Inter-Office Memorandum

To	Communication Protocols	Date	July 2, 1978
From	Ed Taft	Location	Palo Alto
Subject	Pup Telnet Protocol	Organization	PARC/CSL

XEROX

Filed on: [Maxc1]<Pup>Telnet.press

This is a revision of a memo of the same title dated November 11, 1975. Only minor changes have been made.

Telnet is a protocol designed for terminal communication--either process-to-terminal or terminal-to-terminal. It is based entirely on the Pup Byte Stream Protocol (BSP), described elsewhere.

Telnet is intended for providing a sequential communication path between (usually) a user terminal and a server process in a manner similar to that provided by conventional computer terminals. It is specifically not intended for more elaborate forms of communication such as display editing or interactive graphics. Such applications are better (more efficiently) implemented by means of appropriate special protocols.

The conventions we have adopted are based on the Arpanet Telnet protocol for control of terminal connections, omitting most of its provisions (particularly the more complicated ones) for which there is no forseeable need within the PARC environment.

All Telnet control commands consist of a BSP Mark byte (of which there are 256 possible types) followed by zero or more arguments given in data bytes immediately following the Mark.

All bytes that are not part of such a control sequence are treated as ordinary data bytes. Typically, Telnet data will consist only of ASCII characters in the range 0-177 (octal), but we leave the door open to the possibility of 8-bit binary terminal I/O for such purposes as extended character sets.

The control functions defined at present are the following:

Data Mark

A Data Mark (DM, Mark type 1), in conjunction with the BSP Interrupt signal, is used to perform the "Synch" operation. A Telnet sender (user or server) desiring to generate a "Synch" signal inserts a DM in the byte stream and simultaneously transmits an Interrupt. The Interrupt, not being subject to BSP flow control, reaches the receiver even if the byte stream is otherwise clogged up due to allocation or buffering considerations. Upon receipt of the Interrupt, the receiver immediately discards already processed data as appropriate (e.g. clearing terminal buffers), and then reads the byte stream until the DM is encountered. While doing so, the receiver discards all data *except* Telnet control commands and any other "interesting" signals it may choose to act upon.

In Telnet terminal communication, the "Synch" signal is used for two similar purposes. First, it may be generated by the Telnet server when it is called upon to "clear the terminal output buffer".

For example, the Tenex CFOBF JSYS, when executed for a network terminal, generates a "Synch" and thereby causes the user’s terminal output buffer to be cleared and all data already in the byte stream to be flushed. Second, it may be generated by the Telnet user upon discovering a "terminal input buffer full" condition that is even preventing panic signals (e.g., control-C) from reaching the server because the byte stream is blocked due to lack of allocation.

Terminal Parameters

By means of Terminal Parameter commands, a Telnet user may inform the server of terminal characteristics such as Line Width (Mark type 2), Page Length (Mark type 3), and Terminal Type (Mark type 4). Each of these is followed by an argument byte specifying the value of the parameter as an 8-bit binary number. For Line Width and Page Length, this value is simply the maximum number of characters per line and lines per page, with the value zero defined to mean "unknown" or "inapplicable" (e.g., due to variable width characters). For Terminal Type, the argument is a Tenex terminal type number, which at present may be any one of the following:

0	Model 33 TTY (no Tab, Formfeed, or lower case).
1	Model 35 TTY (Tab and Formfeed, no lower case).
2	Model 37 TTY (Tab, Formfeed, lower case).
3	TI (lower case, funny padding, no Tab or Formfeed).
7	NVT (no Tab or Formfeed, lower case, no padding whatsoever).
8	LA30, GE Terminet-300 (special padding).
9	TI-733 (slower CR than normal TI).
10	Scope (pause at end of page).

It is conceded that this aspect of our Telnet protocol is rather strongly oriented toward Tenex conventions. The Arpanet Telnet experience has demonstrated that it is impractical to attempt to do Telnet terminal handling in a "completely general" fashion. We propose simply putting off designing a more general-purpose terminal type specification protocol until such time as we require the added generality. At the present time, we use the Telnet protocol only for communicating with Tenex and Tenex-like servers.

Timing Mark

At any time, either the user or the server may place a Timing Mark (Mark type 5) in the outgoing byte stream. The receiver of the Timing Mark is expected to respond with a Timing Mark Reply (Mark type 6) when all data preceding the Timing Mark has been "completely processed". By "completely processed", we mean that the data has been in some sense accepted and logically acted upon by the user or server program (as opposed, say, to simply being placed in terminal buffers). Hence, a server Telnet should send the Timing Mark Reply only when the server process accepts (or is prepared to accept) data following the incoming Timing Mark (i.e., is "blocked on input"). Similarly, a user Telnet should send the Reply only when all data prior to the incoming Timing Mark has actually been printed or displayed on the user’s terminal.

The intent of this mechanism is to permit a process to synchronize or correlate the incoming and outgoing data. When the initiator of the Timing Mark encounters the Timing Mark reply, it knows that all data following the Reply was generated (by the other process) *after* the time at which the original Timing Mark was sent.

Typical uses of the Timing Mark mechanism may be illustrated as follows. An Alto user Telnet makes a connection to Maxc, transmits a prepared typescript terminated by the command "Logout" and a Timing Mark, then waits for the Timing Mark Reply before terminating the connection. Without the Timing Mark mechanism, it is difficult for the user Telnet to determine the correct moment at which to close the connection, since in general there is some buffering at the server end. If the connection is closed too early, the server process may get "detached" before having read all the input.

The Timing Mark mechanism may also be put to use by the server Telnet in the following way. A program that detects some logical error will typically execute a "clear input buffer" operation (CFIBF in Tenex) to clear characters that might have been typed ahead by the user before the error occurred. But when the terminal is at the other end of a network connection, the server Telnet has no way of ascertaining the point in the incoming byte stream corresponding (chronologically) with the time the error occurred. The Timing Mark facility may be used to solve this problem. When the program executes a "clear input buffer" (presumably after generating some sort of error message), the server Telnet should drop a Timing Mark in the outgoing byte stream (appended to the error message), then discard data bytes from the incoming stream until the Timing Mark Reply is encountered.

Telnet Operations Not Specified

The Arpanet Telnet protocol includes a large number of operations that have not been adopted in the Pup Telnet protocol. These include the following:

1. Network standard representations for common operations, such as EL (Erase Line), IP (Interrupt Process), etc. The justification for this is that within the PARC environment, we do not mind adopting the conventions of whatever server host we happen to be connected to. If users turn out to be unhappy about this, these functions are easy to add.

2. Negotiation about server or user echoing. Since we don't have any half-duplex terminals and always operate over short distances, it seems reasonable to adopt server echoing as the PARC standard.

3. The negotiation mechanism itself. The Arpanet Telnet "option negotiation" protocol is quite complicated, and experience has shown it to be very difficult to implement correctly. When we find something to negotiate about, we should perhaps try to design something simpler.

4. More elaborate interactions such as the Telnet Reconnection protocol and the Remote-Controlled Transmission and Echoing protocol (again, due to lack of any forseeable need for these mechanisms).