	231,232	500-505,507
BYTE	200,331	402,500-505,507
SOCK	200,331	500-505,507
PASV	200,331	500-507
TYPE	200,331	402,500-505,507
STRU	200,331	500-505,507
MODE	200,331	402,500-505,507
RETR	250	402,433,450,451,454,455,457, 500-505,507,550
Secondary Reply	252,257	452
STOR	250	402,433,451,454,455,457, 500-505,507,550
Secondary Reply	252,257	452,453
APPE	250	402,433,451,454,455,457,500-507, 550
Secondary Reply	252,257	452,453
ALLO	200,331	402,500-507
REST	200,331	500-507
RNFR	200	402,433,450,451,455,500-507,550
RNTO	253	402,433,450,451,455,456,500-507, 550
ABOR	201,202,331	500-507
DELE	254	402,433,450,451,455,500-507,550
LIST	250	402,433,450,451,454,455,457, 500-507,550
Secondary Reply	252,257	452
NLST	250	402,433,450,451,454,455,457, 500-507,550
Secondary Reply	252,257	452
SITE	200,331	402,500-507

File Transfer Protocol
(Aug. 12, 1973)
RFC 542 NIC 17759

STAT	100, 110, 150, 151, 331	450, 451, 455, 500-507, 550
HELP	030, 050	500-507
NOOP	200	500-505, 507
Spontaneous Replies	000, 010, 020, 300, 301, 251, 255	400, 401, 434-436

TYPICAL FTP SCENARIOS

TIP User wanting to transfer file from Host X to local printer:

1. TIP user opens TELNET connections by ICP to Host X socket 3.
2. The following commands and replies are exchanged:

TIP	HOST X
<----- 300 Awaiting input <CRLF>	
USER username <CRLF> ----->	
<----- 330 Enter Password <CRLF>	
PASS password <CRLF> ----->	
<----- 230 User logged in <CRLF>	
SOCK 65538 <CRLF> ----->	
<----- 200 Command received OK<CRLF>	
RETR this.file <CRLF> ----->	
(Host X initiates data connection to TIP socket 65538, i.e., PORT 1 receive)	
<----- 250 File transfer started <CRLF>	
<----- 252 File transfer completed <CRLF>	
BYE<CRLF> ----->	
<----- 231 User logged out <CRLF>	

3. Host X closes the TELNET and data connections.

Note: The TIP user should be in line mode.

User at Host U wanting to transfer files to/from Host S:

In general the user will communicate to the server via a mediating user-FTP process. The following may be a typical scenario. The user-FTP prompts are shown in parentheses, '---->' represents commands from Host U to Host S, and '<----' represents replies from Host S to Host U.

File Transfer Protocol
(Aug. 12, 1973)
RFC 542 NIC 17759

LOCAL COMMANDS BY USER	ACTION INVOLVED
ftp (host) multics<CR>	ICP to Host S, socket 3, establishing TELNET connections
username Doe <CR>	<---- 330 Awaiting input <CRLF>
password mumble <CR>	USER Doe<CRLF>----> <---- 330 password<CRLF>
retrieve (local type) ASCII<CR>	PASS mumble<CRLF>----> <---- 230 Doe logged in.<CRLF>
(local pathname) test 1 <CR>	User-FTP opens local file in ASCII.
(for.pathname) testp11<CR>	RETR test.p11<CRLF> ---->
(U+4)	Server makes data connection to
<CRLF>	<---- 250 File transfer starts
complete<CRLF>	<---- 252 File transfer
type Image<CR>	TYPE I<CRLF> ---->
byte 36<CR>	<---- 200 Command OK<CRLF>
BYTE 36<CR>LF ---->	BYTE 36<CR>LF ---->
store (local type) image<CR>	<---- 200 Command OK<CRLF>
(local pathname) file dump<CR>	User-FTP opens local file in Image.
(for.pathname) >udd>cn>fd<CR>	STOR >udd>cn>fd<CRLF> ---->
terminate	<---- 451 Access denied<CRLF>
	BYE <CRLF> ---->
	Server closes all connections.

Jon Postel
24 MAR 76

Revised FTP Reply Codes

This document describes a revised set of reply codes for the File Transfer Protocol.

The aim of this revision is to satisfy the goal of using reply codes to enable the command issuing process to easily determine the outcome of each command. The user protocol interpreter should be able to determine the success or failure of a command by examining the first digit of the reply code.

An important change in the sequencing of commands and replies which may not be obvious in the following documents concerns the establishment of the data connection.

In the previous FTP specifications when an actual transfer command (STOR, RETR, APPE, LIST, NLIST, MLFL) was issued the preliminary reply was sent after the data connection was established. This presented a problem for some user protocol interpreters which had difficulty monitoring two connections asynchronously.

The current specification is that the preliminary reply to the actual transfer commands indicates that the file can be transferred and either the connection was previously established or an attempt is about to be made to establish the data connection.

This reply code revision is a modification of the protocol in described in RFC 542, that is to say that the protocol implementation associated with socket number 21 (decimal) is the protocol specified by the combination of RFC 542 and this RFC.

A note of thanks to those who contributed to this work: Ken Pogran, Mark Krilanovich, Wayne Hathway, and especially Nancy Neigus.

Preceding page blank

Nancy Neigus
Ken Pogran
Jon Postel
24 MAR 76

A New Schema for FTP Reply Codes

Replies to File Transfer Protocol commands were devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups, such as USER, PASS and ACCT, or RNFR and RNTO. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

Details of the command-reply sequence will be made explicit in a state diagram.

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the user-process (the User-PI described in RFC 542) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

Formally, a reply is defined to contain the 3-digit code, followed by Space <SP>, followed by one line of text (where some maximum line length has been specified), and terminated by the TELNET end-of-line code. There will be cases, however, where the text is longer than a single line. In these cases the complete text must be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the TELNET connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to

indicate the state of the transaction. To satisfy all factions it was decided that both the first and last line codes should be the same.

Thus the format for multi-line replies is that the first line will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and TELNET <eol>.

For example:

```
123-First line
Second line
    234 A line beginning with numbers
        123 The last line
```

The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion.

This scheme allows standard system routines to be used for reply information (such as for the STAT reply), with "artificial" first and last lines tacked on. In the rare cases where these routines are able to generate three digits and a Space at the beginning of any line, the beginning of each text line should be offset by some neutral text, like Space.

This scheme assumes that multi-line replies may not be nested. We have found that, in general, nesting of replies will not occur, except for random system messages (called spontaneous replies in the previous FTP incarnations) which may interrupt another reply. Spontaneous replies are no longer defined; system messages (i.e. those not processed by the FTP server) will NOT carry reply codes and may occur anywhere in the command-reply sequence. They may be ignored by the User-process as they are only information for the human user.

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated response by the user-process. The first digit denotes whether the response is good, bad or incomplete. (Referring to the state diagram) an unsophisticated user-process will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error

occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information (e.g. RNT0 command without a preceding RNFR.)

There are four values for the first digit of the reply code:

1yz Positive Preliminary reply

The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g. the command is spelled the same with the same

arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g. after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

- x0z Syntax - These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.
- x1z Information - These are replies to requests for information, such as status or help.
- x2z Connections - Replies referring to the TELNET and data connections.
- x3z Authentication and accounting - Replies for the logon process and accounting procedures.
- x4z Unspecified as yet
- x5z File system - These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

The third digit gives a finer gradation of meaning in each of the function categories, specified by the second digit. The list of replies below will illustrate this. Note that the text associated with each reply is suggestive, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, should strictly follow the specifications in the last section; that is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined.

If additional codes are found to be necessary, the details should be submitted to the FTP committee, through Jon Postel.

A command such as TYPE or ALLO whose successful execution does not offer the user-process any new information will cause a 200 reply to be returned. If the command is not implemented by a particular Server-FTP process because it has no relevance to that computer system, for example ALLO at a TENEX site, a Positive Completion reply is still desired so that the simple User-process knows it can proceed with its course of action. A 202 reply is used in this case with, for example, the reply text: "No storage allocation necessary." If, on the other hand, the command requests a non-site-specific action and is unimplemented, the response is 502. A refinement of that is the 504 reply for a command that IS implemented, but that requests an unimplemented parameter.

- 200 Command okay
- 500 Syntax error, command unrecognized
 - [This may include errors such as command line too long.]
- 501 Syntax error in parameters or arguments
- 202 Command not implemented, superfluous at this site.
- 502 Command not implemented
- 503 Bad sequence of commands
- 504 Command not implemented for that parameter
- 110 Restart marker reply.
 - In this case the text is exact and not left to the particular implementation; it must read:
 - MARK yyyy = mmmm
 - where yyyy is User-process data stream marker, and mmmm is Server's equivalent marker. (note the spaces between the markers and "=".)
- 211 System status, or system help reply
- 212 Directory status
- 213 File status
- 214 Help message (on how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.)
- 120 Service ready in nnn minutes
- 220 Service ready for new user
- 221 Service closing TELNET connection (logged off if appropriate)
- 421 Service not available, closing TELNET connection.
 - [This may be a reply to any command if the service knows it must shut down.]

125 Data connection already open; transfer starting
225 Data connection open; no transfer in progress
425 Can't open data connection
226 Closing data connection; requested file action
successful (for example, file transfer or file
abort.)
426 Connection trouble, closed; transfer aborted.
227 Entering [passive, active] mode

230 User logged on, proceed
530 Not logged in
331 User name okay, need password
332 Need account for login
532 Need account for storing files

150 File status okay; about to open data connection.
250 Requested file action okay, completed.
350 Requested file action pending further information
450 Requested file action not taken: file unavailable
(e.g. file not found, no access)
550 Requested action not taken: file unavailable (e.g.
file busy)
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient storage
space in system
552 Requested file action aborted: exceeded storage
allocation (for current directory or dataset)
553 Requested action not taken: file name not allowed
354 Start mail input; end with <CR><LF>. <CR><LF>

Command-Reply Sequences

In this section, the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. Preliminary replies are listed first (with their succeeding replies under them), then positive and negative completion, and finally intermediary replies with the remaining commands from the sequence following. This listing forms the basis for the state diagrams, which will be presented separately.

ICP
120
220
220
421

Logon

USER
 230
 530
 500, 501, 421
 331, 332
PASS
 230
 202
 530
 500, 501, 503, 421
 332
ACCT
 230
 202
 530
 500, 501, 503, 421

Logoff

QUIT
 221
 500
REIN
 120
 220
 220
 421
 500, 502

Transfer parameters

SOCK
 200
 500, 501, 421, 530
PASV
 227
 500, 501, 502, 421, 530
ACTV
 227
 202
 500, 501, 421, 530
BYTE, MODE, TYPE, STRU
 200
 500, 501, 504, 421, 530

File action commands

ALLO
 200
 202
 500, 501, 504, 421, 530
REST
 500, 501, 502, 421, 530
 350
STOR
 125, 150
 (110)
 226, 250
 425, 426, 451, 552
 532, 450, 452, 553
 500, 501, 421, 530
RETR
 125, 150
 (110)
 226, 250
 425, 426, 451
 450, 550
 500, 501, 421, 530
LIST, NLST
 125, 150
 226, 250
 425, 426, 451
 450
 500, 501, 502, 421, 530
APPE
 125, 150
 (110)
 226, 250
 425, 426, 451, 552
 532, 450, 550, 452, 553
 500, 501, 502, 421, 530
MLFL
 125, 150
 226, 250
 425, 426, 451, 552
 532, 450, 550, 452, 553
 500, 501, 502, 421, 530
RNFR
 450, 550
 500, 501, 502, 421, 530
 350
RNTO
 250
 532, 553

500, 501, 502, 503, 421, 530
DELE
250
450, 550
500, 501, 502, 421, 530
ABOR
225, 226
500, 501, 502, 421
MAIL
354
250
451, 552
450, 550, 452, 553
500, 501, 502, 421, 530

Informational commands

STAT
211, 212, 213
450
500, 501, 502, 421, 530
HELP
211, 214
500, 501, 502, 421

Miscellaneous commands

SITE
200
202
500, 501, 530
NOOP
200
500

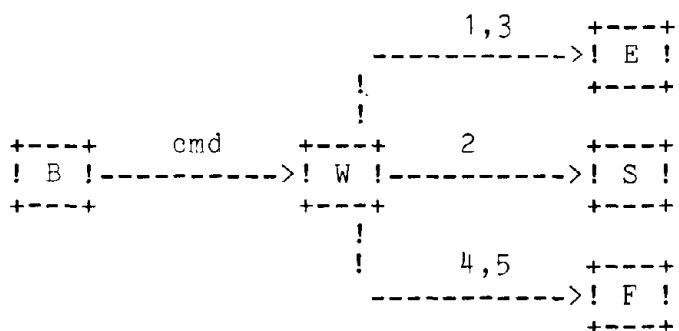
FTP State Diagrams

Here we present state diagrams for a very simple minded FTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of FTP commands or command sequences.

The command groupings were determined by constructing a model for each command then collecting together the commands with structurally identical models.

For each command or command sequence there are three possible outcomes: success (S), failure (F), and error (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

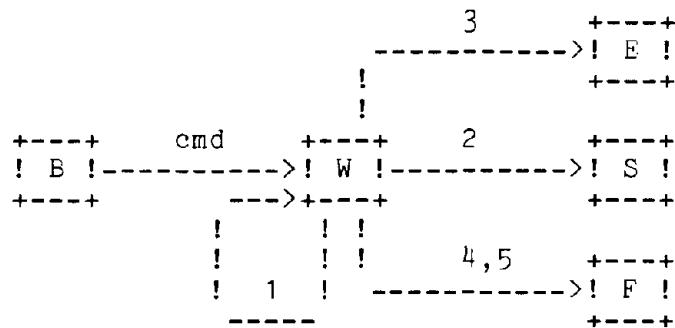
We first present the diagram that represents the largest group of FTP commands:



This diagram models the commands:

ABOR, ACTV, ALLO, BYTE, DELE, HELP, MODE, NOOP, PASV, QUIT,
SITE, SOCK, STAT, STRU, TYPE.

The other large group of commands is represented by a very similar diagram:

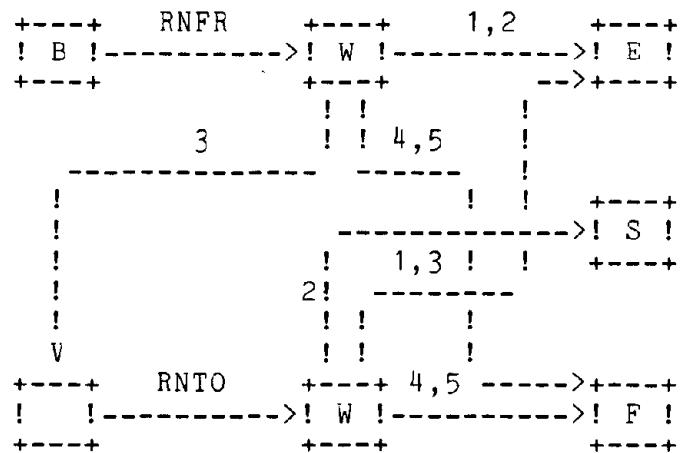


This diagram models the commands:

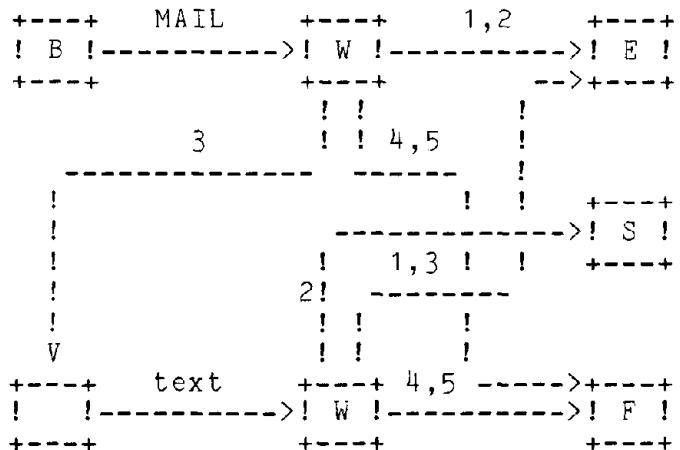
APPE, (ICP), LIST, MLFL, NLST, REIN, RETR, STOR.

Note that this second model could also be used to represent the first group of commands, the only difference being that in the first group the 100 series replies are unexpected and therefore treated as error, while the second group expects (some may require) 100 series replies.

The remaining diagrams model command sequences, perhaps the simplest of these is the rename sequence:

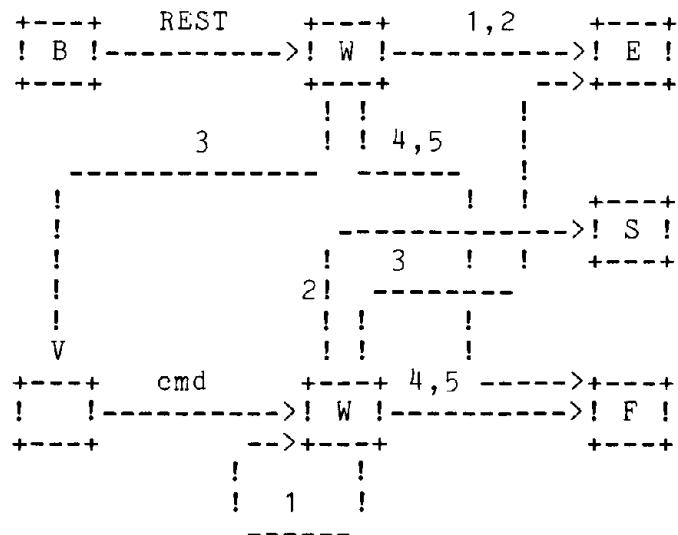


A very similar diagram models the Mail command:



Note that the "text" here is a series of lines sent from the user to the server with no response expected until the last line is sent, recall that the last line must consist only of a single period.

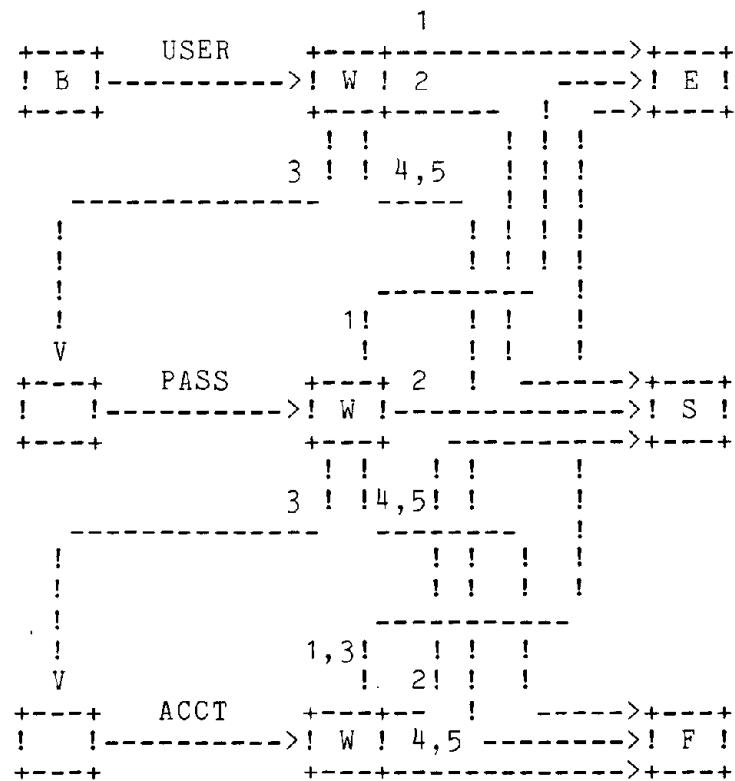
The next diagram is a simple model of the Restart command:



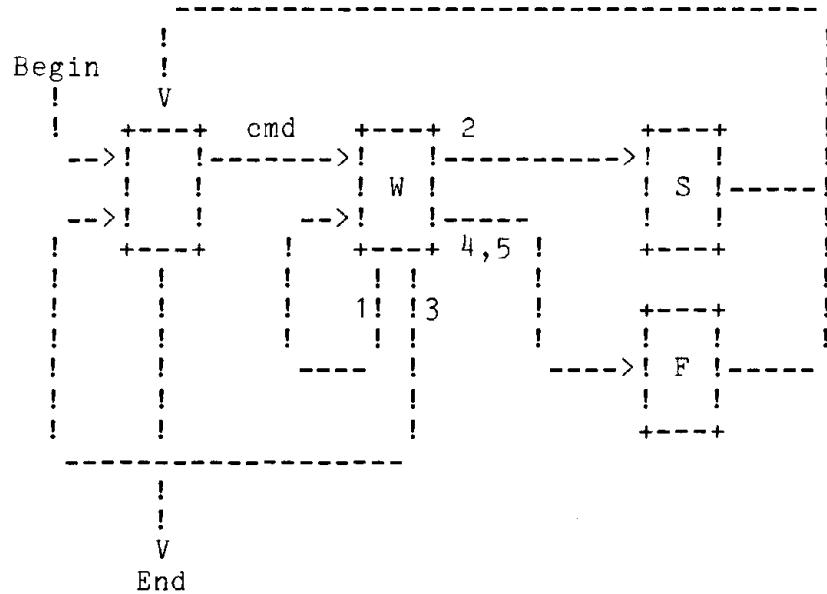
Where "cmd" is APPE, STOR, RETR, or MLFL.

We note that the above three models are similar, in fact the Mail diagram and the Rename diagram are structurally identical. The Restart differs from the other two only in the treatment of 100 series replies at the second stage.

The most complicated diagram is for the Logon sequence:



Finally we present a generalized diagram that could be used to model the command and reply interchange:



Mail Protocols

MAIL PROTOCOLS

Preceding page blank

JBP 18-FEB-76 12:40 27620
Mail Protocol
Jon Postel
18 February 1976

Mail Protocol

Introduction

This document describes the existing mail sending protocols. The mail sending protocol is a subset of the File Transfer protocol, consisting of two additional commands to the set of commands described in the specification of the File Transfer protocol.

Old FTP

A. McKenzie "File Transfer Protocol," RFC 454, NIC 14333,
16-Feb-73.

New FTP

N. Neigus "File Transfer Protocol," RFC 542, NIC 17759,
12-Jul-73.

J. Postel "Revised FTP Reply Codes," RFC 640, NIC 30843,
5-Jun-74.

Commands

Mail File (MLFL)

The intent of this command is to enable a user site to mail data (in form of a file) to another user at the server site. It should be noted that the files to be mailed are transmitted via the data connection in ASCII or EBCDIC type. (It is the user's responsibility to ensure that the type is correct.) These files should be appended to the destination user's mail by the server in accordance with serving HOST mail conventions. The mail may be marked as sent from the particular using HOST and the user specified by the 'USER' command. The argument field may contain one or more system or NIC idents (it is recommended that multiple idents be allowed so the same mail can easily be sent to several users), or it may be empty. If the argument field is empty or blank (one or more spaces), then the mail is destined for a printer or other designated place for site mail.

A NIC ident refers to the standard identification described in the NIC directory of Network Participants. A serving host may keep a table mapping NIC idents into system idents, although

JBP 18-FEB-76 12:40 27620
Mail Protocol
Jon Postel
18 February 1976

NIC idents are not required in the implementation. A system ident is the user's normal identification at the serving HOST. The use of system idents would allow a network user to send mail to other users who do not have NIC identification but whose system ident is known.

Mail (MAIL)

This command allows a user to send mail that is NOT in a file over the TELNET connection. The argument field may contain one or more system or NIC idents, or it may be empty. The idents are defined as above for the MLFL command. After the 'Mail' command is received, the server is to treat the following lines as text of the mail sent by the user. The mail text is to be terminated by a line containing only a single period, that is, the character sequence ".CRLF" in a new line. It is suggested that a modest volume of mail service should be free; i.e., it may be entered before a USER command.

Reply Codes

The MAIL and MLFL commands have the same reply codes as the Append (APPE) command, with the addition of the reply code for MAIL stating that mail is expected over the Telnet connection.

Old FTP

350 - Enter mail, terminate with <CR><LF>.<CR><LF>

New FTP

354 - Start mail input, end with <CR><LF>.<CR><LF>

Syntax

It is strongly urged that for consistency in the handling of mail at the various hosts that all mail sending subsystems or programs use the standard syntax convention documented in RFC 680 (NIC 32116,) for the text of the mail. This will help a great deal in allowing a user or program to intelligently process incoming mail.

Message Transmission Protocol
RFC 680, NIC 32116 (Apr. 30, 1975)

Network Working Group
RFC #680
NIC #32116
April 30, 1975

Message Transmission Protocol

Theodore H. Myer

D. Austin Henderson

BBN-TENEX

This document defines a number of message fields beyond those discussed in RFC 561. The overall message format is compatible with RFC 561; it makes extensive use of the miscellaneous fields defined within RFC 561. The purpose of this document is to establish ARPANET standards with regard to the syntax and semantics for these additional fields. It is fully expected that all fields discussed herein will not be automatically processed by all Message Servers; however, the standard is necessary so that sites which wish to make use of these fields have a standard to work with.

This document attempts to tread the narrow line between features for human processing and features for machine processing. The general feeling is that the fields listed are useful to people even if automatic processing is not supplied. In most cases, machine-readable notations have been enclosed in angle brackets (<>) to allow easy non-ambiguous ways for automatic processes to know whether and where to look in any field. The entire specification has been made excessively general to allow for experimentation. Future documents based on experience will try to be more specific. This is simply the next step following <RFC 561>.

This document is contained in two sections. Section I contains the relevant parts of RFC 561 which define the basic message syntax. Section II lists the new (and existing) header fields together with their proposed uses.

SECTION I: BASIC MESSAGE SYNTAX

<message>	::=	<header><crlf><body>
<header>	::=	<required header><optional header>
<required header>	::=	<date item><sender item>
<date item>	::=	DATE:<sp><date><sp>AT<sp> <time>-<zone><crlf>
<date>	::=	<vdate> ! <tdate>
<vdate>	::=	<dayofmonth><SP><vmonth><SP><vyear>
<tdate>	::=	<tmonth>/<dayofmonth>/<tyear>
<dayofmonth>	::=	one or two decimal digits
<vmonth>	::=	JAN ! FEB ! MAR ! APR ! MAY ! JUN ! JUL ! AUG ! SEP ! OCT ! NOV !

DEC
<tmonth> ::= one or two decimal digits
<vyear> ::= four decimal digits
<tyear> ::= two decimal digits
<zone> ::= EST ! EDT ! CST ! CDT ! MST ! MDT !
 PST ! PDT ! GMT ! GDT
<time> ::= four decimal digits
<sender item> ::= SENDER: <sp><user><sp>AT<sp><host>
 <crlf>
<optional header> ::= <subjects><optional items>
<subjects> ::= !<subject item> !
 <subject item><subjects>
<subject item> ::= SUBJECT:<sp><line><crlf>
<optional items> ::= <optional item> ! <optional item>
 <optional items>
<optional item> ::= <messid> ! <addressee item> !
 <other item>
<addressee item> ::= <addressee keyword>:<sp><addressee
 list><crlf>
<addressee keyword> ::= TO:! CC:! BCC:
<messid> ::= Message-ID:<sp>[<Net
 Address>}]<line>
 <crlf>
<other item> ::= <other keyword>:<sp><line><crlf>
 FROM ! IN-REPLY-TO! REFERENCES!
<other keyword> ::= KEYWORD ! PRECEDENCE !
 MESSAGE-CLASS!
 SPECIAL-HANDLING! AUTHENTICATION!
 ACCESSION-KEY
<address list> ::= <addressee> ! <addressee><addressee
 list>
<addressee> ::= <mailbox> ! <mailbox group>
<mailbox> ::= <user><host spec><attention spec>
<host spec> ::= !@<host>
<attention spec> ::= (ATTN:<sp><user list>
<user list> ::= <user> ! <user><user list>
<mailbox group> ::= <group name>:(<group numbers>)
<group numbers> ::= ! (<mailbox list>)
<mailbox list> ::= <mailbox> ! <mailbox>,<mailboxlist>
<body> ::= <line><CRLF> ! <line><CRLF><body>
<user> ::= <word>
<host> ::= a standard host name
<group name> ::= ! <word>
<line> ::= a string containing any of the 128
 ASCII
 characters except CR and LF
<word> ::= a string containing any of the 128
 ASCII
 characters except CR, LF, and SP
<CRLF> ::= CR LF
<SP> ::= space

Notes: 1. A message may have at most one MESSAGE-ID item.
2. All items with the same keyword must be grasped together.

Please note the following:

- (1) The case (upper or lower) of keywords -- specifically, "FROM", 'DATE", 'SUBJECT', 'AT', <host>, <zone>, <vmonth> and <keyword> -- is insignificant. Although 'FROM', for example, appears in upper-case in the formal syntax above, in the header of an actual message it may appear as 'From', 'from', or 'FRom', etc.
- (2) No attempt has been made to legislate the format of <user> except to exclude spaces from it.
- (3) The time has no internal punctuation.

SECTION II: MESSAGE HEADER FIELDS

A. ORIGINATOR SPECIFICATION FIELDS

FROM

This field contains the identity of the person who wished this message to be sent. This is expected to be the originator field which is specified by the user in the case that the message is being entered by one person for another. The message-creation process should default this field to be the user entering the message. [The usage for FROM and SENDER differs from that of RFC 561.]

SENDER

This field contains the identity of the person who sends the message. This field is expected to be set by the message-creation process automatically. It is possible that some sites will not include this field in external communications.

AUTHENTICATION

This field contains a description of which originator fields have been authenticated, and by which operating systems. This field should be created by message transmission and/or reception processes (FTP/Operating System level).

It is expected that current system will be able to authenticate only the SENDER field; however, later systems might have mechanisms to verify that the FROM actually authorized the SENDER to act on his/her behalf. It is expected that, when the FROM is authenticated, the SENDER will no longer be necessary for external distribution.

B. REFERENCE SPECIFICATION FIELDS

MESSAGE-ID

This field contains a unique identifier to refer to this message. The format for a message identifier is:

[Net Address]Text String CRLF

Examples:

[ISIB]7-DEC-74.14:23:45
[ARC]QJOURNAL 39274a3

The uniqueness of the message identifier is guaranteed by each net-address message processor making the text which follows the net-address unique for that net-address. This, specifically says net-address and not site name. This would allow BBN (for instance) to allocate unique identifiers over all four machines, which may be addressed as BBN within the message system, thus producing a more integrated service for their users.

The text following the net-address is not defined here, as the problems associated with this specification are too great at this time. However, the net-address should allow automatic processes to determine if they can deal intelligently with the following text. Several types of automatic processing by the local message reader are thus possible: 1) if the site uses a filing mechanism known to the reader, the reader can retrieve the message 2) if the site supports remote message access (protocol not currently defined), the message id can be passed to the remote site and the message can be retrieved by the reader; 3) finally, if the message has been filed in the Datacomputer (using the entire message id [including net-address] as the handle), the reader can retrieve it from the Datacomputer.

IN-REPLY-TO

The contents of this field identify previous correspondence which this message answers. If message identifiers are used in this field, they should be enclosed in angle brackets (<>).

REFERENCES

The contents of this field identify other correspondence which this message references. If message identifiers are used, they should be enclosed in angle brackets (<>).

KEYWORDS

This field contains keywords or phrases from the message, separated by commas.

C. RECEIVER SPECIFICATION FIELDS

TO

This field contains the identity of the primary receivers of the message.

CC

This field contains the identity of the secondary receivers of the message.

BCC

This field contains the identity of the tertiary receivers of the message. This field should not be made available to the primary and secondary receivers, but it may be recorded to provide information for access control.

D. MESSAGE-TYPE SPECIFICATION FIELDS

PRECEDENCE

This field describes the importance and urgency of the message. Machine-readable notations will be enclosed in angle brackets (<>). <PRIORITY> means that the message should be delivered as soon as possible. <ROUTINE> means that Priority processing is not necessary. Plain text may also be included in

this field.

MESSAGE-CLASS

This field describes the "legal" status of the message. Examples: Official, Unofficial, Record, Off the Record, Junk Mail. No automatic processing of this field is immediately expected. Certain message creation processes might, for example, always insert:

MESSAGE CLASS: Unofficial ARPANET Message

SPECIAL-HANDLING

This field contains any special instructions with regard to the handling of the message at the receiver's end. Machine-readable notations will be enclosed in angle brackets (<>). <PRIVATE> means that the message reception process should not aid the user in circulating copies to others. Plain text may also be included in this field.

REMOTE JOB ENTRY PROTOCOL

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

Robert Bressler, MIT-DMCG
Richard Guida, MIT-DMCG
Alex McKenzie, BBN-NET

Obsoletes RFC 360

REMOTE JOB ENTRY PROTOCOL

Preceding page blank

REMOTE JOB ENTRY PROTOCOL

INTRODUCTION

Remote job entry is the mechanism whereby a user at one location causes a batch-processing job to be run at some other location. This protocol specifies the Network standard procedures for such a user to communicate over the Network with a remote batch-processing server, causing that server to retrieve a job-input file, process the job, and deliver the job's output file(s) to a remote location. The protocol uses a TELNET connection (to a special standardized logger, not socket 1) for all control communication between the user and the server RJE processes. The server-site then uses the File Transfer Protocol to retrieve the job-input file and to deliver the output file(s).

There are two types of users: direct users (persons) and user processes. The direct user communicates from an interactive terminal attached to a TIP or any host. This user may cause the input and/or output to be retrieved/sent on a specific socket at the specified host (such as for card readers or printers on a TIP), or the user may have the files transferred by file-id using File Transfer Protocol. The other type of user is a RJE User-process in one remote host communicating with the RJE Server-process in another host. This type of user ultimately receives its instructions from a human user, but through some unspecified indirect means. The command and response streams of this protocol are designed to be readily used and interpreted by both the human user and the user process.

A particular user site may choose to establish the TELNET control connection for each logical job or may leave the control connection open for extended periods. If the control connection is left open, then multiple job-files may be directed to be retrieved or optionally (to servers that are able to determine the end of one logical job by the input stream and form several jobs out of one input file) one continuous retrieval may be done (as from a TIP card reader). This then forms a "hot" card reader to a particular server with the TELNET connection serving as a "job monitor". Since the output is always transferred job at a time per connection to the output socket, the output from this "hot" reader would appear when ready as if to a "hot" printer. Another possibility for more complex hosts is to attach an RJE User-process to a card reader and take instructions from a lead control card, causing an RJE control TELNET to be opened to the appropriate host with appropriate log-on and input retrieval commands. This card reader would appear to the human user as a Network "hot" card reader. The details of this RJE User-process are beyond the scope of this protocol.

GENERAL SPECIFICATIONS

User

A human user at a real terminal or a process that supplies the command control stream causing a job to be submitted remotely will be termed the User. The procedure by which a process user receives its instructions is beyond the scope of this protocol.

User TELNET

The User communicates its commands over the Network in Network Virtual Terminal code through a User TELNET process in the User's Host. This User TELNET process initiates its activity via ICP to the standard "RJE Logger" socket (socket 5) at the desired RJE-server Host.

RJE-Server TELNET

The RJE-server process receives its command stream from and sends its response stream to the TELNET channel through an RJE-server TELNET process in the server host. This process must listen for the ICP on the "RJE Logger" socket (and cause appropriate ICP socket shifting).

TELNET Connection

The command and response streams for the RJE mechanism are via a TELNET-like connection to a special socket with full specifications according to the current NWG TELNET protocol.

RJE-Server

The RJE-Server process resides in the Host which is providing Remote Batch Job Entry service. This process receives input from the RJE-server TELNET, controls access through the "log-on" procedure, retrieves input job files, queues jobs for execution by the batch system, responds to status inquiries, and transmits job output files when available.

User FTP

All input and output files are transferred under control of the RJE-server process at its initiative. These files may be directly transferred via Request-for-connection to a specific Host/socket or they may be transferred via File Transfer Protocol. If the latter method is used, then the RJE-server acts through its local User FTP process to cause the transfer. This process initiates

activity by an active Request-for-connection to the "FTP Logger" in the foreign host.

Server FTP

This process in a remote host (remote from the RJE-server) listens for an ICP from the User FTP and then acts upon the commands from the User FTP causing the appropriate file transfer.

FTP

When File Transfer Protocol is used for RJE files, the standard FTP mechanism is used as fully specified by the current NWG FTProtocol.

RJE Command Language

The RJE system is controlled by a command stream from the User over the TELNET connection specifying the user's identity (log-on), the source of the job input file, the disposition of the job's output files, enquiring about job status, altering job status or output disposition. Additional commands affecting output disposition are includable in the job input file. This command language is explicitly specified in a following section of this protocol.

RJE Command Replies

Every command input from the User via TELNET calls for a response message from the RJE-server to the User over the TELNET connection. Certain other conditions also require a response message. These messages are formatted in a standardized manner to facilitate interpretation by both human Users and User processes. A following section of this protocol specifies the response messages.

RJE COMMANDS OVER TELNET CONNECTION

GENERAL CONVENTIONS

1. Each of the commands will be contained in one input line terminated by the standard TELNET "crlf". The line may be of any length desired by the user (explicitly, not restricted to a physical terminal line width). The characters "cr" and "lf" will be ignored by the RJE-server except in the explicit order "crlf" and may be used as needed for local terminal control.
2. All commands will begin with a recognized command name and may then contain recognized syntactic element strings and free-form variable strings (for user-id, file-ids, etc.). Recognized words consist of alphanumeric strings (letters and digits) or punctuation. Recognized alphanumeric string elements must be separated from each other and from unrecognizable strings by at least one blank or a syntactically permitted punctuation. Other blanks may be used freely as desired before or after any syntactic element ("blank" is understood here to mean ASCII SPACE (octal 040); formally: <blank> ::= <blank><ASCII SPACE> | <ASCII SPACE> ; thus, a sequence of SPACES is also permissible in place of <blank>, although there is no syntactic necessity for there to be more than one). The "=" after the command name in all commands except OUT and CHANGE is optional.
3. Recognized alphanumeric strings may contain upper case letters or lower case letters in any mixture without syntactic differentiation. Unrecognizable strings will be used exactly as presented with full differentiation of upper and lower case input, unless the host finally using the string defines otherwise.
4. There are two types of Unrecognizable strings: final and imbedded. Final strings appear as the last syntactic element of a command and are parsed as beginning with the next non-blank character of the input stream and continuing to the last non-blank character before the "crlf".

Imbedded strings include "job-id" and "job-file-id" in the OUT, CHANGE, and ALTER commands. At present these fields will be left undelimited since they must only be recognizable by the server host which hopefully can recognize its own job-ids and file-names.

SYNTAX

The following command descriptions are given in a BNF syntax. Names within angle brackets are non-terminal syntactic elements which are expanded in succeeding syntactic equations. Each equation has the

defined name on the left of the ::= and a set of alternative definitions, separated by vertical lines "|", on the right.

REINITIALIZE

REINIT

This command puts the user into a state identical to the state immediately after a successful connection to the RJE-server, prior to having sent any commands over the TELNET connection. The effective action taken is that of an ABORT and a flushing of all INPUT, OUTPUT and ID information. Naturally, the user is still responsible for any usage charges incurred prior to his REINIT command. The TELNET connection is not affected in any way.

USER

User = <user-id>

This command must be the first command over a new TELNET connection. As such, it initiates a "logon" sequence. The response to this command is one of the following:

1. User code in error.
2. Enter password (if user code ok).
3. Log-on ok, proceed (if no password requested).

Another USER command may be sent by the User at any time to change Users. Further input will then be charged to the new user. A server may refuse to honor a new user command if it is not able to process it in its current state (during input file transfer, for example), but the protocol permits the USER command at any time without altering previous activity. An incorrect subsequent USER command or its following PASS command are to be ignored with error response, leaving the original User logged-in.

It is permissible for a server to close the TELNET connection if the initial USER/PASS commands are not completed within a server specified time period. It is not required or implied that the "logged-on" User's user-id be the one used for file transfer or job execution, but only identifies the submitter of the command stream. Servers will establish their own rules relating user-id with the job-execution-user for Job or Output alteration commands.

Successful "log-on" always clears any previous Input or Output default parameters (INID, etc.).

PASS

Pass = <password>

This command immediately follows a USER command and completes the "log-on" procedure. Although a particular Server may not require a password and has already indicated "log-on ok" after the USER command, every Server must permit a PASS command (and possibly ignore it) and acknowledge it with a "log-on ok" if the log-on is completed.

BYE

BYE

This command terminates a USER and requests the RJE server to close the TELNET connection. If input transfer is not in progress, the TELNET connection may be closed immediately; if input is in progress, the connection should remain open for result response and then be closed. During the interim, a new USER command (and no other command) is acceptable.

An unexpected close on the TELNET connection will cause the server to take the effective action of an ABORT and a BYE.

INID/INPASS

INID = <user-id>
INPASS = <password>

The specified user-id and password will be sent in the File Transfer request to retrieve the input file. These parameters are not used by the Server in any other way. If this command does not appear, then the USER/PASS parameters are used.

INPATH/INPUT

INPATH = <file-id>
INPUT = <file-id>
INPUT

NOTE: The following syntax will be used for output as well.

```
<file-id> ::= <host-socket> | <host-file>
<host-socket> ::= <host>, <socket> <attributes> |
                  <socket> <attributes>
no <host> part implies the User-site host
<host> ::= <integer>
<socket> ::= <integer>
```

```
<integer> ::= D<decimal-integer> | O<octal-integer> |
              H<hexadecimal-integer>
<host-file> ::= <host><attributes>/<pathname>
<attributes> ::= <empty> | :<transmission><code>
<transmission> ::= <empty> | T | A | N
    <empty> implies default which is N for Input files
    and A for Output files
    T      specifies TELNET-like coding with embedded
          "crlf" for new-line, "ff" for new-page
    N      specifies FTP blocked transfer with record
          marks but without other carriage-control
    A      specifies FTP blocked records with ASA
          carriage-control
          (column 1 of image is forms control)
<code> ::= <empty> | E
    <empty> specifies NVT ASCII code
    E specifies EBCDIC
<pathname> ::= <any string recognized by the FTP Server at
               the site of the file>
```

The <file-id> syntax is the general RJE mechanism for specifying a particular file source or destination for input or output. If the <host-socket> form is used then direct transfer will be made by the RJE-Server to the named socket using the specified <attributes>. If the <host-file> form is used then the RJE-server will call upon its local FTP-user process to do the actual transfer. The data stream in this mode is either TELNET-like ASCII or blocked records (which may use column 1 for ASA carriage-control). Although A mode is permitted on input (column 1 is deleted) the usual mode is the default N. The output supplies carriage-control in the first character of each record ("blank" = single-space, "1" = new-page, etc.), while the optional N mode transfers the data only (as to a card punch, etc.).

The <pathname> is an arbitrary Unrecognized string which is saved by RJE-server and sent back over FTP to the FTP-server to retrieve or store the appropriate files.

INPATH or INPUT commands first store the specified <file-id> if one is supplied, and then the INPUT command initiates input. The INPATH name may be used to specify a file-id for later input and the INPUT command without file-id will cause input to initiate over a previously specified file-id. An INPUT "crlf" command with no previous <file-id> specified is illegal.

ABORT

ABORT

This command aborts any input retrieval in progress, discards already received records, and closes the retrieval connection.
Note: ABORT with parameters is an Output Transmission control (see below).

OUTUSER/OUTPASS

OUTUSER = <user-id>
OUTPASS = <password>

The specified user-id and password will be sent in the File Transfer request to send the output file(s). These parameters are not used by the Server in any other way. If this command does not appear, then the USER/PASS parameters are used.

OUT

OUT <out-file> = <disp>

<out-file> ::= <empty> | <job-file-id>
<empty> implies the primary print file of the job
<job-file-id> ::= <string representing a specific output file from the job as recognized by the Server>
<disp> ::= <empty><file-id> | (H) | (S)<file-id>|(D)
<empty> specifies Transmit then discard
(H) specifies Hold-only, do not transmit
(S) specifies Transmit and Save
(D) specifies discard without transmitting

Note: Parentheses are part of the above elements.

<file-id> ::= (same as for INPUT command)

This command specifies the disposition of output file(s) produced by the job. Unspecified files will be Hold-only by default. The OUTUSER, OUTPASS, and OUT commands must be specified before the INPUT command to be effective. These commands will affect any following jobs submitted by this USER over this RJE-TELNET connection. A particular job may override these commands by NET control cards on the front of the input file.

Once output disposition is specified by this OUT command or by a NET OUT card, the information is kept with the job until final output disposition, and is modifiable by the CHANGE command.

On occasion, the server may find that the destination for the output is "busy" (i.e., RFC to either Server-FTP or specified socket is refused), or that the host which should receive the output is dead. In these cases, the server should wait several minutes and then try to transmit again.

OUTPUT RE-ROUTE

```
CHANGE <job-id><blank><out-file> = <disp>
```

This command changes the output disposition supplied with the job at submission. The <job-id> is assumed recognizable by the RJE-server, who may verify if this USER is authorized to modify the specified job. After the job is identified, the other information has the same syntax and semantics as the original OUT command. CHANGE command may be specified for a job-file-id which was not mentioned at submission time and has the same effect as an original OUT command.

OUTPUT CONTROLS DURING TRANSMISSION

```
<command><blank><count><blank><what>  
<command> ::= RESTART | RECOVER | BACK | SKIP |  
ABORT | HOLD
```

These commands specify (respectively):

```
Restart the transmission (new RFC, etc.)  
Recover restarts transmission from last FTP  
Restart-marker-reply  
(see FTP).  
Back up the output "count" blocks  
Skip the output forward "count" blocks  
Abort the output, discarding it  
Abort the output, but Hold it
```

```
<count> ::= <empty> | <integer>  
<empty> implies 1 where defined  
<what> ::= @<file-id> | <job-id><job-file-id>  
<disp> ::= (same as for OUT command)  
<file-id> ::= (same as for INPUT command)  
<integer> ::= (same as for INPUT command)  
<job-id> ::= <server recognized job identifier which was supplied  
at INP completion by the server>  
  
<job-file-id> ::= <server recognized file identifier or if missing  
then the prime printer output of the specified  
job>
```

This collection of commands will modify the transmission of output in progress or recently aborted. If output transmission is cut-off before completion, then the RJE-server will either try to resend the entire file if the file's <disp> was Transmit-and-discard or will Hold the file for further User control if the <disp> was (S) transmit-and-Save. Either during transmission, during the Save part of a transmit-and-Save, or for a Hold-only file, the above commands may be used to control the transmission. The @<file-id> form of <what> is permitted only if transmission is actually in progress.

If the file's state is inconsistent with the command, then the command is illegal and ignored with reply.

STATUS

```
STATUS <job-id>
STATUS <job-id><blank><job-file-id>
```

These commands request the status of the RJE-server, a particular job, or the transmission of an output or input file, respectively. The information content of the Status reply is site dependent.

CANCEL/ALTER

```
CANCEL <job-id>
ALTER <job-id><blank><site dependent options>
```

These commands change the course of a submitted job. CANCEL specifies that the job is to be immediately terminated and any output discarded. ALTER provides for system dependent options such as changing job priority, process limits, Terminate without Cancel, etc.

OP

```
OP (any string)
```

The specified string is to be displayed to the Server site operator when any following job is initiated from the batch queue of the Server. This command usually appears in the input file as a NET OP control card, but may be a TELNET command. It is cancelled as an all-jobs command by an OP "crlf" command (no text supplied).

RJE CONTROL CARDS IN THE INPUT FILE

Certain RJE commands may be specified by control cards in the front of the input file. If these controls appear, they take precedence over the same command given thru the RJE-TELNET connection and affect only this specific job. All these RJE control cards must appear as the first records of the job's input-file. They all contain the control word NET in columns 1 through 3. Scanning for these controls stops when the first card without NET in col 1-3 is encountered.

The control commands appear in individual records and are terminated by the end-of-record (usually an 80 column card-image). Continuation is permitted onto the next record by the appearance of NET+ in columns 1-4 of the next record. Column 5 of the next record immediately follows the last character of the previous record.

```
NET OUTUSER = <user-id>
NET OUTPASS = <password>
NET OUT <out-file> = <disp>
NET OP <any string>
```

See the corresponding TELNET command for details. One option permitted by the NET OUTUSER and NET OUT controls not possible from the TELNET connection is specification of different OUTUSERS for different OUTS, since the TELNET stored and supplies only an initial OUTUSER, but the controls may change OUTUSERS before each OUT control is encountered.

RJE USE OF FILE TRANSFER PROTOCOL

Most non-TIP files will be transferred to or from the RJE-server through the FTP process. RJE-server will call upon its local FTP-user supplying the Host, File-pathname, User-id, Password, and Mode of the desired transfer. FTP-user will then connect to its FTP-server counterpart in the specified host and set up a transfer path. Data will then flow through the RJE-FTP interface in the Server, over the Network, from/to the foreign FTP-server and then from/to the specified File-pathname in the foreign host's file storage space. On output files, the file-pathname may be recognized by the foreign host as directions to a printer or the file may simply be stored; a User-RJE-process can supply an output <file-id> by default which is recognized by its own Server-FTP as routing to a printer.

Although many specifics of the RJE-Server/User-FTP interface are going to be site dependent, there are several FTP options which will be used in a standard way by RJE-Servers:

REMOTE Job Entry Protocol
 (Oct. 16, 1972)
 RFC 407 NIC 12112

1. A new FTP connection will be initiated for each file to be transferred. The connection will be opened with the RJE User supplied User-id (OUTUSER or INUSER) and Password.
2. The data bytesize will be 8 bits.
3. The FTP Type, Structure, and Mode parameters are determined by the RJE transfer direction (I/O), and the <transmission> and <code> options supplied by the User:

I/O	<TRANS>	<CODE>	FTP-TYPE	FTP-STRUCTURE	FTP-MODE
I*	N	-	A	R	B
I	N	E	E	R	B
I	T	-	A	F	S
I	T	E	E	F	S
I	A	-	P	R	B
I	A	E	F	R	B
O*	A	-	P	R	B
O	A	E	F	R	B
O	N	-	A	R	B
O	N	E	E	R	B
O	T	-	A	F	S
O	T	E	E	F	S

(*indicates default)

4. The service commands used will be Retrieve for input and Append (with create) for output. The FTP pathname will be the <pathname> supplied by the RJE User.
5. On output in B form, the User-FTP at the RJE-Server site will send Restart-markers at periodic intervals (like every 100 lines, or so), and will remember the latest Restart-marker-reply with the file. If the file transfer is not completed and the <disp> is (S) then the file will be held pending User intervention. The User may then use the RECOVER command to cause a FTP restart at the last remembered Restart-marker-reply.
6. The FTP Abort command will be used for the RJE ABORT and CANCEL commands.
7. For transfers where the FTP-MODE is defined as B, the user FTP may optionally attempt to use H mode.

The specific form of the FTP commands used by an RJE-Server site, and the order in which they are used will not be specified in this protocol.

Errors encountered by FTP fall into three categories: a) access errors or no storage space error; b) command format errors; and c) transfer failure errors. Since the commands are created by the RJE-Server process, an error is a programming problem and should be logged for attention and the situation handled as safely as possible. Transmission failure or access failure on input cause an effective ABORT and user notification. Transmission failure on output causes RESTART or Save depending on <disp> (see OUT command). Access failure on output is a problem since the User may not be accessible. A status response should be queued for him, should he happen to inquire; a <disp> = (S) file should be Held; and a <disp> = <empty> transmit-and-discard file should be temporarily held and then discarded if not claimed. "Temporarily" is understood here to mean at least several days, since particularly in the case of jobs which generate voluminous output at great expense to the User, he should be given every chance to retrieve his rightful output. Servers may elect, however, to charge the User for the file-storage space occupied by the held output.

REPLIES OVER THE TELNET CONNECTION

Each action of the RJE-server, including entry of each TELNET command, is noted over the TELNET connection to the User. These RJE-server replies are formatted for Human or Process interpretation. They consist of a leading 3-digit numeric code followed by a blank followed by a text explanation of the message. The numeric codes are assigned by groups for future expansion to hopefully cover other protocols besides RJE (like FTP). The numeric code is designed for ease of interpretation by processes. The three digits of the code are interpreted as follows:

The first digit specified the "type" of response indicated:

000

These "replies" are purely informative, and are issued voluntarily by the Server to inform a User of some state of the server's system.

100

Replies to a specific status inquiry. These replies serve as both information and as acknowledgment of the status request.

200

Positive acknowledgment of some previous command/request. The reply 200 is a generalized "ok" for commands which require no other comment. Other 2xx replies are specified for specific successful actions.

300

Incomplete information supplied so far. No major problem, but activity cannot proceed with the input specified.

400

Unsuccessful reply. A request was correctly specified, but could not be correctly completed. Further attempts will require User commands.

500

Incorrect or illegal command. The command or its parameters were invalid or incomplete from a syntactic view, or the command is inconsistent with a previous command. The command in question has been totally ignored.

600-900

Reserved for expansion

The second digit specifies the general subject to which the response refers:

x00-x29

General purpose replies, not assignable to other subjects.

x30

Primary access. These replies refer to the attempt to "log-on" to a Server service (RJE, FTP, etc.).

x40

Secondary access. The primary Server is commenting on its ability to access a secondary service (RJE must log-on to a remote FTP service).

x50

FTP results.

x60

RJE results.

x70-x99

Reserved for expansion.

The final digit specifies a particular message type. Since the code is designed for an automaton process to interpret, it is not necessary for every variation of a reply to have a unique number, only that the basic meaning have a unique number. The text of a reply can explain the specific reason for the reply to a human User.

Each TELNET line (ended by "crlf") from the Server is intended to be a complete reply message. If it is necessary to continue the text of a reply onto following lines, then those continuation replies contain the special reply code of three blanks.

The assigned reply codes relating to RJE are:

000 General information message (time of day, etc.)
030 Server availability information
050 FTP commentary or user information
060 RJE or Batch system commentary or information
100 System status reply
150 File status reply
151 Directory listing reply
160 RJE system general status reply
161 RJE job status reply
200 Last command received ok
201 An ABORT has terminated activity, as requested
202 ABORT request ignored, no activity in progress
203 The requested Transmission Control has taken effect
204 A REINIT command has been executed, as requested
230 Log-on completed
231 Log-off completed, goodbye.
232 Log-off noted, will complete when transfer done
240 File transfer has started
250 FTP File transfer started ok
251 FTP Restart-marker-reply
Text is: MARK yyyy = mmmm
where yyyy is data stream marker value (yours)
and mmmm is receiver's equivalent mark (mine)
252 FTP transfer completed ok
253 Rename completed
254 Delete completed
260 Job <job-id> accepted for processing
261 Job <job-id> completed, awaiting output transfer
262 Job <job-id> Cancelled as requested
263 Job <job-id> Altered as requested to state <status>
264 Job <job-id>, <job-file-id> transmission in progress
300 Connection greeting message, awaiting input
301 Current command not completed (may be sent after
suitable delay, if not "crlf")
330 Enter password (may be sent with hide-your-input mode)
360 INPUT has never specified an INPATH
400 This service is not implemented
401 This service is not accepting log-on now, goodbye.
430 Log-on time or tries exceeded, goodbye.
431 Log-on unsuccessful, user and/or password invalid
432 User not valid for this service
434 Log-out forced by operator action, please phone site
435 Log-out forced by system problem
436 Service shutting down, goodbye
440 RJE could not log-on to remote FTP for input transfer
441 RJE could not access the specified input file thru FTP
442 RJE could not establish <host-socket> input connection

443 RJE could not log-on to remote FTP for output delivery
444 RJE could not access file space given for output
445 RJE could not establish <host-socket> output connection
450 FTP: The named file does not exist (or access denied)
451 FTP: The named file space not accessible by YOU
452 FTP: Transfer not completed, data connection closed
453 FTP: Transfer not completed, insufficient storage space
460 Job input not completed, ABORT performed
461 Job format not acceptable for processing, Cancelled
462 Job previously accepted has mysteriously been lost
463 Job previously accepted did not complete
464 Job-id referenced by STATUS, CANCEL, ALTER, CHANGE, or
Transmission Control is not known (or access denied)
465 Request Alteration is not permitted for the specified job
466 Un-deliverable, un-claimed output for <job-id> discarded
467 Requested REINIT not accomplished
500 Last command line completely unrecognized
501 Syntax of the last command is incorrect
502 Last command incomplete, parameters missing
503 Last command invalid, illegal parameter combination
504 Last command invalid, action not possible at this time
505 Last command conflicts illegally with previous command(s)
506 Requested action not implemented by this Server
507 Job <job-id> last command line completely unrecognized
508 Job <job-id> syntax of the last command is incorrect
509 Job <job-id> last command incomplete, parameters missing
510 Job <job-id> last command invalid, illegal parameter
combination
511 Job <job-id> last command invalid, action impossible at
this time
512 Job <job-id> last command conflicts illegally with previous
command(s)

SEQUENCING OF COMMANDS AND REPLIES

The communication between the User and Server is intended to be an alternating dialogue. As such, the User issues an RJE command and the Server responds with a prompt primary reply. The User should wait for this initial success or failure response before sending further commands.

A second type of reply is sent by Server asynchronously with respect to User commands. These replies report on the progress of a job submission caused by the INPUT command and as such are secondary replies to that command.

The final class of Server "replies" are strictly informational and may arrive at any time. These "replies" are listed below as spontaneous.

COMMAND-REPLY CORRESPONDENCE TABLE

COMMAND	SUCCESS	FAILURE
REINIT	204	467,500-505
USER	230,330	430-432,500-505
PASS	230	430-432,500-505
BYE	231,232	500-505
INID	200	500-505
INPASS	200	500-505
INPATH	200	500-505
INPUT	240	360,440-442,500-505
sec. input retrieval	260	460,461
sec. job execution	261	462,463
sec. output transmission	-	443-445,466
ABORT (input)	201,202	500-505
OUTUSER	200	500-505
OUTPASS	200	500-505
OUT	200	500-505
CHANGE	200	500-505
RESTART/RECOVER/BACK		
/SKIP/ABORT (output)/HOLD	203	464,500-506
STATUS	1xx,264	460-465,500-505
CANCEL	262	464,500-506
ALTER	263	464,465,500-506
OP	200	500-505
Spontaneous	0xx,300,301	434-436

Note: For commands appearing on cards, a separate set of error codes is provided (507-512). Since these error replies are "asynchronously" sent, and thus could cause some confusion if the user is in the process of submitting a new job after the present one, the error replies must identify which job has the faulty card(s).

TYPICAL RJE SCENARIOS

TIP USER WANTING HOT CARD READER TO HOSTX

1. TIP user opens TELNET connection to HOSTX socket 5

2. Commands sent over TELNET to RJE

```
USER=myself
PASS=dorwssap
OUT=H70002
INPUT=H50003
```

3. RJE-server connects to the TIP's device 5 and begins reading. When end-of-job card is recognized, the job is queued to run. The connection to the card reader is still open for more input as another job.
4. The first job finishes. A connection to the TIP's device " is established by RJE-server and the output is sent as an NVT stream.
5. Continue at any time with another deck at step 3.

TIP WITH JOB-AT-A-TIME CARD READER

1. thru 4) the same but User closes Reader after the deck
2. The output finishes and the printer connection closes.
3. INPUT may be typed any time after step 3 finishes and another job will be entered starting at 3.

REMOTE Job Entry Protocol
(Oct. 16, 1972)
RFC 407 NIC 12112

HOSTA USER RUNS JOB AT HOSTC, INPUT FROM HOSTB

1. User TELNET connects to HOSTC socket 5 for RJE

```
USER=roundabout
PASS=aaabbcbc
OUTUSER=roundab1
OUT=:E/.sysprinter
OUT puncher = (S)HOSTB:NE/my.savepunch
INUSER=rounder
INPASS=x.x.x
INPUT=HOSTB:E/my.jobinput
```

2. The RJE-server has FTP retrieve the input from HOSTB using User-id of "rounder" and Password of "x.x.x" for file named "my.jobinput".
3. The job finishes. RJE-server uses FTP to send two files: the print output is sent to HOSTA in EBCDIC with ASA carriage control to file ".sysprinter" while the file known as "puncher" is sent to HOSTB in EBCDIC without carriage-control to file "my.savepunch".
4. when the outputs finish, RJE-server at HOSTC discards the print file but retains the "puncher" file.
5. The User who has signed out after job submission has gotten his output and checked his file "my.savepunch" at HOSTB. He deletes the saved copy at HOSTC by re-calling RJE at HOSTC.

```
USER=roundabout
PASS=aaabbcbc
ABORT job 123 puncher
or
CHANGE job 123 puncher = (D)
```

NETWORK GRAPHICS PROTOCOL

A Network Graphics Protocol

August 16, 1974

**Robert F. Sproull
Xerox Palo Alto Research Center**

**Elaine L. Thomas
Massachusetts Institute of Technology -- Project MAC**

Preceding page blank

(2'3)

Table of Contents

	SECTION	PAGE
I	Guide to the Document	1
II	Introduction	2
	II.1 A Typical Session	3
	II.2 A Model for the Network Graphics Protocol	4
	II.3 Positioned Text.	11
	II.4 Input Facilities	11
	II.5 Inquiry	12
	II.6 UP Implementations.	13
	II.7 Summary.	14
III	The Protocol	15
	III.1 Initial Connection Protocol.	16
	III.2 Output Protocol Formats	18
	III.3 Transformed Format	18
	III.4 Segment Control	19
	III.5 Graphical Primitives	21
	III.6 Coordinate Systems	21
	III.7 Intensity.	21
	III.8 Line Type	22
	III.9 Character Display	22
	III.10 Segment Attributes.	23
	III.11 Segment Readback.	25
	III.12 Positioned Text.	26
	III.13 Input Facilities	28

III.14	Reading the State of Input Devices	29
III.15	Input Events	30
III.16	Enabling Events.	32
III.17	Event Reports	33
III.18	Inquiry	35
III.19	Miscellaneous	38
IV	Implementation Suggestions	40
V	Op-Code Assignments and Options	41
Appendix	44
References	49

Preceding page blank

SECTION I

Guide to the Document

This report describes a network graphics protocol (NGP) developed by the Network Graphics Group, a working group of ARPA network members. We believe that the design presented here should appeal to two groups:

- Those interested in device-independent graphics systems and graphics standards. Indeed, the philosophy behind the protocol design originates in principles of graphics system design.
- Those interested in using graphics via any network or communication system. Although the design was developed for the ARPA network, we believe that it has far wider applicability.

There are three major parts to the report: (1) an introductory section that gives the goals of the protocol, the general philosophy that gave rise to the particular protocol design, and definitions for a number of terms used throughout the report; (2) the details of the protocol; and (3) some suggestions to aid implementation of the protocol.

The authors of this report are by no means the only contributors to the ideas presented here. The Network Graphics Group members, too numerous to mention, have all contributed. Especially useful ideas were provided by Jim Michener, Dan Cohen, Ira Cotton, William Newman, Ken Victor and Ed Taft.

Preceding page blank ²⁷⁹⁾

SECTION II

Introduction

Protocol proposals tend to contain so much detail that the overall intent and design considerations are lost in a pile of bits. This introduction attempts to describe the high-level considerations of the graphics protocol, in preparation for the mass of detail in section III.

The aim of a graphics protocol is to allow users with various different kinds of display hardware at different sites in a network to make use of common "graphics application programs." For example, a user may want to access the BIOMOD system at Rand, the On-Line System at UCSB or the NLS system at SRI with his or her display hardware.

One approach is a collection of "special-purpose protocols." Each application program would publish a description of the protocol needed to drive the program and to view the output (e.g. the UCSB system was so documented). The prospective user would then write a program at his site to interpret the published protocol and to drive his display (probably making use of existing graphics programming facilities at his site). This might permit a convenient division of labor between the computer executing the application program and the computer driving the display (in pursuit of true resource sharing); it might be able to achieve very smooth performance despite poor network response, etc. The disadvantage of this approach is that the user must write a new program to interface his display to each different application protocol. In addition, there is no guarantee that the protocol required by the application program can actually be implemented within the user's operating system and display hardware.

Another approach is to develop one "general-purpose protocol" that tries to provide facilities that a large number of application programs could use and that a large number of user sites could interpret. Thus one user program can be used to interface to a number of application programs. The disadvantage of this approach is that the generality may preclude adequate response through the network, or that some application programs will find the general-purpose protocol too restrictive to be used at all. In addition, the design of such a protocol is not easy: attempting to provide a common device-independent framework for driving hardware of qualitatively different capabilities may be very difficult.

The protocol proposed in this document is, of course, a general-purpose protocol. However, we anticipate that special-purpose protocols will be absolutely necessary for certain applications. In these cases, the general-purpose protocol can perhaps be used as a starting point for development of special-purpose protocols. The general-purpose protocol should be used whenever possible, however, so that separate user programs need not be written for each user site.

The network protocol considered in this document is not intended to satisfy all graphics needs, for all terminals, now and in the future. It is limited to calligraphic pictures, to moderate interaction demands, and to "best effort" attempts to generate the required graphics. Certainly the impact of video technology will increase demand for variable character sets, shading, and maybe even animation techniques. These uses are clearly too new to consider in the present protocol.

II.1. A Typical Session

At the beginning of an interactive session, a (human) user sits down at a graphics display attached to a computer on the network (the user host, UH; see Figure 1). He uses a keyboard associated with the display to communicate with the user host and to initiate a program that is dubbed the "user program," UP. This program initially performs TELNET* functions, allowing the user to establish a connection with another host on the network where a service he desires is offered (the server host, SH). The TELNET functions can be used to log in, query system status, and so forth, and eventually to initiate a graphics application program (AP) in the server.

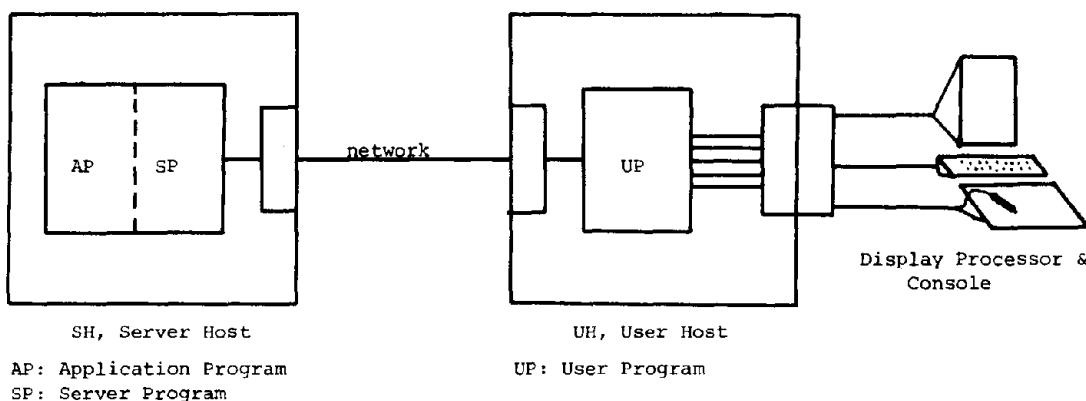


Figure 1: A network graphics session. A graphics application program running in one computer (the Server Host) drives a display console attached to another computer (the User Host).

The application program, when it wishes to produce graphical output or to request graphical input, makes use of a server program (SP) which may simply be a subroutine package. The job of the SP is to interface to the network graphics protocol on one side and to a "graphics language" on the other. (We shall use the term "graphics language" loosely here. It may just be a set of subroutine calls. The reason for making the distinction at all will appear below.)

The protocol transmitted between the SP and the UP is the **graphics protocol** described in this document. It provides facilities so that:

1. The SP can cause images to appear on the user's display screen.
2. The UP can report to the SP any interactions that the user initiates, such as keys depressed on the keyboard or stylus interactions.

* TELNET is the name of a class of programs provided by many sites on the ARPA network that allow users to log in on time-sharing or multi-access systems at other sites in the network. The term TELNET also refers to the message-transmission protocol used by the network to accomplish this function. (See L.G. Roberts and B.D. Wessler, "Computer Network Development to Achieve Resource Sharing," AFIPS SJCC Proceedings, Vol. 36, May 1970, p. 543-587.)

3. The SP can discover various special properties of the user's display terminal, and can take advantage of them in conjunction with the application program.

We shall dub these types of protocol transmission "output," "input," and "inquiry."

The job of the UP is to interpret the protocol and to generate appropriate device-dependent information so that the image shown on the user's display corresponds to that specified by the SP using the protocol. It also implements the input and inquiry aspects of the protocol. The UP may have many "local options" that are not covered by the protocol. For example, the UP might contain facilities for creating hard copies of the image on the display without engaging in any protocol exchanges.

If the display terminal is attached to a TIP*, the organization changes very little. The TIP is not the "user host," but only provides communications between the UH and the display terminal itself. In this case, the UH is often dubbed the "last intelligent host." Note that nothing prevents the UH and SH from being the very same computer.

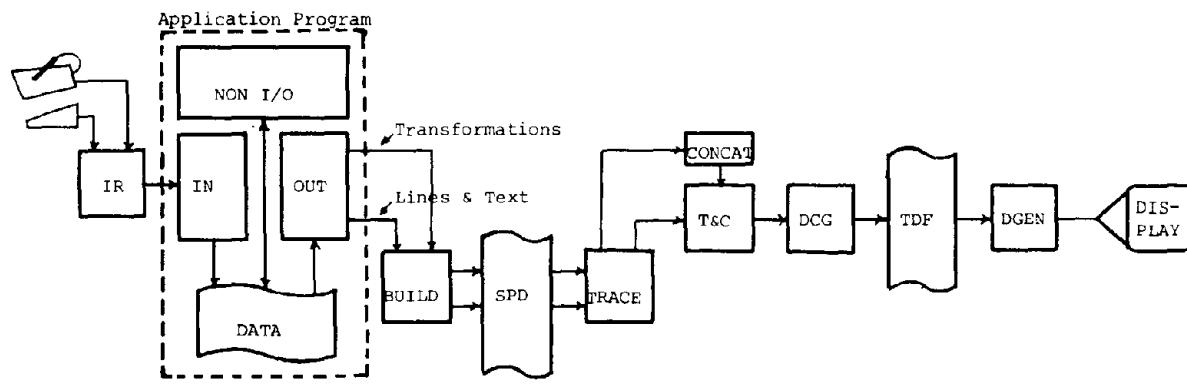
II.2. A Model for the Network Graphics Protocol

The analog of the task of defining a general-purpose graphics protocol is that of designing a "general-purpose interactive graphics system." This activity has been pursued by graphics system designers for years. Of course, these designs have been single-site designs; issues of device-independence and networking have been rarely addressed.

The philosophy behind the network graphics protocol can be demonstrated by giving a model of a general-purpose graphics system and combining it with the network model of Figure 1. The model of the system is shown in Figure 2. (The model is described in detail in [N&S] and [10GR]; we present only a brief description here.) To avoid misunderstandings, and for the sake of defining terminology, it is worthwhile to describe briefly each of the elements of Figure 2:

1. The input devices -- keyboard, stylus etc. -- are used by the operator of the application program to provide data and to control the program during execution.
2. The application data structure contains data, basically non-graphical, relating to the application program.
3. The input routines receive data from the input devices, make appropriate changes to the application data structure, and hand control to other routines.
4. The non-I/O routines analyze and modify the application data structure.

* TIP is the name of a particular kind of small computer attached to the ARPA network that implements TELNET functions for a number of dial-up communications lines. The TIP performs no significant computations for any users that dial it, but simply interfaces the network TELNET protocol to a number of terminals.

**Application Program:**

IN	Input Routines
OUT	Output Routines
NON I/O	Application Routines
DATA	Application Data Structure

Graphics Package and Hardware:

IR	Interrupt Routines
BUILD	Routines to build the SPD
SPD	Structured Picture Definition
TRACE	Routines to trace the SPD
CONCAT	Concatenation routine
T&C	Transformation and clipping routines
DCG	Display code generator
TDF	Transformed Display File
DGEN	Display generator

Figure 2: A Conceptual Model of a Graphics Application Program, a Graphics Package and Display Console.

6. The output routines build a structured picture definition from data drawn from the application data structure. Effectively they define how this data may be visualized for display purposes.

6. The structured picture definition defines the entire picture to be displayed, part or all of which may be visible on the screen. The picture definition is generally made up of a number of elements, representing parts of the picture known collectively as subpictures; subpictures may themselves be made up of other subpictures. A familiar type of subpicture is the symbol which is often used many times within a single picture or subpicture. Each reference or call to a subpicture element may denote a transformation -- scale, rotation, etc. -- to be applied to the subpicture.

7. The transformation routines apply the transformations specified in the structured picture definition and clip out information that lies

off-screen. Often an arbitrary rectangular viewport is used as the clipping boundary instead of the screen edge. These routines also handle the concatenation of transformations necessitated by multi-level structured picture definitions.

8. The transformed display file is essentially what remains of the picture after transformation and clipping. It is defined in the screen's coordinate system and is generally stored in a format that allows direct refresh or regeneration of the picture. The display file contains "primitives" that specify lines, dots and text to be displayed.

9. The display generator generally includes a vector generator and a character generator, which transform the contents of the transformed display file into signals that the display's deflection system can understand.

10. The display itself.

We shall now analyze Figure 2 to see how the system can be implemented. The diagram illustrates three information structures (an application data structure, the structured picture definition, and the transformed display file), and three processes (the output routines, the transformations, and the display generator). The design of a general-purpose graphics system requires specifying the roles of all of these elements, with the exception of the application data structure. Our examination amounts to asking: what can the hardware do, and what does the graphics programmer think he can do? (For a more careful treatment of the concepts discussed in the next few paragraphs, see [10GR].)

Most display hardware is "processor" hardware, capable of implementing some or all of the three processes in Figure 2. If, for example, a display processor is available that implements transformations and display generation, then the transformed display file is absent, and we can view the hardware as "an interpreter of structured picture definitions." If the available hardware has fewer capabilities (e.g. no transformation ability), the picture is refreshed from the transformed display file, and the hardware is "an interpreter of transformed picture definitions." Or, if the hardware is a storage-tube terminal that does not require a display file for refreshing purposes, part of the display generation process is a software process. (However, a display file is still required, as we shall see below.)

Determination of what the programmer can do is somewhat more subtle, because the graphics system designer has complete control of the facilities he presents to the programmer. For example, the programmer of the system shown in Figure 2 would think he was creating a structured picture definition: the output routines allow him to create, modify, and delete elements of that structure. The remaining processes (transformation and display generation) and structures (transformed display file, if it exists) are inaccessible to the programmer; in some sense, he thinks that a "display processor" is interpreting the structured picture definition in order to generate a display. He cannot determine whether a transformed display file exists, nor whether transformations are done in hardware or software, etc.

If the structured picture definition is deleted, the programmer sees a quite different set of facilities. The output routines create (after transformation) a transformed display file. The programmer now thinks that the hardware is a "transformed display file interpreter." Again, the programmer is unaware of the

details of the display generator process (for example, if the display is a storage tube terminal, the display generator is a combination of a software display-file interpreter and some hardware vector and character generators. If, on the other hand, the transformed display file is used to refresh a display, the display generator is presumably entirely in hardware).

The discussion suggests that relative device independence can be provided if we permit two different kinds of output information: (1) information for building structured picture definitions, or (2) information for building transformed picture definitions directly. The first of these is matched to high-performance displays such as the LDS-2 or LDS-1; the second to standard refresh displays such as the IMLAC or GT-40.

How does this relate to network protocol? We can essentially view the UP as a "display processor," i.e. an "interpreter of structured picture definitions," or an "interpreter of transformed picture definitions." The network protocol is used to build and modify a picture definition contained in the user host. Various UP options are shown in Figure 3 (the dotted lines surround those processes that are implemented with display processor hardware). Figures 3a and 3b show the network used to transmit transformed information; the UP uses this information to build a display file for refreshing a display or for updating a storage-tube. Figures 3c, 3d and 3e show the network being used to transmit information for building a structured picture definition; the UP is an interpreter of a structured display file.

Figure 4 illustrates some possibilities for the server; the server can generate either structured or transformed display information. In Figures 4a and 4b the programmer "sees" a structured picture definition; in Figure 4c a transformed picture definition.

In the protocol, the UP tells the SP which kinds of format it can implement. It is perfectly possible for the UP to implement both -- the display images due to each of the formats are merged onto the screen.

The two different kinds of output format can be summarized as follows:

Transformed

The protocol is used to build and modify a set of "segments" (sometimes called "records") of a transformed display file, stored in the user host. A segment is a list of graphical primitives that specify lines, dots and text to be displayed at specific positions on the display screen. Individual segments may be deleted or replaced; they may be added to or removed from a list of segments to actually display on the screen (this is called "posting" and "unposting" segments). If a picture is composed of many segments, changes to the picture can often be made by replacing one or two segments; thus segmenting the display file helps to reduce the amount of information that must be transmitted through the network to effect a change. Considerable experience with this type of picture definition has demonstrated that device independence can easily be achieved [Omnigraph and Omnigraph.brief].

One advantage of transformed format is that the UP can be kept very simple; no transformations need be performed in the UH. A UP along the lines of Figure 3a should be able to be implemented in an IMLAC. The burden of transformation is left to the SH, presumably a large computer quite capable of being programmed to do transformations.

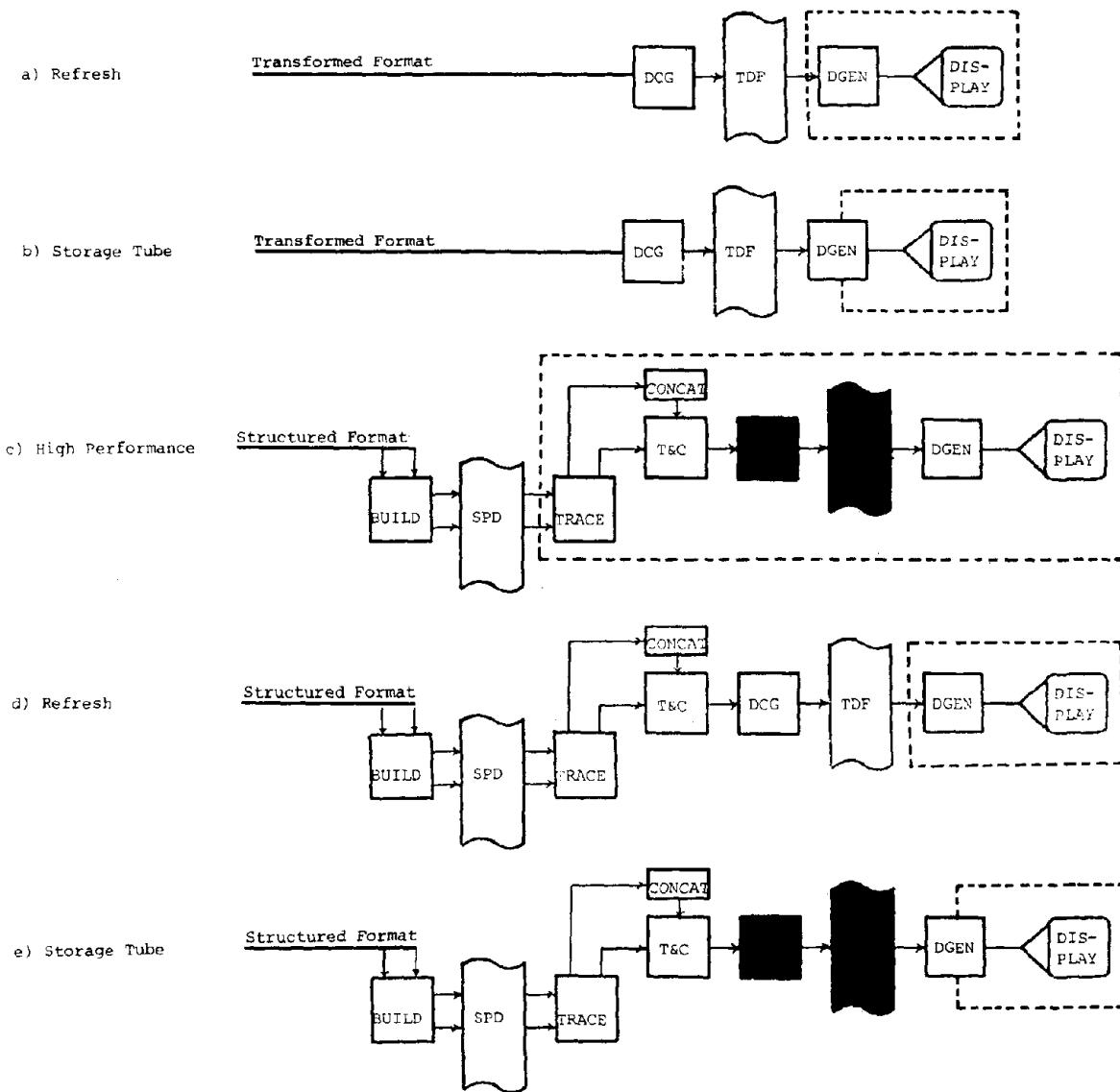


Figure 3: Various Configurations for the UP. Dashed lines surround processes implemented with hardware. Blackened processes are omitted in the particular configuration.

Structured

The protocol is used to build and modify "figures;" each figure is a collection of "units." A unit may be a list of graphical primitives, such as lines, dots and text; or it may specify a "call" on another figure, together with a transformation to apply to the called figure. The protocol can replace individual units; altering a figure that is called in several places may cause widespread changes to the visible display. The structured format requires (in principle) even less network bandwidth for updates than does the transformed format; many updates may involve changing a

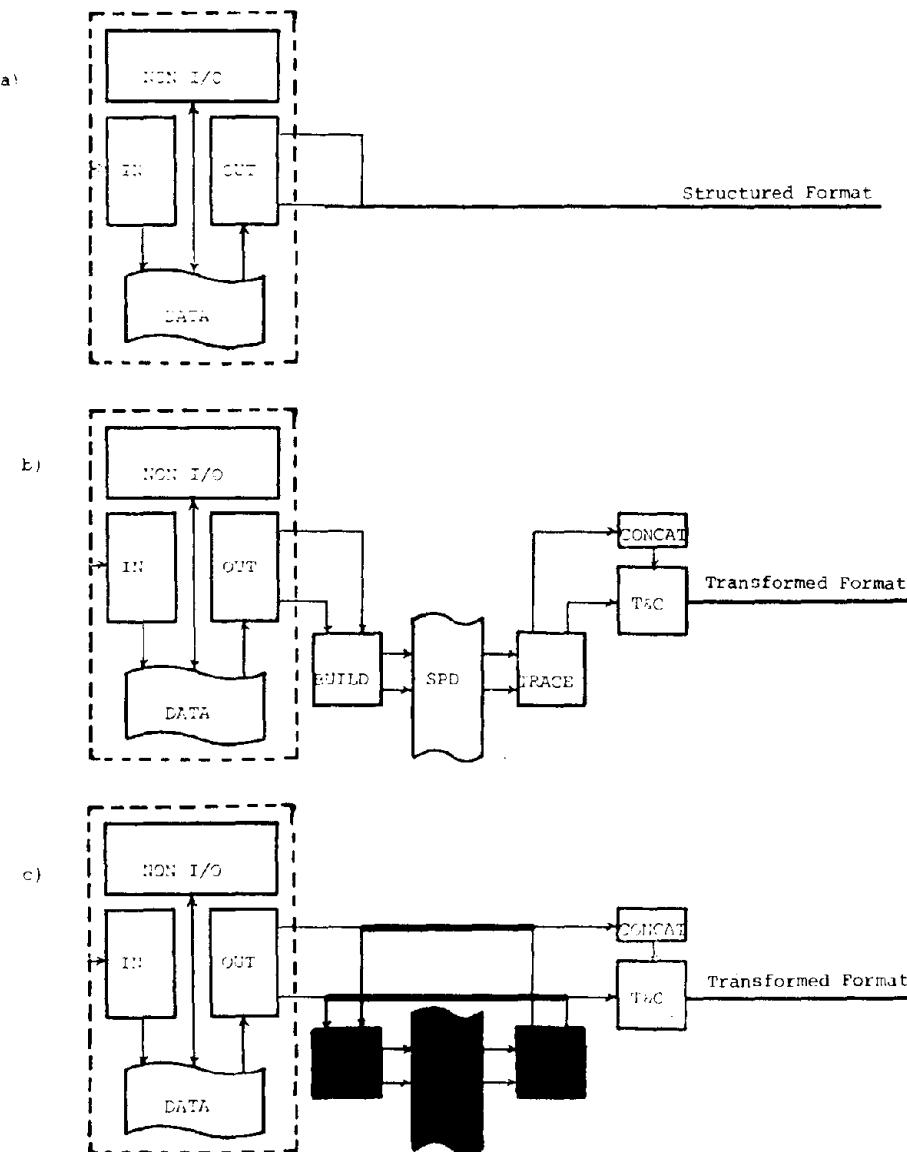


Figure 4: Various Configurations for the SP. These mate with the UP configurations of Figure 3.

single transformation (e.g. to change the viewing transformation for a three-dimensional object).

Although a structured picture definition can be interpreted directly by a few display processors that have transformation ability, most UP's that implement structured format will perform the transformation in software (Figure 3d,e; [10GR]). This implementation is relatively difficult, and certainly requires a fairly powerful UU computer.

The kind of network protocol (structured or transformed) should be clearly distinguished from the graphics language and the "output routines" of Figure 2. For example, the application data structure might be elaborately structured (e.g.

a circuit diagram, with instances of flip-flops and gates, etc.), and the programmer may think of the display generation in a structured way even though the network traffic is transformed (i.e. unstructured). This effect can be achieved with the display procedure technique.

Another way of looking at this is as follows: if we view the UP as a "display processor," then we might view the SP as a "graphics package" for driving the display processor. Most graphics packages attempt to insulate the programmer from the vagaries of a particular display processor, to provide structure even though the display processor does not, etc. In other words, the application programmer may have a structured view of the world even though the UP cannot provide such a view.

We have glossed over a significant problem introduced when some of the processes of Figure 2 are implemented in software. If the programmer thinks he is producing a structured picture definition, but the hardware is interpreting a transformed picture definition, when is the transformation software run? A similar problem occurs in Figure 3b: when is the display-generation software run in order to update the display? One answer is: whenever a display file is changed, necessary software processes are invoked to update the display. This has several bad effects. A more useful technique is for the SP (and application program) to signal the UP whenever a collection (or batch) of changes is complete and the display should be updated. This technique has the following advantages:

1. Transformation software (e.g. 3d, 3e) is executed only when necessary in order to update the display; we never needlessly repeat transformations.
2. Screen erasures on storage tubes (e.g. 3b) can be minimized -- we need erase the screen a maximum of once per batch of updates, rather once per update. See [Omnigraph.brief].
3. All changes to the display appear "instantly." Network speed means that display-file updates will arrive at the UP over a longish period of time. If the effect of these changes is delayed until a batch is finished, the display will appear to change "all at once," a much more satisfying effect than many slow changes. This is particularly true if an image is being replaced by another image that is a scaled-up version of the original: some things grow before others, and the display passes through a number of nonsensical states.

To take full advantage of this technique, the AP should specify when the screen should be updated to represent precisely what is specified in the display file. These "end batch of updates" commands should probably precede each request for new user input -- thus the user will see an up-to-date image before formulating his response. Specifying this information is quite easily done, and is not at all unnatural for the application programmer.

If this policy of delaying changes is not to a programmer's liking, the SP could be instructed by the AP to issue an "end batch of updates" command following each and every update to a segment. Updates only occur as a result of a small number of protocol commands (see section III); this should not be difficult.

II.3. Positioned Text

The graphics facilities already described can be used to put text information on the screen. However, certain application programs display exclusively text (e.g. NLS [NLS]) and require a rather different set of facilities for controlling the display. Strings of text are "positioned" on a display screen and "edited" by commands from the server host.

The positioned text facilities have been separated from the graphics facilities for two reasons: (1) users with simple alphanumeric terminals (e.g. Hazeltine) can in fact implement the positioned text protocol even though they cannot implement the full graphics protocol; (2) It simplifies the design of user programs that only need to provide text interfaces (e.g. allows one to build a UP in an IMLAC that is optimized for NLS use).

This graphical output format is completely independent of the transformed and structured formats. A UP may report to the SP that it implements *only* the positioned text format -- this might be the case of a UP for NLS use.

II.4. Input Facilities

The problem of providing input facilities is even harder than that of providing output facilities. The difficulties are chiefly those of device independence and of adequate performance. The device independence issue is easily demonstrated: display hardware can have a large number of very different kinds of input gadgets attached (light pens, tablet and stylus, joysticks, knobs, buttons, etc.) that have different properties and different methods of reporting their output (e.g. periodically, on computer demand, or "when something changes"). In addition, operating systems at user sites often enforce restrictions on the use of input equipment in order to avoid undue system degradation.

The problem of performance is nicely demonstrated by Figure 1 -- If each input must be shipped to the server host, processed by the AP and SP, then any display updates shipped to the user host and processed by the UP before the user sees the response, it would be impossible to use many interactive graphical techniques.

The protocol attacks these problems in simple and probably inadequate ways. For this reason, the input facilities are the most controversial and experimental of the protocol.

The device independence issue is solved by inquiry: the UP reports to the SP a list of available devices. The SP and AP can then collaboratively arrive at an acceptable set needed for operating the AP. If the set of input devices is insufficient, the AP can perhaps engage in a dialog with the (human) user to seek remedies. Perhaps a spare device can be plugged in; perhaps another version of the UP can be run which implements the required device. Or, if the AP and SP are sufficiently flexible, perhaps the command language of the application program can be altered dynamically to permit its operation with the available devices. For example, if the UP responds that it has no coordinate input device (and that it is not willing to simulate one with, say, two knobs) then the AP might want to use a keyboard-based interaction sequence and to organize the entire command system differently.

The performance difficulties are addressed by permitting the SP to ask the UP to

use a particular interactive technique in conjunction with an input device, and to report the results of the interaction. We shall term such interaction techniques events. The techniques often involve providing "local feedback," so that the user sees the results of his interaction without a long network delay. Examples are: displaying a "tracking dot" at the current location of a coordinate input device, or displaying a trail of "ink" behind the tracking dot, etc. Examples of the special events that the protocol provides are:

Positioning -- Using a coordinate (or other) device to provide a pair of coordinates to specify the position for something. The UP will usually display a tracking dot to aid the user in coordinating his input with the display.

Pointing -- Using a coordinate (or other) device to identify an object currently being displayed on the screen. Again, the UP will probably provide tracking. There are two ways of providing this technique: one is to assume that the display terminal has some hardware feature that aids identifying a graphical feature being pointed at (e.g. light pen or comparator). However, the same information can be deduced from a positioning interaction and some software calculation (either in the SP or UP) to determine what object is being identified. Thus, even if a UP cannot provide the "pointing" interaction, the SP may be able to (see [N&S] for a description of the process).

Stroke -- Using a coordinate (or other) device to trace out a free-form curve and reporting a stream of coordinate points on the curve. The UP will provide tracking and leave a trail of dots ("ink") along the curve.

Dragging -- Using a coordinate (or other) device to cause some portion of the display image to move in synchronism with the coordinate device. This technique could be classed as "highly interactive," and some display terminals cannot provide it.

The protocol provides the SP with two basic methods for dealing with input devices: (1) to request and obtain the state of an input device, e.g. the current position of a coordinate input device, and (2) to enable various events, and to obtain a "report" describing the events resulting from user actions.

The user site has a wide latitude in implementing these input facilities. Since Inquiry is used to find out what devices and events the user site implements, the site may implement as many or as few as it likes. The latitude permits individual users to use devices differently or to establish special feedback mechanisms (e.g. a number displayed on the screen that represents the current reading of a knob). It also permits user sites with weird hardware or operating systems to emulate input devices in any way they choose. (For example, no requirements are put on sampling rates for inked strokes; something "reasonable and proper" is adequate.)

II.5. Inquiry

The protocol has no set of "standard" features; there is thus no standard graphics terminal as viewed by the protocol. The Inquiry function is used to transmit to the SP a certain amount of detailed information about the terminal in use and the UP that drives it. This information is a "constant" that will probably be requested by the SP when it initiates a graphics session.

The information transmitted by the Inquiry response is in part for information only. However, some of the information returned is essential in order for the SP to transmit legal protocol to the UP. In outline, the information returned is:

List of implemented protocol commands. This list tells the SP whether the UP implements transformed format, or structured format, or positioned text, or any combination of them, and so forth. In addition, this report tells which optional parts of the protocol are implemented by the UP.

Coordinate Information. This information is necessary for the SP to carry out transformations that generate coordinates in the coordinate system used by the terminal.

Parameters that describe available character sizes, available intensity resolution, available line textures, etc.

A list of available input devices and events. A "device number" is specified for each device; this is used when reading the state of a device. Similarly, an "event number" is specified for each event, and is cited when enabling or disabling it.

An ASCII text string that describes the terminal, e.g. "IMLAC PDS-1 in room 22."

The information in the Inquiry response (that transmitted from UP to SP) that is not essential to further protocol operation may still be useful to the SP in order to drive the terminal intelligently. For example, inquiry can determine the kind of display being used, not so as to send device-specific code to it, but so that the AP does not try to use a graphic technique on a terminal that cannot handle it (e.g. some sort of dynamics on a storage tube).

II.6. UP Implementations

Although the description of the protocol is quite lengthy, the protocol itself is quite simple. The aim of the design is that the UP could be implemented in an IMLAC or GT-40 or similar "smart" terminal. Of course, it could also be implemented on any host computer, with a less smart terminal attached to the host.

There are three main mechanisms for simplifying the implementation: (1) the Inquiry function specifies many of the terminal details to the SP, thus freeing the UP from coping with complicated logic to implement complicated operations; (2) much of the protocol is optional, a subset being quite adequate for most applications; (3) in thorny areas (e.g., input protocol), the protocol is deliberately vague, allowing the UP implementation considerable latitude to obey the protocol as best it can.

The philosophy of "making the SP drive the terminal," rather than making the UP achieve an ideal performance is key. This approach puts ingenious graphics programming and command languages where they belong, in the server.

II.7. Summary

Here is a brief summary of the main philosophical points of the protocol:

- O. The protocol is based on the premise that much, but not all, graphics can be done within a "general-purpose" framework. Special-purpose protocols are inevitable.
1. The protocol facilities for graphical output can be likened to those of a graphics system driving a display processor. The protocol creates and modifies a display file at the user site.
2. The protocol provides options: the SP and UP must agree on what kind of "display processor" the UP can implement (Structured, Transformed, Positioned Text, or some combination) and on selection of input devices. The protocol thus implements no fixed "virtual display" but rather a large variety of display processors.
3. Portions of the protocol are left deliberately vague. The user program is expected to implement features to the best of its ability; if the implementation is inadequate, the shortcomings will probably be quickly discovered by a user.
4. Although the protocol appears largely "device Independent," Inquiry functions permit the application program to discover many hardware details necessary to drive the display intelligently.

SECTION III

The Protocol

This section presents details of the graphics protocol. The topics covered are the connection protocol, the graphical output protocols, the graphical input protocol and the inquiry protocol.

The section describes network traffic as a series of commands and operands, all expressed in a common notation. The smallest unit of traffic is an 8-bit byte. The construct <...> refers to a specific byte, i.e. a command that is assigned a particular op-code (listed in the appendix). Constructs of the form <*...*> refer to sequences of bytes that are defined elsewhere in this document.

Following are some standard definitions:

All numbers in this document are decimal unless preceded by an apostrophe, in which case they are octal (8='10).

A <small.integer> is one 8-bit byte that contains the integer (range 0 to 255).

A <large.integer> is two 8-bit bytes that together comprise a 16-bit integer. The first byte transmitted is the high-order 8 bits; the second the low-order 8 bits (range 0 to '177777).

A <small.fraction> is a two's complement fraction in the range [-1:1-1/128]. It is defined as (<small.integer>-128)/128.

A <large.fraction> is a two's complement fraction in the range [-1:1-1/32768]. It is defined as (<large.integer>-32768)/32768.

A <count> is either one or two 8-bit bytes, depending on the size of the count. If the count is less than or equal to 127, then <count> is simply one byte that contains the count; otherwise it is two bytes, and the count is computed as (byte1-128)*256+byte2.

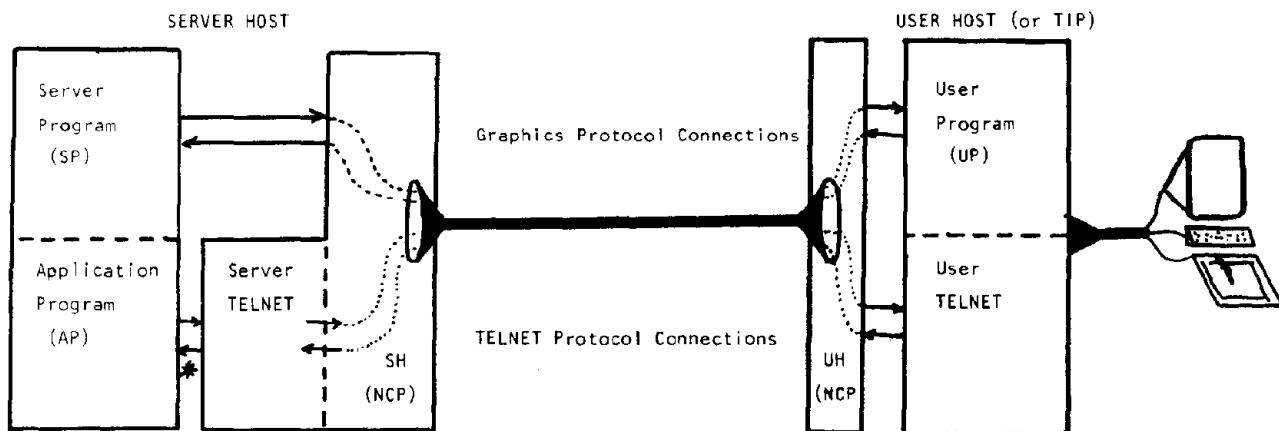
A text string <text> is defined as a character count, <count>, followed by that number of 8-bit bytes. If the text string is intended to be interpreted as ASCII text, then the following conventions are observed: (1) every printing character in the ASCII set is in the set (high-order bit is zero), (2) the ASCII control characters carriage return ('15), line feed ('12), tab ('11) and formfeed ('14) may appear; (3) codes in the range '200 to '377 may be used for whatever purposes user and server desire, but the protocol establishes no conventional meanings.

The various forms of picture definitions in the user host are given names (e.g. <seg.name>, <ptext.name>). These are all defined as 16-bit quantities, in <large.integer> format. The name spaces are all separate.

III.1. Initial Connection Protocol

This section describes the mechanism for establishing connections between the SP and UP in the ARPA network. An understanding of this section is not required in order to understand the remainder of the protocol.

A user session with a graphics application program (see Figure 5, an elaboration of Figure 1) has many close analogies to a TELNET session with a program: the user must log into a server system, execute several system commands, initiate the program, communicate with the program as it is running, perhaps interrupting a running program, logging out, etc. A graphics session certainly requires all of these features; in addition it will require a pathway for transmitting graphics protocol information in both directions.



In the case of an intelligent terminal attached to a TIP, the boxes labeled UH(NCP) and User TELNET are implemented by the TIP; the box labeled UP would be implemented in the intelligent terminal.

* This communication path is accomplished with operating-system calls that type characters on the controlling terminal and that accept characters from that terminal. (That is, this is not a direct network connection.)

Figure 5: A Network Graphics Connection and Associated Processes

The graphics protocol traffic is inherently separate from the TELNET traffic, and should use a separate network connection. It is inconvenient to multiplex graphics protocol on the TELNET connections because many operating systems give AP's access to the TELNET connection only via the mechanisms that they use to access a controlling terminal: the application program "types" to the

terminal. In addition, the TELNET connection is used by some operating systems when signaling errors; this summary use might interfere with a multiplexing scheme. Thus, the protocol requires a pair of network connections dedicated to graphics traffic.

This approach can also accomodate intelligent terminals (e.g. IMLAC's) attached to TIP's. If the IMLAC program interprets (1) ASCII text and (2) graphics protocol, and uses a simple multiplexing scheme to transmit these two kinds of traffic to and from the TIP, then the TIP can support the traffic with minor modification. The TIP need only be able to establish the extra pair of connections and to multiplex the two kinds of traffic for consumption by the intelligent terminal. (This is *not* the same as multiplexing the TELNET connection.)

There are two connection schemes, one to be used for now, and one that is intended for use when the new style TELNET protocol is fully operational and when a sufficient number of operating systems have been modified to give the UP and SP access to the negotiation mechanism.

For now.

When a graphics SP is started (usually by the user's issuing an appropriate command to the user host through the TELNET connection), and wishes to initiate a graphics dialog, it gets a pair of complementary socket numbers from its operating system. These are called SP-send and SP-receive. The SP then "types" over its TELNET connection an ASCII string that consists of 17 characters: the first six characters are *GICP*; the remaining 11 characters are the 11 octal digits (in ASCII, of course) of the socket number for SP-receive. Note that SP-receive is an even number, and that SP-send = (SP-receive)+1. (This is a network convention that seems quite nice for now.) After typing this information, the SP does "listens" on those socket numbers.

The UP recognizes the special character string and following digits, and issues RFC's (requests for connection) from two of its sockets (UP-send and UP-receive) to the complementary sockets at the SP. The SP will return the RFC's, thus completing the connections. The dust has settled, and the connections UP-send=>SP-receive and SP-send=>UP-receive are ready for use.

Ultimately.

When reason descends on the world, the TELNET negotiation mechanism (see NIC 15372) will be used to establish the willingness to transmit graphics information (DO, DONT, WILL, WONT GRAPHICS), and a subnegotiation transmission will transmit the socket number. In the case of an intelligent graphics terminal attached to a TIP, the TIP TELNET responds to the negotiation and establishes the connections.

Even after graphics protocol has been initiated, not all communications between SP and UP will be graphics protocol: there still may be TELNET traffic. From UP to SP will go "breaks;" from SP to UP will go text "typed" by the application program (rather than enclosed in some graphical command for displaying text) as well as system error or informational messages. Such SP-to-UP text is called "unescorted text." The user will probably wish to read this text, since it is often important for understanding or operating the AP. The text can be handled by the UP in either or both of:

(295)

1. Show it on the display screen. This option is controlled by the graphics protocol (see positioned text).
2. Local (UP) option. This treatment is up to the UP: the text can be displayed regardless of provisions of method 1; it can be typed on an adjacent hard-copy terminal, or whatever.

III.2. *Output Protocol Formats*

The network graphics protocol makes provision for three kinds of formats for graphical output:

- 1: Transformed format.
- 2: Structured format.
- 3: Positioned text format.

It is possible to design a UP that can correctly interpret any combination of formats coming from the SP. The UP reports to the SP, via the inquiry response, which formats are implemented (this information is contained in the list of implemented commands).

The design of the structured format is still in preliminary stages. This is chiefly because of difficulties designing a suitably device-independent view of transformations (e.g., clipping and rotation conflicts, providing for the constraints of transformations performed with analog hardware). A brief description of the preliminary design appears in the appendix.

III.3. *Transformed Format*

The protocol for "transformed format" is used to build and modify a set of segments of the picture definition stored in the UH. All coordinate transformations are performed in the SP prior to sending data to the UP; the segmented picture definition thus contains descriptions of lines, dots and text that will have a fixed location on the screen.

A segment is created in the following fashion. The "open segment" command is sent to the UP, together with a "name" for the segment. Any subsequent graphical primitives (e.g. line, dot, text) sent to the UP are added, in order received, to the currently open segment. The creation process is terminated by a "close segment" command. The segment now specifies how to draw some (or all) of the desired display image.

The creation of a segment simply specifies a list of graphical primitives and not the use to which they are put. If the segment is to be displayed, a "post" command specifies that a segment is to be added to a list of segments to be displayed. The "unpost" command removes a segment from the display list. The "kill" command is used to destroy the segment altogether.

No changes to the visible display are made until the "end batch of updates" command is received at the UP. Thus, the effects of the "post," "unpost," and "kill" commands must be delayed until this command is received.

Although the graphical primitives that are contained in a segment cannot be

modified, a certain number of "attributes" associated with each segment may be individually modified. These facilities are described more fully below.

III.4. Segment Control

A detailed description of the commands follows:

<seg.open> <*seg.name*>

This command opens a new segment and specifies its name. All subsequent graphical primitives will be added, in order received, to the open segment. Graphical primitives need not follow contiguously -- other graphics protocol commands may intervene between specification of primitives to be added to the segment. If a segment with the same name already exists, it is not destroyed at this point (this technique is called "superceding" a segment; the protocol insists that the segment being created is double-buffered).

Immediately after the **<seg.open> <*seg.name*>**, any attributes that are to be associated with the new segment must be set, before the first graphical primitive is specified. This operation indicates to the UP which attributes of this segment might be changed later on. Such attribute settings are accomplished with the **<*attribute*>** sequence, described in section III.10. The reason for this convention is that the UP may wish to build the display file in a slightly different way if certain attributes are specified, so that they may later be changed.

<seg.close>

This command signals the end of generation of (or appending to) the currently open segment. The segment can now be posted, unposted, or killed.

<seg.post> <*seg.name*>

The specified segment is added to the list of segments to be displayed. If the named segment is the currently open segment, it is "closed" first. No change is made to the currently visible display.

<seg.unpost> <*seg.name*>

The specified segment is removed from the display list. Again, no change is made to the currently visible display.

<seg.kill> <*seg.name*>

The specified segment is deleted entirely. If the segment is currently in the display list, it is "unposted" first. Note that this means deletion may be delayed so as not to alter the currently visible display until the "end batch of updates" command arrives.

<seg.append> <*seg.name*>

This command specifies that all subsequent graphical primitives are to be added to the end of the segment named, which must already exist. Note,

however, that even if the segment named is currently being displayed, the appended information cannot be displayed until the next "end batch of updates" command. A segment that is appended to leaves attribute settings unchanged. In particular, the set of available attributes cannot be augmented beyond those specified originally following the <seg.open> command.

The "append" feature is entirely optional; if the UP implementation does not permit appending, the inquiry response will so specify. Failing to implement this command has no effect on the interpretation of the rest of the commands.

<end.batch.of.updates>

This command specifies that a collection of updates is complete, and the visible display should be updated to reflect the changes. In particular:

- Any killed segments are entirely deleted and returned to free storage.
- The old versions of any superceded segments are deleted and replaced by the new versions.
- Any "posts" or "unposts" transmitted since the last "end batch of updates" can be performed.
- Any "appends" specified are actually added to the appropriate segment.

(If the display file is not used to refresh a display, as is the case with a storage tube terminal, the modifications to the display file structure itself need not be delayed until the <end.batch.of.updates> command arrives, but all screen changes must be delayed. This minimizes the number of full-screen erasures required on storage tubes.)

Some application programs may wish to cause changes to appear as soon as the change has been successfully transmitted to the UH (e.g. when a <seg.post> is sent). In this case, the AP or SP can simply arrange to follow each such modification command with a <end.batch.of.updates> command.

One drawback of delaying changes is that up to twice the amount of display-file storage can be consumed, compared to that required to store one picture. This will happen if a batch of changes involves superseding every segment. This might cause the UP to exhaust the free storage available to it for display files. If this happens, the UP may simulate the effect of an <end.batch.of.updates> command prematurely, and thus reclaim storage used for superseded segments. This should really be considered an error.

A number of anomalous or "illegal" sequences of the segment-controlling commands might occur. Specific remedies are described below. (A UP implementation is not required to follow these conventions, and SP implementations should not count on them.)

1. If graphical primitives are received by the UP when no segment has been opened (either by <seg.open> or <seg.append>) they are discarded. The UP might issue an error indication.

2. If a <seg.open> or <seg.append> is received when a segment is already open, the newly-received command is ignored. Again, the UP might issue an error.
3. If the segment named by a <seg.append> does not exist, the command should be treated as a <seg.open>.
4. If the segment named in a <seg.kill>, <seg.post> or <seg.unpost> does not exist, the command is ignored.
5. <end.batch.of.updates> may occur anywhere, even while a segment is open. Primitives added to the currently-open segment should not, however, be displayed (because the segment is being created, and is not yet finished!).
6. If the <seg.kill> or <seg.unpost> commands give a name that matches that of the currently open segment, the command refers to the old version of the segment (if any), not to the one currently open.

III.5. Graphical Primitives

The following commands cause primitives to be added to the currently open segment:

```
<seg.dot> <"x.s.coord"> <"y.s.coord">
<seg.move> <"x.s.coord"> <"y.s.coord">
<seg.draw> <"x.s.coord"> <"y.s.coord">
<seg.text> <"text!">
```

These primitives are the familiar commands for adding points, lines and text to the open segment. No relative mode is provided: the SP can easily let the application program specify relative information, and convert to absolute for transmission.

III.6. Coordinate Systems

The coordinate system used for arguments to dot, move and draw in the transformed format is called the "screen coordinate system." *The format of a <" .s.coord"> construct is determined from the inquiry response, and is in the coordinate system actually used by the graphics terminal.* The Inquiry response defines how many 8-bit bytes are used to specify such a coordinate, and what values correspond to the left, right, bottom and top addressable points on the screen. (For a discussion of other possibilities for the screen coordinate system, see [NIC 19933].) See section III.16, Item 2 for an example of the screen coordinate calculations.

III.7. Intensity

The SP can select an intensity that is to be used for all subsequent graphical primitives added to segments (except as noted below). The command

(299)

<set.intensity> <*count*>

specifies the intensity in <*count*> format. Permissible values of the <*count*> are returned in the inquiry response. Exception: If intensity is specified as an attribute of a segment, that value overrides any that is specified within the segment. (See the section on attributes, below. One of the reasons for declaring the intention to change attributes when opening a segment is so that the UP can generate the segment in such a way that the attribute may be implemented efficiently on the display hardware.) A default intensity (anything visible) is established by the UP initially. *

III.8. Line Type

The SP can govern the type of line added to segments by the draw primitive. The sequence

<set.type> <*count*>

sets the line type for all subsequent lines. The inquiry function can be used to find out how many distinct types (e.g., dotted, dashed) are supported by the UP. Type 0 is always a normal solid vector, and is established as the default type by the UP initially. The protocol makes no precise definition of line types.

III.9. Character Display

The text primitive adds alphanumeric information to the open segment; most terminals will have hardware character generators for actually drawing the characters. However, the SP can control certain aspects of character display: size and orientation. There are two methods of specifying the size or orientation: (1) the SP may select one of several discrete character sizes and orientations available (details about these sizes are contained in the inquiry response), or (2) may select an exact size. All subsequent text primitives (up to the next time the SP asks to change) use that style.

Initially, the UP sets a default size (any legible size) and orientation (horizontal). The UP implementation should try to be resilient when the SP generates text that lies off the screen.

The orientation is selected by the sequence:

<set.character.orientation.discrete> <*count*>

where <*count*> specifies one of several discrete orientations available, as specified in the inquiry response. Alternatively, the following (optional) command may be used:

* All that is needed to implement this facility in the UP is one variable that maintains the "current intensity." Whenever a line, dot or text string is added to an open segment, the value of the variable specifies the intensity. A similar method is applicable for line type, character size, and character orientation.

`<set.character.orientation.continuous> <"large.fraction">`

where the fraction is a measure of the angle between the text drawing direction and the horizontal as fractions of 2π . The fraction for horizontal direction is 0, for vertical text running up is $1/4$, etc. If this command is implemented, the UP agrees to draw characters at any orientation.

The size is selected by one of two methods. A specific discrete size, the details of which are returned in the inquiry response, is set with

`<set.character.size.discrete> <"count">`

where the count is the index of the character size returned by the inquiry response.

The following command can be used to specify a "continuous" character size. This command is optional; if it is implemented, the UP agrees to display characters of any size:

`<set.character.size.continuous> <"char.size.description">`

where `<"char.size.description">` is

`<"BW.s.coord"><"BH.s.coord">
<"CW.s.coord"><"CH.s.coord"><"CT.s.coord">`

The coordinates specify five dimensions, in screen coordinates, of the character, as shown in Figure 6.

III.10. Segment Attributes

Although the graphical primitives that make up a segment may not be modified, a small number of "attributes" may be. This section describes the attributes and the mechanisms for changing them.

Just after the `<seg.open>` command is transmitted, any number of `<"attribute">`s may be specified: these are attributes that are to be associated with the new segment. At some later time, the `<change.attribute>` command can be used to change any attribute that was specified in this original list. The reason for the initial list is that the UP may wish to generate the display file for a segment differently if it knows that certain attributes might later be changed.

The attributes are individually optional. If the the UP does not implement the `<set.xxxx.attribute>` command, then it has no facilities for dealing with the corresponding attribute.

Attributes may be changed by the sequence:

`<change.attribute> <"seg.name"> <"attribute">`

Such changes do not take effect immediately; an `<end.batch.of.updates>` command causes changes.

The possible `<"attribute">` sequences are:

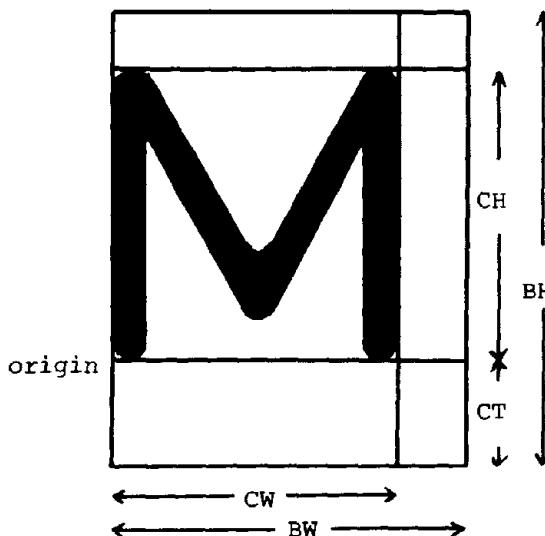


Figure 6: A Character Size Description. All measurements are made in the screen coordinate system. The point labeled "origin" is the reference point for the character, i.e. it coincides with the (x,y) point set by the <seg.move>, <seg.dot> or <seg.draw> previous to the <seg.text>.

Highlighting. This is an on/off condition for highlighting (e.g. blinking or intensifying unnaturally) an entire segment. The relevant <*attribute*> sequence is either

<set.highlight.attribute> <on> or
 <set.highlight.attribute> <off>

The default is <off>.

Hit sensitivity. This on/off attribute is used in conjunction with input facilities (see III.13). If a segment is hit sensitive, then its name may be reported to the SP if a coordinate input device is pointed at any line, dot or text that is part of the segment. The <*attribute*> sequence is either

<set.hit.sensitivity.attribute> <on> or
 <set.hit.sensitivity.attribute> <off>

Default is <off>.

Intensity. This attribute controls the intensity of all graphical primitives in the entire segment. It overrides any <set.intensity> settings within the segment.

<set.intensity.attribute> <*count*>

The permissible values of <*count*> are returned in the inquiry response. Default is something visible.

The intensity attribute and <set.intensity> are mutually exclusive within a segment: a segment created with an intensity attribute will ignore any <set.intensity> commands; one created without an intensity attribute will honor all <set.intensity> commands.

Screen select. If a user site has several display screens, the SP can control which screens a particular segment is to be viewed on (see Inquiry section for a description of how the SP can discover the number of

screens in use at the user site). Screens are enabled by a mask byte, with a bit for each of up to eight display screens (the high-order bit of the byte is the first display, the next the second, etc.). If the bit is on, 'posting' the segment will cause the segment to appear on the corresponding screen. The default is '200 (screen number 1 on).

`<set.screen.select.attribute> <"small.integer">`

Initial position. This command sets the position on the screen of the first item in the segment. This indicates the intention of the SP to either cause dragging to be performed, or to change the position of the segment at some future time.

`<set.position.attribute> <"x.s.coord"> <"y.s.coord">`

As an example: `<seg.open> <seq.name=2> <set.position.attribute> <coordinate=500> <coordinate=500> <seg.move> <coordinate=400> <coordinate=500> <seg.draw> <coordinate=600> <coordinate=500> <seg.post> <seg.name=2> <end.batch.of.updates>` causes a segment with one visible line to be created. The line extends 100 units left of the initial position, and 100 units to the right. If we later wish to change the position, the command `<change.attribute> <seg.name=2> <set.position.attribute> <coordinate=200> <coordinate=500>` will move the line 300 units to the left. Note that some UP's will wish to store segments that are to be repositioned in an internal format that is different from that for other segments (e.g., as a sequence of relative moves and vectors).

III.11. Segment Readback

This section describes facilities for transmitting information about a segment stored in the UP back to the SP. This allows the SP to save "copies" of segments to use later or to drive hard-copy equipment. The facilities can also be used for debugging or for certain kinds of "group sessions" in which users at several points in the network may be working concurrently and wish to "transmit" pictures to each other.

This collection of facilities is optional. The implementation should, however, be very simple, and we expect that most UP's will offer this service.

The command sent from the SP to the UP

`<seg.readback.seglist>`

causes the UP to transmit to the SP the sequence:

`<seg.readback.seglist> <"count"> <"seg.name"> ...
<"count"> <"seg.name"> ...`

The UP returns the count of posted segments, followed by their segment names, followed by the count of unposted segments, followed by their segment names.

The command from SP to UP

`<seg.readback.seg> <"seg.name">`

causes the UP to send back to the SP a sequence in network format that completely describes the named segment. The sequence can be any legal sequence of segment commands described in this section (including character size and type control commands) sufficient to unambiguously describe the segment (i.e., if the sequence were transmitted to the same UP, it would generate an identical display). The sequence is <seg.open> <"seg.name"> followed by any attributes of the segment, such as <set.hit.sensitivity.attribute> <on> or <set.intensity.attribute> <.6> followed by a sequence of graphical primitives that are contained in the segment.

Intensity, line type and character size commands (just like those sent from SP to UP) are imbedded in the sequence of primitives whenever necessary in order to specify the segment exactly.

Finally, at the end of the sequence of primitives, a <seg.close> is sent if the segment was unposted, or a <seg.post> <"seg.name"> is sent if the segment was posted.

III.12. Positioned Text

This section describes protocol for displaying positioned text on the screen. The positioned text facilities are intended to cater for the needs of display-oriented text editors, like NLS. The SP can instruct the UP to create "text windows" on the screen: a text window is a rectangular area in which a series of "lines" of text can be displayed. Protocol commands from the SP permit characters within a line to be changed, permit lines to be moved, and the like. All changes to the text windows specified by positioned text protocol take effect immediately.

The same mechanisms for dealing with positioned text can be used to provide "teletype simulation" for TELNET-like dialog with the server host, although this is entirely up to the UP implementation.

The commands for creating and destroying text windows are

```
<ptext.open> <"ptext.name"> <"count"> <"x.s.coord"> <"y.s.coord">
              <"x.s.coord"> <"y.s.coord"> <"count">

<ptext.kill> <"ptext.name">
```

The open command creates a text window, specifies a unique name for it, the number of lines it is to contain (the first <"count">), the coordinates of the lower left and upper right corners of the window in screen coordinates, and the (discrete) character size to use for all characters in the window (the second <"count">). If the name specified in the open command already exists, the old window is destroyed.

The kill command destroys the named text window. If no such window exists, the command is ignored.

A number of attributes of a text window can be set with the optional command

```
<ptext.set> <"ptext.name"> <"ptext.flags">
```

The <"ptext.flags"> byte is simply a <"small.integer"> of flag bits:

[`001] Accept unescorted characters ("teletype simulation"). If this bit is on, any characters transmitted by the server host as part of TELNET protocol should be displayed in this text window. They are added "at the end," with normal ASCII conventions about format characters. Optional.

[`002] Automatically scroll when text window fills up. This is probably only useful in conjunction with teletype simulation. Optional.

[`004] Wrap long lines around to a new line. This too is most useful in conjunction with teletype simulation. A wrapped string counts as a new string. Optional.

[`010] Make this text window visible. If the bit is off, the text window remains (and may be edited) but is not displayed. Optional.

[`020] Make this text window "hit sensitive." That is, the input facilities for pointing can be applied to the window. Optional.

A default setting of <ptext.flags> to `010 is performed when a text window is first created.

There are several commands for changing the contents of strings in a text window. Strings within a text window are identified by a number (0 to n-1 where n is the number of strings in the window and the topmost string in the window is numbered 0). A <string.number*> is simply the number of the string, in <count> format. Each string has an implicit length associated with it (internally in the UP); characters in a string are referred to by position (numbered starting at 1). Strings will normally contain standard ASCII printing characters.

The commands that modify strings are:

<ptext.scroll.up> <ptext.name*> <string.number*> <string.number*>

Within a text window, scroll lines from the first string number to the second string number up one line, i.e. If the two string numbers specified are x and y, replace line x with x+1, x+1 with x+2, ... y-1 with y, and y with a null string.

<ptext.scroll.down> <ptext.name*> <string.number*> <string.number*>

Similar, but scroll down.

<ptext.move> <ptext.name*> <string.number*> <ptext.name*> <string.number*>

This command causes the first string (specified by the text window name and string number) to be moved to the second string location. The first string is replaced with a null string.

<ptext.edit> <ptext.name*> <string.number*> <pos1*> <pos2*> <text*>

This is the main command for editing strings. The arguments <pos1*> and <pos2*> are character positions within the specified string (in <count> format). The effect of the edit command is to replace the substring beginning with character pos1 and ending with character pos2-1 with the specified text string. If pos1 = pos2, this simply means that the string is inserted before the pos1 th character. There are two classes of illegal substrings that can be specified with pos1 and pos2:

(305)

`pos1 > length of string.` In this case, the text string is appended to the end of the present string.

`pos2 > (length of string)+1.` In this case, the command has the same effect as if `pos2` were exactly `(length of string)+1`.

Note that the text specified may be a null string -- hence if `pos1=1`, `pos2=large number (e.g. 127)`, and `text=null`, the entire specified string is replaced by the null string.

```
<pTEXT.modify> <*pTEXT.name*> <*string.number*> <*pos1*> <*pos2*>
<*pTEXT.feature*>
```

This optional command is used to select a substring that is to receive special treatment (e.g. highlighting, or blanking). It can also be used to make individual lines visible and later invisible. The values of `<*pTEXT.feature*>` are:

```
<pTEXT.highlight.on>
<pTEXT.highlight.off>
<pTEXT.visible.on>
<pTEXT.visible.off>
```

Whenever a substring is inserted with the `<pTEXT.edit>` command, the default (`<pTEXT.highlight.off>`, `<pTEXT.visible.on>`) is established.

```
<pTEXT.remote.edit> <*pTEXT.name*> <*string.number*>
```

This optional command is an *ad hoc* attempt to allow UP's to implement various local line-editing regimens, without requiring interaction with the SH or SP. This command specifies a string in a positioned text window that is to be "edited" by the user. When the editing is finished, the line is returned to the SP via the TELNET connection. (This solution is unsatisfactory for several reasons, but it seems necessary to offer such a capability.)

(There have been some suggestions that we try to devise a subset of these facilities that could be implemented on some kinds of text terminal with no external character memory. If the terminal has the ability to insert and delete characters at will, and to do the scrolling functions listed above, then the only troublesome commands are the `<pTEXT.move>` command, which requires reading characters from an arbitrary line, and the `<pTEXT.modify>` command.)

III.13. Input Facilities

This section describes a set of input facilities for the graphics protocol. They have been kept very simple to allow simple interaction with graphics programs without tremendous complexity. Specific interactive requirements may necessitate special-purpose protocols. The input facilities are optional; many graphics application programs need only keyboard facilities provided by TELNET.

Many details concerning provision of input facilities are left to the designers of the UP. The main reason for this approach stems from vast differences among operating systems and input device hardware -- no detailed set of specifications could be met by any one site. For example, an operating system may use some

"thinning" algorithm when passing tablet coordinates to a program in the system; different operating systems will surely have different conventions. In addition, individual (human) users may prefer slightly different styles of interactions; some of the stylistic flexibility that the protocol leaves open to the UP can be exploited by the user. However, the protocol does specify the purpose of each kind of input device and event. The UP should abide by the spirit of these descriptions, although the details may vary.

The input provisions provide mechanisms for the server to:

1. Read the state of a device on demand.
2. Use an interactive technique, called an event.

III.14. Reading the State of Input Devices

This section describes facilities for reading the state of any input device available at the UH.

The repertoire of available input devices is reported by the UP to the SP in the inquiry response. This list includes, for each device that is available, a one-byte "index" that uniquely identifies the device. This index, referred to as an <input.device.number>, is used by the SP to request measurement of the state of the device.

The protocol provides for the following five types of devices:

Coordinate Device

The state of a coordinate device is a pair of coordinates in the screen coordinate system. These coordinates could be derived from a tablet and stylus, from a light pen, from two knobs, from a keyboard (by typing in values, or using a keyboard-propelled cursor) or whatever.

Linear Device

The purpose of linear devices, such as knobs, is to provide a fraction in the range 0 to 1 to the SP. The UP may, if it wishes, provide some "echoing" of the current knob value, such as a changing number on the screen.

Key Device

The purpose of keys (or buttons) is to provide "function button" information to the SP. A key may be a single button (on/off) or a collection, arranged to form a number of possible code combinations (e.g., NLS keyset). The state of the key is a code describing the state of the key (bit=0 for normal position, bit=1 for depressed position).

Keyboard Device

A keyboard device provides a way of typing characters in the standard ASCII set. The state of the keyboard is the ASCII character code for the key most recently struck, or zero if the character has already been reported. (Note that the "state" of a keyboard is somewhat nonsensical -- we are more interested in the "events" caused by a keyboard. The keyboard is listed here for completeness.)

(307)

Time

If the UP provides a "time" input device, then it is willing to report to the SP a measurement of the current time. The protocol makes no tight specifications about how time is measured, except that it should be counted up once every 10 to 100 milliseconds. The UP may choose to count time somewhat inaccurately (e.g., by counting in an idle loop).

The SP can request that the state of a number of input devices be read and reported back with the optional command:

`<input.report> <*count*> <*input.device.number*> ...`

which specifies a list of devices whose states are to be measured as nearly simultaneously as possible and returned to the SP in the format:

`<input.report> <*device.report.sequence*>`

where `<*device.report.sequence*>` is a list:

`<*count*> <*input.device.report*> ...`

Each class of device has a canonical reporting format, `<*input.device.report*>`. The following paragraphs describe the format of `<*input.device.report*>`.

Keyboard Device: `<*input.device.number*> <*count*>`

The character code of the last key struck is returned in `<*count*>` format. Zero is returned if no key has been struck since the last report.

Key Device: `<*input.device.number*> <*count*>`

This report simply specifies the current value of the code of the key device (0 to n-1 where n is the number of states of the device).

Linear Device: `<*input.device.number*> <*large.fraction*>`

This simply reports the reading of the device as a fraction of its maximum excursion.

Coordinate Device: `<*input.device.number*> <*x.s.coord*> <*y.s.coord*> <*sw*>`

This report gives the current coordinates of the input device, and an indication of whether the "pen switch" (if it exists) is depressed (`<*sw*> = <on>`) or not (`<*sw*> = <off>`).

Time Device: `<*input.device.number*> <*time*>`

The time report specifies the current time. The format of `<*time*>` is `<*large.integer*>`. Note that time is recorded modulo 2^{16} .

III.15. Input Events

The protocol provides a set of facilities for reporting to the SP the results of several kinds of input events initiated by the user. This is in contrast to the

"report-on-demand" facilities of the previous section. In particular, the protocol associates with several of the events a particular kind of interactive technique, often involving local feedback.

Keyboard Event

A keyboard event occurs when a key of a "keyboard device" is struck. The "report" for this event is the ASCII code of the key struck.

Key Event

Key events cause reports to be sent to the SP when the user manipulates a key device. A key event can be reported when the key is depressed or when it is released. (The NLS keyset is almost impossible to use unless key values are reported when a key is released.) The "report" is identical to reporting the state of a key, as described above.

Linear Event

The definition of this event, which can only be provided by "linear devices," is left to the UP. It might occur if the user changes the reading of a knob more than a certain threshold amount, or whatever.

Positioning Event

A coordinate device is used to specify a particular position or coordinate pair. For example, the user might briefly depress a tablet stylus and thus specify a position. The details of how the positioning event is caused are left up to the UP.

Pointing Event

A coordinate device is used to identify some segment, figure (structured picture definitions) or portion of positioned text that is currently displayed and that has been made "hit sensitive." The user can thus point at something on the screen. The SP expects to be told the name of the thing pointed at and the coordinates where the "hit" occurred. The UP can use a number of devices and techniques to provide these features (e.g. light pen, tablet and stylus with a hardware or software comparator). The details of when the hit is caused are left to the UP. The size of the "window" used to search for the hit is also left up to the UP (the human user may want to control this). As a local option, the UP may want to highlight in some way the segment or character that is pointed at. This identification can be removed when pointing is disabled or when the user is no longer pointing at the object (after a possible time lag). These details are up to the UP.

Stroke Event

A coordinate device is used to "draw" a stroke. The UP shows on the screen a track of ink left behind the coordinate positions, and reports to the SP the coordinates of points along the stroke. The details about when a stroke is terminated (and hence when the inking event is caused) are left to the UP. One common convention is to begin a stroke when the stylus pen switch is depressed, to continue recording points at some rate as long as the switch stays closed, and to terminate the stroke when the switch opens.

(309)

Multiple Stroke Event

This event is similar to the stroke event, but is used to record a sequence of several strokes. This is useful for on-line character recognition. The usual technique is to assume the user is finished with a collection of strokes when a period of, say, 0.5 second has elapsed since the completion of the last stroke.

Dragging Event

A coordinate device is used to move figures around on the display screen without requiring intervention from the SP. The idea is to attach a segment to the coordinates delivered from a coordinate input device, such as a tablet stylus. This interaction is not possible on many kinds of displays (e.g. storage tubes); UP's driving these terminals will not be able to provide dragging. On many refresh displays, an implementation of dragging is very simple: each segment can be defined as an absolute position, followed by relative motions (i.e. relative vectors, and relative invisible motions). In addition as the segment is recorded in the UP, we record the maximum and minimum values of x and y that the beam visits. In order to drag the segment about, we receive coordinates from the input device, check against the x and y maxima and minima to be sure that the new coordinate position will not cause any portion of the segment to go off the edge of the screen, and if not, the coordinates of the initial position instruction are replaced by the tablet coordinates (this check is only necessary for displays that cannot tolerate vectors or text that go off the screen). In order to be dragged, a segment must have been created with the <set.position.attribute> attribute specified (thus the UP can generate the segment of the display file as described above).

Pendown, Penup Events

These are special cases of positioning events which may be meaningful in certain applications. The pendown event is caused when a stylus is pressed onto a tablet surface; the penup event when it is raised.

If any of the positioning, pointing, stroke, multiple stroke, dragging, penup or pendown events are enabled, the UP should cause a tracking dot (or some form of positional feedback) to appear on the screen.

III.16. Enabling Events

The repertoire of input events is reported by the UP to the SP in the inquiry response. This list includes a one-byte index, the <*input.event.number*>, that uniquely identifies each event implemented by the UP. If, for example, a pointing event can be caused by a light pen or by a stylus device, then two separate <*input.event.number*>s are assigned, one for the corresponding event on each device.

The SP may send commands that enable and disable input events. If an event is not enabled, the UP can ignore the corresponding device (i.e. need not buffer events that occur on an un-enabled device). (As a local option, the UP may send characters typed on an unenabled keyboard through the TELNET connection as "unescorted characters.")

When the SP enables for an event, it specifies the event being enabled, the conditions under which the event will be disabled, and what is to be reported when the event occurs. The command is:

```
<input.enable> <"input.event.number">
    <"input.disable.condition">
        <"input.report.sequence">
```

The <"input.disable.condition"> is one of:

- | | |
|--|---|
| <never> <with.this.event> <with.any.event> | Event remains enabled (until further notice from SP). Disable this event when this event occurs. Disable this event when the next event occurs. |
|--|---|

When an enabled event occurs, the UP sends to the SP an <"input.event.report"> that describes the results of the event that occurred. In addition, the application program may desire that the state of various input devices be measured when the event occurs, and that these readings also be reported. When an event is enabled, the <"input.report.sequence"> specifies an ordered list of devices whose state should be measured:

```
<"count"> <"input.device.number"> ...
```

The UP should save the report list and associate it with the event that is enabled with this command. When the event occurs, the report list is used to compose a message for the SP that describes the event. (This is the mechanism used to report the time at which an event occurs.)

The dragging event requires an additional argument, the <"seg.name"> of the segment that is to be dragged. This is treated as a special case, and is set (before enabling) with the command:

```
<input.drag.set> <"input.event.number"> <"seg.name">
```

An event may be explicitly disabled by

```
<input.disable> <"input.event.number">
```

(If stroke collection is disabled, the ink is cleared. If pointing is disabled, any highlighting for the object pointed at can be cleared.)

The entire input system is cleared and all events disabled with the command

```
<input.reset.system>
```

III.17. Event Reports

When an input event occurs, an event report is sent from the UP to the SP. The form of an event report is:

```
<input.report> <"input.event.report"> <"input.report.sequence">
```

(311)

The <input.report.sequence> is defined above, and is the collection of state measurements made on various input devices when the event occurred. The <input.event.report> has a canonical form for each event class as follows (unless otherwise noted, the format is the same as the corresponding <input.device.report>):

Keyboard Event: <input.event.number> <count>

Key Event: <input.event.number> <count>

Linear Event: <input.event.number> <large.fraction>

Positioning Event: <input.event.number> <x.s.coord> <y.s.coord>

Pendown Event: <input.event.number> <x.s.coord> <y.s.coord>

Penup Event: <input.event.number> <x.s.coord> <y.s.coord>

These last three reports simply specify a coordinate pair, in the screen coordinate system.

Pointing Event: <input.event.number> <hit>

This report varies with different kinds of things that are hit. A <hit> is one of two things:

<segment.hit> <seg.name> <x.s.coord> <y.s.coord>
<ptext.hit> <ptext.name> <string.number> <pos1>

The first specifies the name of the entity that was pointed to. The last specifies the name of the text window, the string number and character position within the string that was identified.

Stroke Event: <input.event.number> <timed> <stroke>

There are two ways of reporting stroke information: with and without times associated with each point. The SP requests that time be reported by including the "time" device in the <input.report.sequence> when enabling the stroke event. The UP will then record times if it can.

The stroke report contains an indication of whether timing information is associated with each coordinate pair. If <timed> is <off>, a <stroke> is:

<count> <x.s.coord> <y.s.coord> ...

where <count> is the number of coordinate pairs that follow. If <timed> is <on>, a stroke is:

<count> <x.s.coord> <y.s.coord> <time> ...

In other words, each coordinate pair has a time associated with it as well.

Multiple Stroke Event: <input.event.number> <timed> <count> <stroke> ...

This report is very similar to the stroke report, but has a provision for listing several strokes. The <count> is the number of strokes reported.

Dragging Event: <*input.event.number*> <*x.s.coord*> <*y.s.coord*>

This report simply specifies the coordinate pair that replaces the original home (i.e. first 'move' instruction) of the dragged segment.

III.18. Inquiry

The UP can transmit to the SP a number of bytes that describe the terminal serviced by the UP. This information is constant (so that a UP implementation does not have to compute it each time), and is usually requested by the SP at the beginning of a graphics session.

The SP can ask for the inquiry response by transmitting to the UP the command:

<Inquire>

The UP then transmits to the SP the sequence

<Inquire.response> <*count*> <*response.phrase*> ...

This response includes a count of the number of response "phrases" that follow, and the response phrases, in any order. A <*response.phrase*> is

<*response.tag*> <*count*> <*response.value*>

The count is the number of bytes in this particular <*response.value*>. The reason for this organization is to make the inquiry responses open-ended: the SP can ignore <*response.value*>s it cannot interpret.

In the description below, the tags and values for each kind of information are specified. If a default is specified, it may be assumed if the inquiry response does not include a phrase that overrides the default.

UP Features**1. What protocol commands are implemented in the UP?**

Tag: <iImplemented.commands>

Value: up to 32 bytes of bit mask

The i th bit of the mask is a 1 if command i is implemented (i ranges from 0 to 255). Since the <*response.phrase*> for this information contains a count of the number of bytes that comprise the response, it is not necessary to provide the entire 32 bytes in the response. In the present protocol, there are only 102 command codes assigned (0-101), so only 13 bytes are required to completely describe which commands are implemented. This information is mandatory in the response.

Terminal Features**2. What is the screen coordinate system?**

Tag: <iScreen.coordinates>

Value: <*x.left.lv*> <*y.bottom.lv*> <*x.right.lv*> <*y.top.lv*> <*count*>

This specifies precisely the admissible values for the <*x.s.coord*> and <*y.s.coord*> sequences in the subsequent protocol. The <*count*> specifies how many bytes are used to make up a coordinate value (e.g., this would be 2 for displays that are addressed as 0 to 1023). Thus, the number of bytes in a <* .s.coord*> is fixed by the UP.

(3'(3)

The four <*.lv*> constructs that follow are examples of the screen coordinate system. Each <*.lv*> is a 4-byte sequence that specifies a 32-bit signed two's complement number. The four constructs thus specify the left, bottom, right, and top limits of the addressing space of the terminal.

Example: An IMLAC requires two bytes per coordinate; the lower-left corner of the screen is (0,0) and the upper right is (1023,1023). The 17 bytes of response are thus:

```
'000 '000 '000 '000 ;x left
'000 '000 '000 '000 ;y bottom
'000 '000 '003 '377 ;x right
'000 '000 '003 '377 ;y top
      '002 ;number of bytes in <*.s.coord*>
```

As an example of the computation of a screen coordinate, suppose that we wish to compute the x screen coordinate of a spot that is the fraction f of the width of the screen ($f=0$ is at the left edge, $f=1$ at the right). Compute $s = (<x.right.lv*>-<x.left.lv*>)*f + <x.left.lv*>$. Then transmit to the UP the low-order n bytes of the result, where n is the value of <count> in the response.

This response is mandatory.

3. What is the screen size?

Tag: <i.screen.size>

Value: <text> <text>

The two text strings specify, in decimal format (e.g. 126.43), the x and y dimensions of the screen in centimeters. Note that this format is chosen so that roll plotters can work effectively: they might choose a huge screen coordinate system, and specify a long dimension as well, e.g. 1000 centimeters.

4. How many screens are there?

Tag: <i.screen.number>

Value: <count>

Default is 1.

5. What is the device name?

Tag: <i.terminal.name>

Value: <text> <text>

Two strings are returned: the first is some form of manufacturer's name for the terminal (e.g. "IBM 2250"). The second is a string that can uniquely identify the terminal at the user site (e.g. "Terminal in room 34."). The protocol makes no specific format requirements on these strings -- they are for information only.

6. What is the terminal type?

Tag: <i.terminal.type>

Value: either <storage.terminal>, <storage.with.selective.erase>, <refresh.calligraphic> or <refresh.video>

7. How many resolvable intensity values are there?

Tag: <i.intensities>

Value: <count>

If the <count> value is n, then the permissible values as arguments to

the intensity-setting functions are 0 (no intensity) through n-1 (maximum intensity). Default n=2.

8. How many different line types are there?

Tag: <i.line.type>

Value: <*count*>

If the <*count*> value is n, then the permissible values as arguments to the type-setting functions are 0 (solid) through n-1. Default n=1.

9. What characters can be displayed on the terminal?

Tag: <i.characters>

Value: 32 bytes

Each of the 128 ASCII characters has a 2-bit code for it. The codes are 00 (cannot display), 01 (can display exactly), 10 (can transliterate, e.g. lower case to upper case, or tab to spaces), 11 (this is some visible character, but not ASCII). Default: 64 character ASCII.

10. What character orientations are there?

Tag: <i.character.orientations>

Value: <*count*> <*large.fraction*> ...

This response returns a list of available discrete orientations. Each fraction represents an angle (in range 0 to 2π) that is available (e.g. 0=normal horizontal; 1/4 is vertical running upward, etc.). If the <*count*> is n, then indices passed to the <set.character.orientation.discrete> command are in the range 0 to n-1. Default n=1, and the corresponding direction is 0.

11. What are the character sizes?

Tag: <i.character.size>

Value: <*count*> <*char.size.description*> ...

If <*count*> is n, then there are n discrete character sizes available, and the arguments to <set.character.size.discrete> should range from 0 to n-1. The character size descriptions are (as nearly as possible) in order of increasing size.

12. What input devices are available?

Tag: <i.available.input.device>

Value: <*input.device.number*> <*input.device.description*>

<*events.provided*> <*text*>

This response phrase specifies the identity of one input device (they are broken out so that the SP can skip over individual ones whose format it cannot interpret). The <*input.device.number*> is one byte that is to be used by the SP to reference the device. The <*text*> is a text string for human consumption that describes the device. (This string might be used by the SP to engage in a dialog with the user, asking him which devices to use for what function.) The <*input.device.description*> is one of:

```

<device.keyboard>
<device.key> <*count*>
<device.linear>
<device.coordinate>
<device.time> <*count*>

```

The key device gives, in <*count*>, the number of states the key can assume (e.g., an NLS keyset can assume 32 states). The time device gives, in <*count*>, the approximate number of milliseconds that elapse between increments to the time counter.

Each device also lists the kinds of events it can provide, and the <input.event.number>s used to reference the events. The reason for providing unique <input.event.number>s for each (device,event) pair is so that more than one device can provide similar interactive techniques. The <events.provided> is:

```
<count> <input.event.number> <event.description> ...
```

The count is the number of different events this device can provide. Each event is specified by its unique index and the <event.description>, which is one of:

```
<event.keyboard>
<event.key>
<event.linear>
<event.positioning>
<event.pointing>
<event.stroke>
<event.multiple.stroke>
<event.dragging>
<event.pendown>
<event.penup>
```

III.19. Miscellaneous

This section describes a collection of miscellaneous commands provided in the protocol.

The sequence

```
<escape.protocol> <text>
```

is a way for the SP to transmit device-dependent information to the UP or from the UP to the SP. The protocol makes no provision for the format or encoding of the text string.

The command from SP to UP

```
<synchronize> <count>
```

causes the UP to respond

```
<synchronize> <count>
```

This provides a method of synchronizing things if absolutely necessary (e.g. a way of knowing whether a certain input event was caused before or after the display was changed with an "end batch of updates" command).

The command

```
<reset>
```

causes the UP to reset itself to a point right after the initial connection protocol has been completed and the connections opened. During early stages of debugging server and user software, this will doubtless be very useful.

Error conditions detected in the UP may be handled in several ways (the protocol makes no precise requirements):

1. Ignore them, or have the UP try to continue with as little damage as possible. Even severe errors should not crash the UP, since its operation is essential to permit the user to communicate with the SH and the application program.
2. Inform the SP of the error detected.
3. Inform the user (locally) of the error detected.

As a general strategy, the UP should probably try these in order. Certainly the UP should attempt to deal adequately with all errors (particularly such things as running out of display buffer space) so as to minimize the chances of a user losing a session's work.

Experience with common errors is probably needed before a useful error-management scheme can be devised. For this reason, we provide a mechanism for the UP to report to the SP any error conditions it detects (in free text format) and vice-versa. Thus system programmers at each end can, by saving and later examining error messages, keep track of major sources of error. The error report is

`<error.string> <*text*>`

In some networks, it may be necessary to establish synchronization of sender and receiver of protocol. For example, if a message is lost in the network, the UP may start interpreting operand bytes as if they were command codes. Since this is not a serious problem in the ARPA network, the present protocol does not enforce a synchronization mechanism. The following scheme is believed to be adequate, and will be instituted if network reliability is a problem:

We shall define a synchronization command, called GS, that has a code different from that of any protocol command. Whenever a data byte that is equal to GS is transmitted, it must be doubled (i.e., it is transmitted as GS, GS). A single GS may precede any protocol command code. Thus, whenever a receiver encounters a single GS, it knows that the next byte is a protocol command, and not an operand. The sender may transmit a GS preceding any command byte. It may choose to transmit as many or as few of these as seem appropriate.

SECTION IV

Implementation Suggestions

The details of the protocol may seem prohibitively forbidding. This section attempts to dispel that notion and to make it all appear so simple that standard mortals can implement it.

The op-code definitions (see following section) group the commands into four groups:

Mandatory
Group 1 -- Transformed Format
Group 2 -- Positioned Text
Group 3 -- Input

Once the mandatory functions are implemented, any combination of the remaining three groups and the various optional commands may be implemented. For example, a UP that implements the Mandatory functions and Group 1 will be very useful indeed: many curve-plotting and mathematics packages that wish to do graphic output but are content with teletype-like input (provided by TELNET) can now be used.

An implementation of Mandatory and Group 2 would be adequate for simple page-oriented text editors; addition of Group 3 permits NLS and other more interactive systems to work. (The only combination that is probably not meaningful is just Mandatory and Group 3.)

Since many of the features of the protocol are optional, it is likely that many implementations will not include many of the options. There is no stigma associated with omitting optional portions.

In order to distribute information about sites that have implemented server or user programs that conform to the protocol, we would like to establish an informal "clearing-house" for such information. Those with information to give or request should address:

Robert F. Sproull
Xerox Palo Alto Research Center
3180 Porter Drive
Palo Alto, Calif. 94304

or

SPROULL@PARC-MAXC (ARPA network mail)

Especially welcome is information about implementations of the protocol that can be offered to others (e.g., if someone writes an SP facility for INTERLISP, or a UP for an IMLAC).

SECTION V

Op-Code Assignments and Options

This section assigns a number for each of the protocol command bytes described in section III, and tries to indicate what is optional and what is not.

The options column describes the conditions under which the UP should implement the command. M stands for "mandatory," and O for "optional." M-1 means mandatory if any commands in group 1 are implemented, i.e. if any M-1 command is implemented, then all M-1 commands must be. O-1 means optional if M-1 commands are implemented; otherwise not implemented.

The op-codes are grouped so that they may be decoded without requiring a full dispatch table if entire groups are unimplemented. The high-order three bits of the op code are the group number, the remaining five bits are the command within the group.

Inquiry Commands

<Inquire>	1	M
<Inquire.response>	2	M

General Commands

<escape.protocol>	3	O
<synchronize>	4	O
<reset>	5	O
<error.string>	6	O

Transformed Format Commands (Group 1)

<seg.open>	32	M-1
<seg.close>	33	M-1
<seg.post>	34	M-1
<seg.unpost>	35	M-1
<seg.kill>	36	M-1
<end.batch.of.updates>	37	M-1
<seg.append>	38	O-1
<seg.dot>	39	M-1
<seg.move>	40	M-1
<seg.draw>	41	M-1
<seg.text>	42	M-1
<set.intensity>	43	O-1
<set.type>	44	O-1
<set.character.orientation.discrete>	45	O-1
<set.character.orientation.continuous>	46	O-1
<set.character.size.discrete>	47	O-1
<set.character.size.continuous>	48	O-1
<change.attribute>	49	O-1
<set.highlight.attribute>	50	O-1
<set.hit.sensitivity.attribute>	51	O-1
<set.intensity.attribute>	52	O-1
<set.screen.select.attribute>	53	O-1
<set.position.attribute>	54	O-1

(319)

<code><seg.readback.seglist></code>	55	O-1
<code><seg.readback.seq></code>	56	O-1
Positioned Text Commands (Group 2)		
<code><ptext.open></code>	64	M-2
<code><ptext.kill></code>	65	M-2
<code><ptext.set></code>	66	O-2
<code><ptext.scroll.up></code>	67	O-2
<code><ptext.scroll.down></code>	68	O-2
<code><ptext.move></code>	69	O-2
<code><ptext.edit></code>	70	M-2
<code><ptext.modify></code>	71	O-2
<code><ptext.remote.edit></code>	72	O-2
Input Commands (Group 3)		
<code><input.enable></code>	96	M-3
<code><input.disable></code>	97	M-3
<code><input.report></code>	98	O-3
<code><input.event></code>	99	M-3
<code><input.reset.system></code>	100	M-3
<code><input.drag.set></code>	101	O-3
Other code assignments (these are not commands)		
<code><off></code>	0	
<code><on></code>	1	
Positioned Text		
<code><ptext.visible.off></code>	0	
<code><ptext.visible.on></code>	1	
<code><ptext.highlight.off></code>	2	
<code><ptext.highlight.on></code>	3	
Input facilities		
<code><device.keyboard></code>	0	
<code><device.key></code>	1	
<code><device.linear></code>	2	
<code><device.coordinate></code>	3	
<code><device.time></code>	4	
<code><event.keyboard></code>	0	
<code><event.key></code>	1	
<code><event.linear></code>	2	
<code><event.positioning></code>	5	
<code><event.pointing></code>	6	
<code><event.stroke></code>	7	
<code><event.multiple.stroke></code>	8	
<code><event.dragging></code>	9	
<code><event.pendown></code>	10	
<code><event.penup></code>	11	
<code><never></code>	0	
<code><with.this.event></code>	1	
<code><with.any.event></code>	2	
<code><segment.hit></code>	0	
<code><ptext.hit></code>	1	

Inquiry tag definitions

<i.implemented.commands>	1	M
<i.screen.coordinates>	2	M
<i.screen.size>	3	O
<i.screen.number>	4	O
<i.terminal.name>	5	O
<i.terminal.type>	6	O
<i.intensities>	7	O
<i.line.type>	8	O
<i.characters>	9	O
<i.character.orientations>	10	O
<i.character.size>	11	O
<i.available.input.device>	12	O
<refresh.calligraphic>	0	
<storage.terminal>	1	
<storage.with.selective.erase>	2	
<refresh.video>	3	

(321)

Appendix

Outline of Structured Format Protocol

This appendix presents a preliminary design for a structured output format protocol. It is similar to the "groups and items" technique [N&S]. It caters primarily for high-performance displays that are capable of implementing transformations in hardware and of interpreting a structured display file. However, software processes can be used by a UP to simulate these facilities if the display does not have the capability.

Display Structure

A *display structure* consists of *figures*, each containing any number of *units*. There are two types of units, *primitive* units and *call* units. Primitive units contain drawing instructions and associated coordinates that may generate visible information on the display screen. Drawing instructions and coordinates can occur only in primitive units.

Call units give the display structure a subroutine capability. A call unit invokes the display of another figure. In other words, a call unit allows one figure to contain instances of other figures. As well as providing for subroutine-style control transfer, call units can be used to establish the parameters to be used in the display of the subfigure. For example, a call unit can be used to call a figure with a specified intensity setting and translation. On return from the called figure, these parameters are restored to their original values.

A figure is an ordered list of units which can be any mixture of primitive and call units. Each figure begins with a *header* and terminates with the *figure end unit*. The ordering of units within a figure does not affect the display produced, but, particularly in languages such as LISP, it may be convenient to have this ordering correspond to some application data structure ordering.

In order to understand how control passes through a structure, one can think of the display elements as follows: figures are subroutines and units are linked blocks of in-line code. When all of the units contained in a figure have been executed, the figure end unit returns control to wherever the figure was called from. A primitive unit contains line and character descriptions and a transfer to the next unit. A call unit contains a subroutine call to a subfigure and a transfer to the next unit in line. Figure 7 shows a typical display structure.

Accessing Mechanisms

Figures are referenced by user-assigned names. Units may be given names at the option of the user. No two figures can have the same name, and no two units within a figure can have the same name. However, units in separate figures can have identical names, and a unit can be "named after" (i.e., carry the same name as) a figure. Figure names are called *global* and unit names *local* to reflect the fact that a unit name only distinguishes between the units within a figure.

To reference a figure, one merely refers to it by name. To reference a unit within a figure, one supplies both the figure and unit names. It is this naming mechanism which makes it possible to make incremental changes in the display.

The display structure exists at the OH, rather than at the application program.

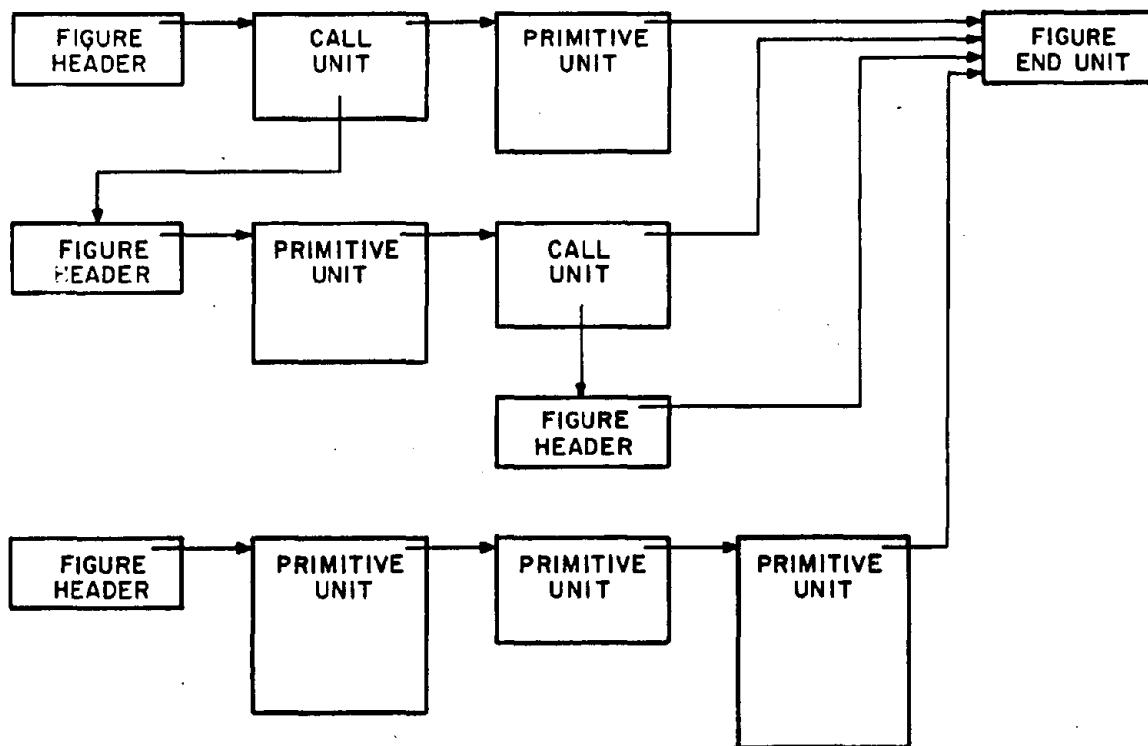


Figure 7: A Typical Display Structure

For figure and unit names to be useful, the application program should generate display names carefully to insure a one-to-one correspondence between display names and application program data structure parts.

Primitive Unit

A primitive unit can be used to draw any combination of lines and characters. More specifically, a primitive unit can consist of any number or combination of graphical primitives for drawing dots, lines and text (similar to the primitives for the transformed format, section III.6). Display coordinates are two's complement fractions of appropriate resolution centered about the point (0,0).

An option should be provided in the protocol to specify graphical primitives in a three-dimensional coordinate system.

Call Unit

Call units enable several similar picture parts to be described by the same figure. For example, the display of a circuit diagram can be constructed from a figure which is composed of the lines for a resistor and call units of this figure for each resistor in the display. All coordinate transformations and changes in intensity are specified as parameters of call units.

Master and instance rectangles are used to specify the clipping and scaling of

(323)

coordinates. A *master rectangle* is an area in the called figure which is to be mapped into an area specified by the *instance rectangle* in the calling figure (see Figure 8). Lines in the called figure which have coordinates outside the master rectangle are not displayed in the calling figure; lines in the called figure which cross the master rectangle are clipped when they are displayed in the calling figure.

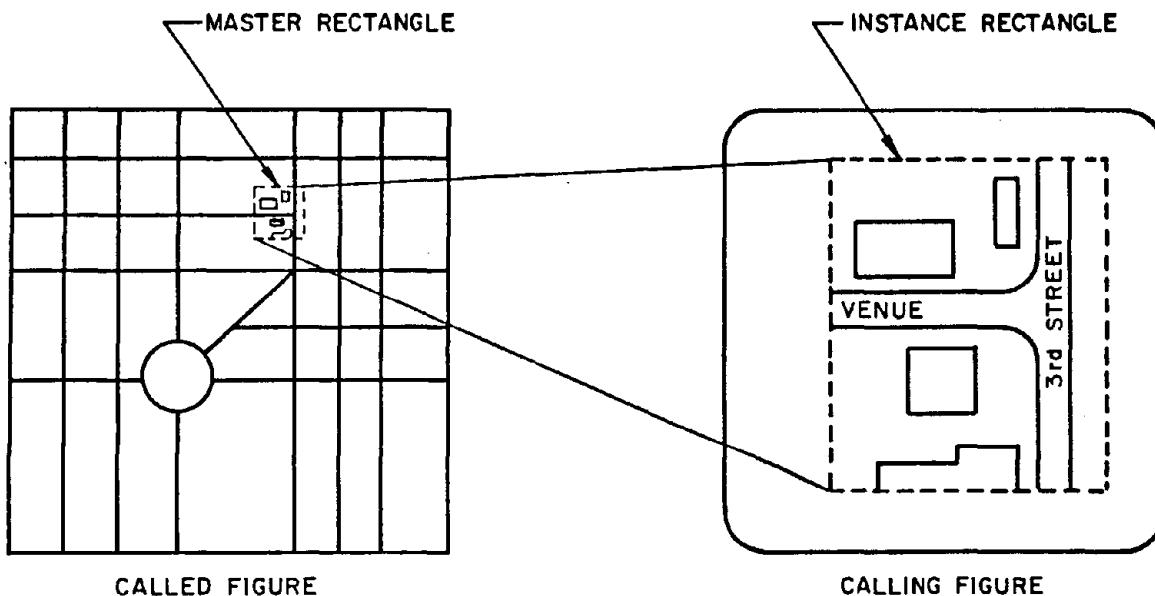


Figure 8: Clipping: Master and Instance Rectangles

Several ways of describing the master and instance rectangles of a call unit should be provided in the protocol:

1. If no master and instance rectangles are specified when creating a call unit, the coordinates of the called figure are not clipped or transformed but are displayed just as they appear in the called figure.
2. A *move call unit* has master and instance rectangles which are maximum in size and offset with respect to each other, thus "moving" all the coordinates of the called figure by a specified amount in the calling figure.
3. A *scale call unit* moves and scales the called figure with respect to the calling figure. Either the master rectangle or the instance rectangle is maximum in size, and the other rectangle is suitably smaller to achieve the desired scale.
4. Both the master and instance rectangles can be specified directly by the user. A rectangle is designated by its lower left-hand and upper right-hand corners.

A call unit also can specify an angle of rotation to be applied to the called figure. Display coordinates are rotated before they are scaled and clipped.

An option should be provided so that a call unit can specify a three-dimensional transformation. This allows displays of graphical primitives that have been specified in three dimensions. Ideally, the transformation should include the ability to specify a perspective view.

The intensity value specified in a call unit is the absolute intensity level at which the called figure is to be displayed. If no intensity is specified, the intensity remains unchanged from that of the calling figure. The default intensity of the display is the scope's brightest intensity.

Many displays have blinking line and dashed line capabilities. A call unit can be used to change the line type to dashed and/or blinking for all the lines of the called figure. If a particular scope does not have the requested capability then the line type remains unchanged (analogous to transformed format, section III.8).

Display Structure Construction and Modification

A display structure is constructed by creating its units. When a unit is created, the figure containing the unit must be specified. If the figure does not exist, it is also created. In addition, if the figure called by a call unit does not exist, it is created. Thus, new figures are created implicitly by placing units in them or by calling them from some other figure.

A unit can be inserted in the structure in one of three ways:

1. It can be inserted at the end of a figure.
2. It can be inserted after a particular unit in a figure (where the figure header is an acceptable unit after which to insert).
3. It can replace a particular unit which already exists.

If a unit with the same local name as the new unit already exists in the figure, the old unit will be deleted as a result of the creation of the new unit.

The function "display figure" causes the specified figure to be displayed. The arguments to this function are similar to those of a call unit: a master rectangle is applied to the called figure, and mapped onto a "viewport," specified in the screen coordinate system. This call unit is distinguished from all others because it alone specifies a mapping from the large two's complement coordinate system of figures and units to the screen coordinate system (which may not have a square aspect ratio). At any one time, only a single figure can be displayed. However, this is not restrictive since the figure on display can call any or all other figures. As units are added to the displayed figure, they are displayed. An empty figure is created as a result of the "display figure" function if the figure does not already exist. The function "clear scope" removes all such displayed figures from view.

The protocol should provide several mechanisms to change a display structure incrementally. Individual units and figures can be deleted from the display structure, the names of figures and units can be changed, and units can be blanked and unblanked.

Three types of deletion operations may be performed:

1. A single unit can be deleted.

2. A figure can be cleared, an operation which deletes each of the units of a figure but retains the figure header.

3. A figure can be deleted, thereby deleting all the units of the figure, the figure header, and all call units which reference the figure.

In all of the deletion operations, there is no error generated if the object to be deleted does not exist.

A unit or figure can be given a new name. If a unit is given the same name as some other unit in the same figure, the unit originally carrying the name is eliminated. If a figure name is changed to that of some other figure, the figure that already had the name is eliminated, along with any calls to it.

A blanked unit is a unit which exists in the display structure but which is marked not to be displayed. Thus, the lines and characters of a blanked primitive unit are not drawn and the figure referenced by a blanked call unit is not processed. Blanking and unblanking a unit is more efficient than deleting and recreating it.

If the transformations are being interpreted in software, it is often desirable to make several changes to the display structure before repainting the scope with the updated picture. The "end batch of updates" command is used to cause a complete update to the visible display.

Input Facilities

Two of the input facilities of the protocol take on a different meaning in conjunction with a structured display file: dragging and pointing.

For pointing, the SP can make individual figures "hit sensitive." Then, if the pointing event is enabled and the user identifies an object visible on the screen, the event report cites the global name and local name (if any) of the unit "hit."

Dragging is accomplished by identifying a move call unit whose parameters are to be changed by coordinates delivered from a coordinate input device. (For simplicity, the input device coordinates are used directly as the offset of the called figure to the calling figure. Thus, if the coordinates of the calling figure are transformed in any way, the movement of the called figure will be related to, but not directly tied to the movement of the input device.)

August 16, 1974

REF E R E N C E S

[N&S]

W.M. Newman and R.F. Sproull, *Principles of Interactive Computer Graphics*, McGraw Hill, 1973.

[10GR]

W.M. Newman and R.F. Sproull, "An Approach to Graphics System Design," *Proceedings of the IEEE*, April 1974.

[Omnigraph.brief]

R.F. Sproull, "Omnigraph -- Simple Terminal-Independent Graphics Software," Xerox PARC Report CSL-73-4.

[Omnigraph]

"PDP-10 Display Systems," available from Computer Center Branch, Division of Computer Research and Technology, National Institutes of Health, Bethesda, Maryland 20014.

[NIC 19933]

"Proposed Network Graphics Protocol," Network Information Center 19933. (Unfortunately, this is no longer available from the NIC.)

[NLS]

D.C. Engelbart and W.K. English, "A Research Center for Augmenting Human Intellect," FJCC 1968, p. 395.

(327)

INDEX

<attribute> 23
<char.size.description> 23
<count> 15
<device.report.sequence> 30
<hit> 34
<input.device.number> 29
<input.device.report> 30
<input.disable.condition> 33
<input.event.number> 32
<input.event.report> 34
<input.report.sequence> 33
<large.fraction> 15
<large.integer> 15
<ptext.feature> 28
<ptext.flags> 26
<ptext.name> 15
<response.phrase> 35
<response.tag> 35
<response.value> 35
<seg.name> 15
<small.fraction> 15
<small.integer> 15
<string.number> 27
<text> 15
<time> 30
<timed> 34
<x.s.coord> 21, 35
<y.s.coord> 21, 35
<change.attribute> 23
<end.batch.of.updates> 20
<error.string> 39
<escape.protocol> 38
<input.disable> 33
<input.drag.set> 33
<input.enable> 33
<input.report> 30, 33
<input.reset.system> 33
<inquire.response> 35
<inquire> 35
<never> 33
<ptext.edit> 27
<ptext.kill> 26
<ptext.modify> 28
<ptext.move> 27
<ptext.open> 26
<ptext.remote.edit> 28
<ptext.scroll.down> 27
<ptext.scroll.up> 27
<ptext.set> 26
<reset> 38
<seg.append> 19

`<seg.close>` 19
`<seg.dot>` 21
`<seg.draw>` 21
`<seg.kill>` 19
`<seg.move>` 21
`<seg.open>` 19
`<seg.post>` 19
`<seg.readback.seg>` 25
`<seg.readback.seglist>` 25
`<seg.text>` 21
`<seg.unpost>` 19
`<set.character.orientation.continuous>` 23
`<set.character.orientation.discrete>` 22
`<set.character.size.continuous>` 23
`<set.character.size.discrete>` 23
`<set.highlight.attribute>` 24
`<set.hit.sensitivity.attribute>` 24
`<set.intensity.attribute>` 24
`<set.intensity>` 21
`<set.position.attribute>` 25
`<set.screen.select.attribute>` 25
`<set.type>` 22
`<synchronize>` 38
`<with.any.event>` 33
`<with.this.event>` 33

DATA CONFIGURATION PROTOCOL

Preceding page blank

6780

**Network Working Group
Request for Comments: 166
NIC 6780**

25 May 1971	
Bob Anderson	Rand
Vint Cerf	UCLA
Eric Harslem	Rand
John Heafner	Rand
Jim Madden	U. of Ill.
Bob Metcalfe	MIT
Arie Shoshani	SDC
Jim White	UCSB
David Wood	Mitre

**DATA RECONFIGURATION SERVICE --
AN IMPLEMENTATION SPECIFICATION**

Preceding page blank

(333)

CONTENTS

I.	INTRODUCTION	1
	Purpose of this RFC	1
	Motivation	1
II.	OVERVIEW OF THE DATA RECONFIGURATION SERVICE ...	3
	Elements of the Data Reconfiguration Service ...	3
	Conceptual Network Connections	4
	Connection Protocols and Message Formats	5
	Example Connection Configurations	8
III.	THE FORM MACHINE	10
	Input/Output Streams and Forms	10
	Form Machine BNF Syntax	11
	Alternate Specification of Form Machine Syntax .	12
	Forms	13
	Rules	13
	Terms	14
	Term Format 1	14
	Term Format 2	14
	Term Format 3	17
	Term Format 4	18
	The Application of a Term	18
	Restrictions and Interpretations of Term Functions	20
	Term and Rule Sequencing	21
IV.	EXAMPLES	23
	Remarks	23
	Field Insertion	23
	Deletion	23
	Variable Length Records	24
	String Length Computation	24
	Transposition	24
	Character Packing and Unpacking	25

Preceding page blank

(335)

I. INTRODUCTION

PURPOSE OF THIS RFC

The purpose of this RFC is to specify the Data Reconfiguration Service (DRS.)

The DRS experiment involves a software mechanism to reformat Network data streams. The mechanism can be adapted to numerous Network application programs. We hope that the results of the experiment will lead to a future standard service that embodies the principles described in this RFC.

MOTIVATION

Application programs require specific data I/O formats yet the formats are different from program to program. We take the position that the Network should adapt to the individual program requirements rather than changing each program to comply with a standard. This position doesn't preclude the use of standards that describe the formats of regular message contents; it is merely an interpretation of a standard as being a desirable mode of operation but not a necessary one.

In addition to differing program requirements, a format mismatch problem occurs where users wish to employ many different kinds of consoles to attach to a single service program. It is desirable to have the Network adapt to individual console configurations rather than requiring unique software packages for each console transformation.

One approach to providing adaptation is for those sites with substantial computing power to offer a data reconfiguration service; this document is a specification of such a service.

The envisioned modus operandi of the service is that an applications programmer defines forms that describe data reconfigurations. The service stores the forms by name. At a later time, a user (perhaps a non-programmer) employs the service to accomplish a particular transformation of a Network data stream, simply by calling the form by name.

We have attempted to provide a notation tailored to some specifically needed instances of data reformatting while keeping the notation and its underlying implementation within some utility range that is bounded on the lower end by a notation expressive enough to make the experimental service useful, and that is bounded on the upper end by a notation short of a general purpose programming language.

II. OVERVIEW OF THE DATA RECONFIGURATION SERVICE

ELEMENTS OF THE DATA RECONFIGURATION SERVICE

An implementation of the Data Reconfiguration Service (DRS) includes modules for connection protocols, a handler of some requests that can be made of the service, a compiler and/or interpreter (called the Form Machine) to act on those requests, and a file storage module for saving and retrieving definitions of data reconfigurations (forms).

This section describes connection protocols and requests. The next section covers the Form Machine language in some detail. File storage is not described in this document because it is transparent to the use of the service and its implementation is different at each DRS host.

CONCEPTUAL NETWORK CONNECTIONS

There are three conceptual Network connections to the DRS, see Fig. 1.

- 1) The control connection (CC) is between an originating user and the DRS. Forms specifying data reconfigurations are defined over this connection. The user indicates (once) forms to be applied to data passing over the two connections described below.
- 2) The user connection (UC) is between a user process and the DRS.
- 3) The server connection (SC) is between the DRS and the serving process.

Since the goal is to adapt the Network to user and server processes, a minimum of requirements are imposed on the UC and SC.

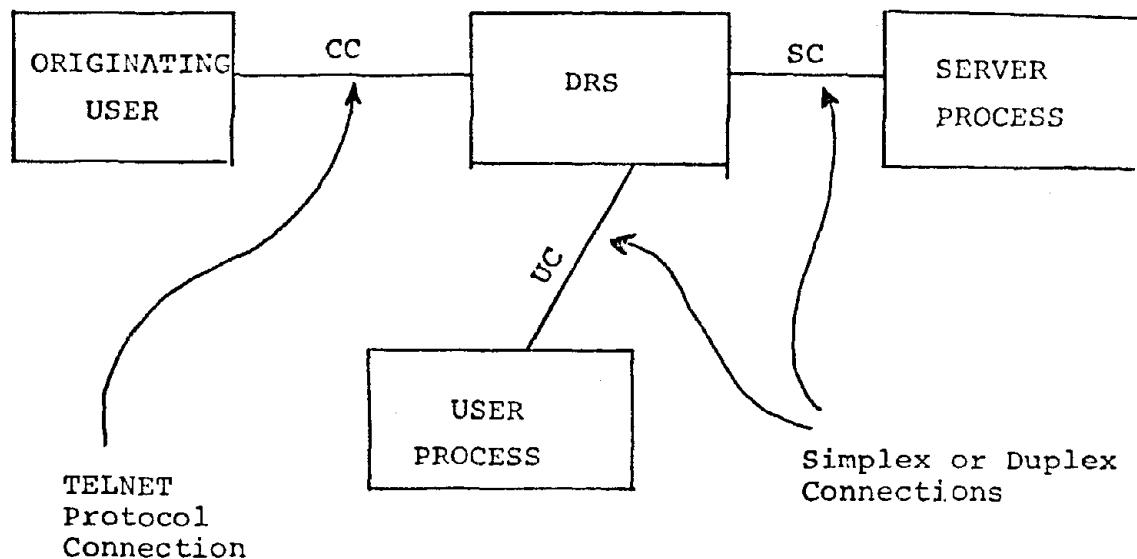


Figure 1. DRS Network Connections

CONNECTION PROTOCOLS AND MESSAGE FORMATS

Over a control connection the dialog is directly between an originating user and the DRS. Here the user is defining forms or assigning predefined forms to connections for reformatting.

The user connects to the DRS via the standard initial connection protocol (ICP). Rather than going through a logger, the user calls on a particular socket on which the DRS always listens.* DRS switches the user to another socket pair.

Messages sent over a control connection are of the types and formats specified for TELNET. (The data type code should specify ASCII -- the default.) Thus, a user at a terminal should be able to connect to a DRS via his local TELNET, for example, as shown in Fig. 2.

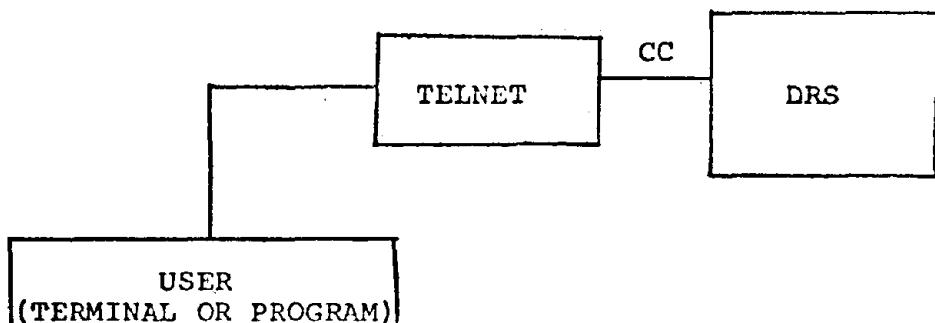


Figure 2. A TELNET Connection to DRS

When a user connects to DRS he supplies a six-character user ID (UID) as a qualifier to guarantee the uniqueness of his form names. He will initially have the following commands:

1. DEFFORM (form)
2. ENDFORM (form)

These two commands define a form, the text of which is chronologically entered between them.

*Experimental socket numbers will be published later.

The form is stored in the DRS local file system.

3. PURGE (form)

The named form, as qualified by the current UID, is purged from the DRS file system.

4. LISTNAMES (UID)

The unqualified names of all forms assigned to UID are returned.

5. LISTFORM (form)

The source text of a named form is returned.

6. DUPLEXCONNECT (user site, user receive socket, user method, server site, server receive socket, server method, user-to-server form name, server-to-user form name)

A duplex connection is made between two processes using the receive sockets and the sockets one greater. Method is defined below. The forms define the transformations on these connections.

7. SIMPLEXCONNECT (user site, user socket, user method, server site, server socket, server method, form)

A simplex connection is made between the two sockets as specified by method.

8. ABORT (site, receive socket)

The reconfiguration of data is terminated by closing both the UC and SC specified in part in the command.

Either one, both, or neither of the two parties specified in 6 or 7 may be at the same host as the party issuing the request. Sites and sockets specify user and server for the connection. Method indicates the way in which the connection is established.

The following rules apply to these commands:

- 1) Commands may be abbreviated to the minimum number of characters to identify them uniquely.
- 2) All commands should be at the start of a line.
- 3) Parameters are enclosed in parentheses and separated by commas.
- 4) Imbedded blanks are ignored.
- 5) The parameters are:

form name	1-6 characters
UID	1-6 characters
Site	1-2 characters specifying the hexadecimal host number
Socket	1-8 characters specifying the hexadecimal socket number
Method	A single character

- 6) Method has the following values:

C	The site/socket is already connected to the DRS as a dummy control connection (should not be the real control connection).
I	Connect via the standard ICP (does not apply to SIMPLEXCONNECT).
D	Connect directly via STR, RTS.

The DRS will make at least the following minimal responses to the user:

- 1) A positive or negative acknowledgment after each line (CR/LF)
- 2) If a form fails or terminates
TERMINATE, ASCII Host # as hex, ASCII Socket # as hex,
ASCII Return Code as decimal
thus identifying at least one end of the connection.

EXAMPLE CONNECTION CONFIGURATIONS

There are basically two modes of DRS operation: 1) the user wishes to establish a DRS UC/SC connection(s) between two programs and 2) the user wants to establish the same connection(s) where he (his terminal) is at the end of the UC or the SC. The latter case is appropriate when the user wishes to interact from his terminal with the serving process (e.g., a logger).

In the first case (Fig. 1, where the originating user is either a terminal or a program) the user issues the appropriate CONNECT command. The UC/SC can be simplex or duplex.

The second case has two possible configurations, shown in Figs. 3 and 4.

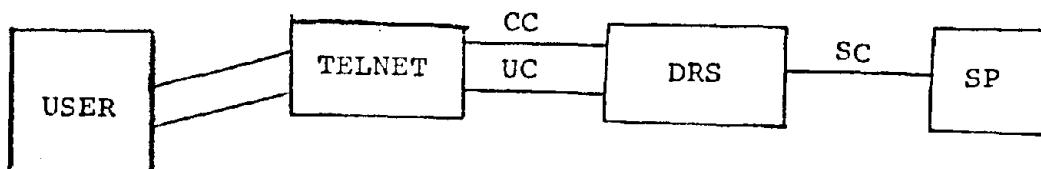


Figure 3. Use of Dummy Control Connection

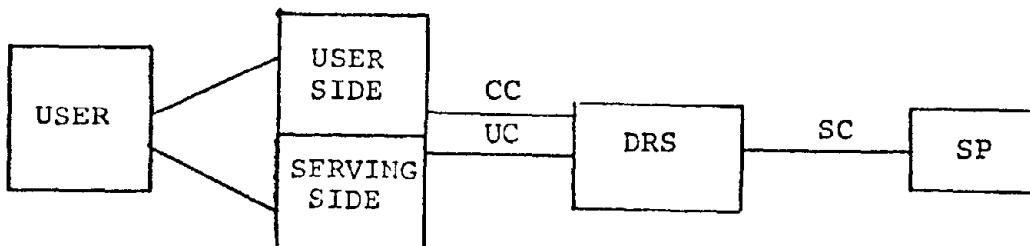


Figure 4. Use of Server TELNET

In Fig. 3 the user instructs his TELNET to make two duplex connections to DRS. One is used for control information (the CC) and the other is a dummy. When he issues

the CONNECT he references the dummy duplex connection (UC) using the "already connected" option.

In Fig. 4 the user has his TELNET (user side) call the DRS. When he issues the CONNECT the DRS calls the TELNET (server side) which accepts the call on behalf of the console. This distinction is known only to the user since to the DRS the configuration in Fig. 4 appears identical to that in Fig. 1. Two points should be noted:

- 1) TELNET protocol is needed only to define forms and direct connections. It is not required for the using and serving processes.
- 2) The using and serving processes need only a minimum of modification for Network use, i.e., an NCP interface.

III. THE FORM MACHINE

INPUT/OUTPUT STREAMS AND FORMS

This section describes the syntax and semantics of forms that specify the data reconfigurations. The Form Machine gets an input stream, reformats the input stream according to a form describing the reconfiguration, and emits the reformatted data as an output stream.

In reading this section it will be helpful to envision the application of a form to the data stream as depicted in Fig. 5. An input stream pointer identifies the position of data (in the input stream) that is being analyzed at any given time by a part of the form. Likewise, an output stream pointer locates data being emitted in the output stream.

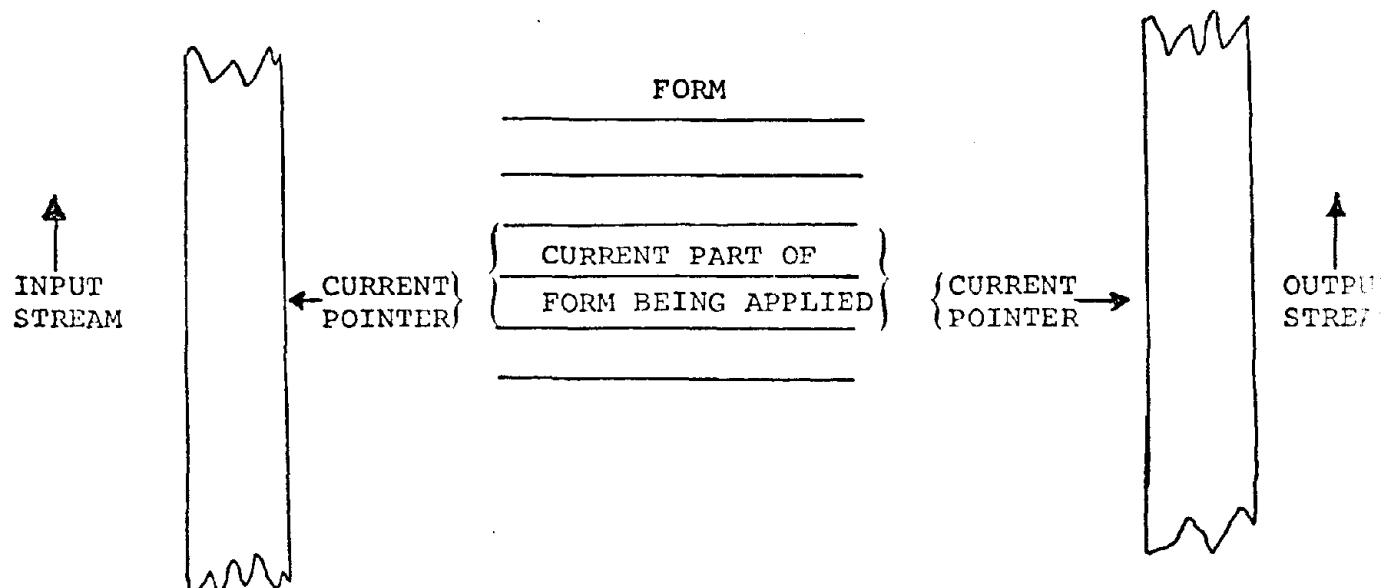


Figure 5. Application of Form to Data Streams

FORM MACHINE BNF SYNTAX

```
form          ::= rule | rule form
rule          ::= label inputstream outputstream ;
label         ::= INTEGER | null
inputstream   ::= terms | null
terms         ::= term | terms , term
outputstream  ::= : terms | null
term          ::= identifier | identifier descriptor |
                  descriptor | comparator
identifier    ::= an alpha character followed by 0 to 3
                  alphameric
descriptor    ::= (replicationexpression , datatype ,
                  valueexpression , lengthexpression control)
comparator   ::= (value connective value control) |
                  (identifier .<= control)
replicationexpression ::= # | arithmeticexpression | null
datatype      ::= B | Ø | X | E | A
valueexpression ::= value | null
lengthexpression ::= arithmeticexpression | null
connective    ::= .LE. | .LT. | .GE. | .GT. | .EQ. | .NE.
value          ::= literal | arithmeticexpression
arithmeticexpression ::= primary | primary operator
                      arithmeticexpression
primary        ::= identifier | L(identifier) | V(identifier) |
                  INTEGER
operator       ::= + | - | * | /
literal        ::= literaltyp "string"
literaltyp    ::= B | Ø | X | E | A
string         ::= from 0 to 256 characters
control        ::= : options | null
options         ::= S(where) | F(where) | U(where) |
                  S(where) , F(where) |
                  F(where) , S(where)
where          ::= arithmeticexpression | R(arithmeticexpression)
```

ALTERNATE SPECIFICATION OF FORM MACHINE SYNTAX

```

form          ::= {rule}^∞
rule          ::= {INTEGER}^1_o {terms}^1_o {:terms}^1_o ;
terms         ::= term {,term}^∞_o
term          ::= identifier | {identifier}^1_o
                  descriptor | comparator
descriptor    ::= ({replicationexpression}^1_o , datatype ,
                  {value}^1_o , {arithmeticexpression}^1_o {:options}^1_o ) |
comparator   ::= (value connective value {:options}^1_o ) |
                  (identifier .<=. value {:options}^1_o )
connective   ::= .LE. | .LT. | .GE. | .GT. | .EQ. | .NE.
replicationexpression ::= # | arithmeticexpression
datatype     ::= B | Ø | X | E | A
value         ::= literal | arithmeticexpression
arithmeticexpression ::= primary {operator primary}^∞_o
operator      ::= + | - | * | /
primary       ::= identifier | L(identifier) |
                  V(identifier) | INTEGER
literal       ::= literaltyp " {CHARACTER}^256_o "
literaltyp   ::= B | Ø | X | A | E
options        ::= S(where) {,F(where)}^1_o |
                  F(where) {,S(where)}^1_o | U(where)
where         ::= arithmeticexpression |
                  R(arithmeticexpression)
identifier    ::= ALPHABETIC {ALPHAMERIC}^3_o

```

FORMS

A form is an ordered set of rules.

form ::= rule | rule form

The current rule is applied to the current position of the input stream. If the (input stream part of a) rule fails to correctly describe the contents of the current input then another rule is made current and applied to the current position of the input stream. The next rule to be made current is either explicitly specified by the current term in the current rule or it is the next sequential rule by default.*

If the (input stream part of a) rule succeeds in correctly describing the current input stream, then some data may be emitted at the current position in the output stream according to the rule. The input and output stream pointers are advanced over the described and emitted data, respectively, and the next rule is applied to the now current position of the input stream.

Application of the form is terminated when an explicit return (R(arithmeticeexpression)) is encountered in a rule. The user and server connections are closed and the return code (arithmeticeexpression) is sent to the originating user.

RULES

A rule is a replacement, comparision, and/or an assignment operation of the form shown below.

rule ::= label inputstream outputstream ;

* Flow of control is more fully described under TERM AND RULE SEQUENCING.

A label is the name of a rule and it exists so that the rule may be referenced elsewhere in the form for explicit rule transfer of control. Labels are of the form below.

label ::= INTEGER | null

The optional integer labels are in the range $0 \leq$ INTEGER ≤ 9999 . The rules need not be labeled in ascending numerical order.

TERMS

The inputstream (describing the input stream to be matched) and the outputstream (describing data to be emitted in the output stream) consist of zero or more terms and are of the form shown below.

```
inputstream ::= terms | null
outputstream ::= :terms | null
terms       ::= term | terms , term
```

Terms are of one of four formats as indicated below,

```
term ::= identifier | identifier descriptor |
        descriptor | comparator
```

Term Format 1

The first term format is shown below.

identifier

The identifier is a symbolic reference to a previously identified term (term format 2) in the form. It takes on the same attributes (value, length, type) as the term by that name. Term format 1 is normally used to emit data in the output stream.

Identifiers are formed by an alpha character followed by 0 to 3 alphabetic characters.

Term Format 2

The second term format is shown below.

identifier descriptor

Term format 2 is generally used as an input stream term but can be used as an output stream term.

A descriptor is defined as shown below.

```
descriptor ::= (replicationexpression, datatype,  
                valueexpression, lengthexpression  
                control)
```

The identifier is the symbolic name of the term in the usual programming language sense. It takes on the type, length, value, and replication attributes of the term and it may be referenced elsewhere in the form.

The replication expression, if specified, causes the unit value of the term to be generated the number of times indicated by the value of the replication expression. The unit value of the term (quantity to be replicated) is determined from the data type, value expression, and length expression attributes. The data type defines the kind of data being specified. The value expression specifies a nominal value that is augmented by the other term attributes. The length expression determines the unit length of the term.*

The replication expression is defined below.

```
replicationexpression ::= # | arithmeticexpression | null  
arithmeticexpression ::= primary | primary operator  
                           arithmeticexpression  
operator ::= + | - | * | /  
primary ::= identifier | L(identifier) | V(identifier) |  
           INTEGER
```

The replication expression is a repeat function applied to the combined data type value, and length expressions. It

* See the IBM SRL Form C28-6514 for a similar interpretation of the pseudo instruction, define constant, after which the descriptor was modeled.

expresses the number of times that the nominal value is to be repeated.

The terminal symbol # means an arbitrary replication factor. It must be explicitly terminated by a match or non-match to the input stream. This termination may result from the same or the following term.

A null replication expression has the value of one. Arithmetic expressions are evaluated from left-to-right with no precedence.

The L(identifier) is a length operator that generates a 32-bit binary integer corresponding to the length of the term named. The V(identifier) is a value operator that generates a 32-bit binary integer corresponding to the value of the term named. (See Restrictions and Interpretations of Term Functions.) The value operator is intended to convert character strings to their numerical correspondants.

The data type is defined below.

datatype ::= B | Ø | X | E | A

The data type describes the kind of data that the term represents.

<u>Data Type</u>	<u>Meaning</u>	<u>Unit Length</u>
B	Bit string	1 bit
Ø	Bit string	3 bits
X	Bit string	4 bits
E	EBCDIC character	8 bits
A	Network ASCII character	8 bits

* It is expected that additional data types, such as floating point and user-defined types, will be added as needed.

The value expression is defined below.

```
valueexpression ::= value | null  
value ::= literal | arithmeticexpression  
literal ::= literaltypes "string"  
literaltypes ::= B | Ø | X | E | A
```

The value expression is the nominal value of a term expressed in the format indicated by the data type. It is repeated according to the replication expression.

A null value expression in the input stream defaults to the data present in the input stream. The data must comply with the datatype attribute, however.

A null value expression generates padding according to Restrictions and Interpretations of Term Functions.

The length expression is defined below.

```
lengthexpression ::= arithmeticexpression | null
```

The length expression states the length of the field containing the value expression.

If the length expression is less than or equal to zero, the term succeeds but the appropriate stream pointer is not advanced. Positive lengths cause the appropriate stream pointer to be advanced if the term otherwise succeeds.

Control is defined under TERM AND RULE SEQUENCING.

Term Format 3

Term format 3 is shown below.

descriptor

It is identical to term format 2 with the omission of the identifier. Term format 3 is generally used in the output stream. It is used in the input stream where input data is to be passed over but not retained for emission or later reference.

Term Format 4

The fourth term format is shown below.

```
comparator ::= (value connective value control) |
               (identifier .<= value control)
value      ::= literal | arithmeticexpression
literal     ::= literaltyp "string"
literaltyp  ::= B | Ø | X | E | A
string      ::= from 0 to 256 characters
connective  ::= .LE. | .LT. | .GE. | .GT. | .EQ. | .NE.
```

The fourth term format is used for assignment and comparison.

The assignment operator .<= assigns the value to the identifier. The connectives have their usual meaning. Values to be compared must have the same type and length attributes or an error condition arises and the form fails.

The Application of a Term

The elements of a term are applied by the following sequence of steps.

1. The data type, value expression, and length expression together specify a unit value, call it x.
2. The replication expression specifies the number of times x is to be repeated. The value of the concatenated xs becomes y of length L.

3. If the term is an input stream term then the value of y of length L is tested with the input value beginning at the current input pointer position.
4. If the input value satisfies the constraints of y over length L then the input value of length L becomes the value of the term.

In an output stream term, the procedure is the same except that the source of input is the value of the term(s) named in the value expression and the data is emitted in the output stream.

The above procedure is modified to include a one term look-ahead where replicated values are of indefinite length because of the arbitrary symbol, #.

Restrictions and Interpretations of Term Functions

1. Terms having indefinite lengths because their values are repeated according to the # symbol, must be separated by some type-specific data such as a literal.*
2. Truncation and padding is as follows:
 - a) Character to character (A ↔ E) conversion is left-justified and truncated or padded on the right with blanks.
 - b) Character to numeric and numeric to numeric conversations are right-justified and truncated or padded on the left with zeros.
 - c) Numeric to character conversation is right-justified and left-padded with blanks.
3. The following are ignored in a form definition over the control connection.
 - a) TELNET control characters.
 - b) Blanks except within quotes.
 - c) /* string */ is treated as comments except within quotes.
4. The following defaults prevail where the term part is omitted.
 - a) The replication expression defaults to one.
 - b) # in an output stream term defaults to one.
 - c) The value expression of an input stream term defaults to the value found in the input stream, but the input stream must conform to data type

* A literal isn't specifically required, however. An arbitrary number of ASCII characters could be terminated by a non-ASCII character.

- and length expression. The value expression of an output stream term defaults to padding only.
- e) The length expression defaults to the size of the quantity determined by the data type and value expression.
 - f) Control defaults to the next sequential term if a term is successfully applied; else control defaults to the next sequential rule. If where evaluates to an undefined label the form fails.
5. Arithmetic expressions are evaluated left-to-right with no precedence.
 6. The following limits prevail.
 - a) Binary lengths are \leq 32 bits
 - b) Character strings are \leq 256 8-bit characters
 - c) Identifier names are \leq 4 characters
 - d) Maximum number of identifiers is \leq 256
 - e) Label integers are \geq 0 and \leq 9999
 7. Value and length operators product 32-bit binary integers. The value operator is currently intended for converting A or E type decimal character strings to their binary correspondants. For example, the value of E'12' would be 0.....01100. The value of E'AB' would cause the form to fail.

TERM AND RULE SEQUENCING

Sequencing may be explicitly controlled by including control in a term.

```
control ::= :options | null
options ::= S(where) | F(where) | U(where)
           S(where) , F(where) |
           F(where) , S(where)

where    ::= arithmeticexpression | R(arithmeticexpression)
```

S, F, and U denote success, fail, and unconditional transfers, respectively. Where evaluates to a rule label, thus transfer can be effected from within a rule (at the end of a term) to the beginning of another rule. R means terminate the form and return the evaluated expression to the initiator over the control connection (if still open).

If terms are not explicitly sequenced, the following defaults prevail.

- 1) When a term fails go to the next sequential rule.
- 2) When a term succeeds go to the next sequential term within the rule.
- 3) At the end of a rule, go to the next sequential rule.

Note in the following example, the correlation between transfer of control and movement of the input pointer.

```
1 XYZ(,B,,8:S(2),F(3)) : XYZ ;  
2 . . . . .  
3 . . . . .
```

The value of XYZ will never be emitted in the output stream since control is transferred out of the rule upon either success or failure. If the term succeeds, the 8 bits of input will be assigned as the value of XYZ and rule 2 will then be applied to the same input stream data. That is, since the complete left hand side of rule 1 was not successfully applied, the input stream pointer is not advanced.

IV. EXAMPLES

REMARKS

The following examples (forms and also single rules) are simple representative uses of the Form Machine. The examples are expressed in a term-per-line format only to aid the explanation. Typically, a single rule might be written as a single line.

FIELD INSERTION

To insert a field, separate the input into the two terms to allow the inserted field between them. For example, to do line numbering for a 121 character/line printer with a leading carriage control character, use the following form.

```
(NUMB.<=.1);      /*initialize line number counter to one*/
1 CC(,E,,1:F(R(99))), /*pick up control character and save
                        as CC*/
                    /*return a code of 99 upon exhaustion*/
LINE(,E,,121 : F(R(98)))/*save text as LINE*/
:CC,                /*emit control character*/
(,E,NUMB,2),        /*emit counter in first two columns*/
(,E,E".",1),        /*emit period after line number*/
(,E,LINE,117),      /*emit text, truncated in 117 byte field*/
(NUMB.<=.NUMB+1:U(1)); /*increment line counter and go to
                        rule one*/;;
```

DELETION

Data to be deleted should be isolated as separate terms on the left, so they may be omitted (by not emitting them) on the right.

```
(,B,,8),          /*isolate 8 bits to ignore*/
SAVE(,A,,10)       /*extract 10 ASCII characters from
                        input stream*/
```

```
:,(E,SAVE,);      /*emit the characters in SAVE as EBCDIC
                      characters whose length defaults to the
                      length of SAVE, i.e., 10, and advance to
                      the next rule*/
```

In the above example, if either input stream term fails, the next sequential rule is applied.

VARIABLE LENGTH RECORDS

Some devices, terminals and programs generate variable length records. The following rule picks up variable length EBCDIC records and translates them to ASCII.

```
CHAR(#,E,,1),      /*pick up all (an arbitrary number of)
                      EBCDIC characters in the input stream*/
(,X,X"FF",2)      /*followed by a hexadecimal literal,
                      FF (terminal signal)*/
:(,A,CHAR,),       /*emit them as ASCII*/
(,X,X"25",2);     /*emit an ASCII carriage return*/
```

STRING LENGTH COMPUTATION

It is often necessary to prefix a length field to an arbitrarily long character string. The following rule prefixes an EBCDIC string with a one-byte length field.

```
Q(#,E,,1),          /*pick up all EBCDIC characters*/
TS(,X,X"FF",2)      /*followed by a hexadecimal literal, FF*/
:(,B,L(Q)+2,8),     /*emit the length of the characters
                      plus the length of the literal plus
                      the length of the count field itself,
                      in an 8-bit field*/
Q,                  /*emit the characters*/
TS;                /*emit the terminal*/
```

TRANSPOSITION

It is often desirable to reorder fields, such as the following example.

Q(,E,,20), R(,E,,10) , S(,E,,15), T(,E,,5) : R, T, S, Q ;

The terms are emitted in a different order.

CHARACTER PACKING AND UNPACKING

In systems such as HASP, repeated sequences of characters are packed into a count followed by the character, for more efficient storage and transmission. The first form packs multiple characters and the second unpacks them.

```
/*form to pack EBCDIC streams*/
/*returns 99 if OK, input exhausted*/
/*returns 98 if illegal EBCDIC*/
/*lock for terminal signal FF which is not a legal EBCDIC*/
/*duplication count must be 0-254*/
1 (,X,X"FF",2 : S(R(99))) ;
/*pick up an EBCDIC char*/
CHAR(,E,,1) ;
/*get identical EBCDIC chars*/
LEN(#,E,CHAR,1)
/*emit the count and the char*/
: (,B,L(LEN)+1,8), CHAR, (:U(1));
/*end of form*/;

/*form to unpack EBCDIC streams*/
/*look for terminal*/
1 (,X,X"FF",2 : S(R(99))) ;
/*emit character the number of times indicated*/
/*by the count, in a field the length indicated*/
/*by the counter contents*/
CNT(,B,,8), CHAR(,E,,1) : (CNT,E,CHAR,1:U(1));
/*failure of form*/
(:U(R(98))) ;
```

