

Mark Crispin
SU-AI
7 October 1977

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
 \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
 \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
 \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000
 \000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000\000

[illegible]

[illegible]

[illegible]

INPUT -- THE INTELLIGENT TERMINAL PROTOCOL

Note: only the parts of the intelligent terminal protocol relevant to SUPDUP are discussed here. For more information, read ITS TTY.

CHARACTER SETS

There are two character sets available for use with SUPDUP; the 7-bit character set of standard ASCII, and the 12-bit character set of extended ASCII. Extended ASCII has 5 high order or "bucky" bits on input and has graphics for octal 000-037 and 177 (see the section entitled "Stanford/ITS character set" for more details). The two character sets are identical on output since the protocol specifies that the host should never send the standard ASCII formatting characters (ie, TAB, LF, VT, FF, CR) as formatting characters; the characters whose octal values are the same as these formatting characters are never output unless the user job has these characters enabled (setting %TOSAI and %TOSA1 generally does this).

Input differs dramatically between the 7-bit and 12-bit character sets. In the 7-bit character set, all characters input whose value is 037 octal or less are assumed to be (ASCII) control characters. In the 12-bit character set, there are 5 "bucky" bits which may be attached to the character. The two most important of these are CONTROL and META, which form a 9-bit character set. TOP is used to distinguish between printing graphics in the extended character set and ASCII controls. The other two are reserved and should be ignored. Since both 7-bit and 12-bit terminals are commonly in use, 0001, 0301, and 0341 are considered to be <CONTROL>A on input by most programs, while 4001 is considered to be downwards arrow.

MAPPING BETWEEN CHARACTER SETS

Many programs and hosts do not process 12-bit input. In this case, 12-bit input is folded down to 7-bit as follows: TOP and META are discarded. If CONTROL is on, then if the 7-bit part of the character specifies a lower case alphabetic it is converted to upper case; then if the 7-bit part is between 077 and 137 the 100 bit is complemented or if the 7-bit part is 040 the 040 bit is subtracted (that's right, <CONTROL>? is converted to [RUBOUT] and <CONTROL>[SPACE] is converted to [NULL]). In any case the CONTROL bit is discarded, and the remainder is treated as a 7-bit ASCII character. It should be noted that in this case downwards arrow is read by the program as standard ASCII <CONTROL>A.

Servers which expect 12-bit input and are told to use the 7-bit character set should do appropriate unfolding from the 7-bit character set to 12-bit. It is up to the individual server to decide upon the unfolding scheme. On ITS, user programs that use the 12-bit character set generally have an alternative method for 7-bit; this often takes the form of prefix characters indicating that the next character should be "controllified" or "metized", etc.

[illegible]

BUCKY BITS

Name	Value	Description
%TXTOP	4000	This character has the [TOP] key depressed.
%TXSFL	2000	Reserved, must be zero.
%TXSFT	1000	Reserved, must be zero.
%TXMTA	400	This character has the [META] key depressed.
%TXCTL	200	This character has the [CONTROL] key depressed.
%TXASC	177	The ASCII portion of the character

is sent as three bytes. The first byte is always 034 octal (that is why 034 must be quoted). The next byte contains the "bucky bits", ie, the %TXTOP through %TXCTL bits, shifted over 7 bits (ie, %TXTOP becomes 20) with the 100 bit on. The third byte contains the %TXASC part of the character. Hence the character <CONTROL><META>[LINE FEED] is sent as 034 103 012.

The intelligent terminal protocol also is involved when a network interrupt (INR/INS) is received by the user program. The user program should increment a count of received network interrupts when this happens. It should not do any output, and if possible abort any output in progress, if this count is greater than zero (NOTE: the program MUST allow for the count to go less than zero).

Since the server no longer knows where the cursor is, it suspends all output until the user informs it of the cursor position. This also gives the server an idea of how much was thrown out in case it has to have some of the aborted output displayed at a later time. The user program does this when it receives a %TDORS from the server. When this happens it should decrement the "number of received network interrupts" count described in the previous paragraph and then send 034 followed by 020, the vertical position, and the horizontal position of where the cursor currently is located on the user's screen.

[illegible]

Display output is somewhat simpler. Codes less than 200 octal are printing characters and are displayed on the terminal (see the section describing the "Stanford/ITS character set"). Codes greater than or equal to 200 (octal) are known as "%TD codes", so called since their names begin with %TD. The %TD codes that are relevant to SUPDUP operation are listed here. Any other code received should be ignored, although a bug report might be sent to the server's maintainers. Note that the normal ASCII formatting characters (011 - 015) do NOT have their formatting sense under SUPDUP and should not occur at all unless the Stanford/ITS extended ASCII character set is in use (ie, %TOSAI is set in the TTYOPT word).

%TD code	Value	Meaning
%TDMOV	200	General cursor position code. Followed by four bytes; the first two are the "old" vertical and horizontal positions and may be ignored. The next two are the new vertical and horizontal positions. The cursor should be moved to this position.

On printing consoles (non %TOMVU), the old vertical position may differ from the true vertical position; this can occur when scrolling. In this case, the user program should set its idea of the old vertical position to what the %TDMOV says and then proceed. Hence a %TDMOV with an old vpos of 20. and a new vpos of 22. should always move the "cursor" down two lines. This is used to prevent the vertical position from becoming infinite.

%TDMV1	201	An internal cursor motion code which should not be seen; but if it is, it has two argument bytes after it and should be treated the same as %TDMV0.
--------	-----	---

%TDEOF	202	Erase to end of screen. This is an optional function since many terminals do not support this. If the terminal does not support this function, it should be treated the same as %TDEOL.
--------	-----	---

%TDEOF does an erase to end of line, then erases all lines lower on the screen than the cursor. The cursor does not move.

%TDEOL	203	Erase to end of line. This erases the character position the cursor is at and all positions to the right on the same line. The cursor does not move.
--------	-----	--

[illegible]

[illegible]

%TD code	Value	Meaning
%TDDLf	204	Clear the character position the cursor is on. The cursor does not move.
%TDCRL	207	If the cursor is not on the bottom line of the screen, move cursor to the beginning of the next line and clear that line. If the cursor is at the bottom line, scroll up.
%TDNOP	210	No-op; should be ignored.
%TDORS	214	Output reset. This code serves as a data mark for aborting output much as IAC DM does in the ordinary TELNET protocol.
%TDQOT	215	Quotes the following character. This is used when sending 8-bit codes which are not %TD codes, for instance when loading programs into an intelligent terminal. The following character should be passed through intact to the terminal.
%TDFS	216	Non-destructive forward space. The cursor moves right one position; this code will not be sent at the end of a line.
%TDMV0	217	General cursor position code. Followed by two bytes; the new vertical and horizontal positions.
%TDCLR	220	Erase the screen. Home the cursor to the top left hand corner of the screen.
%TDBEL	221	Generate an audio tone, bell, whatever.
%TDILP	223	Insert blank lines at the cursor; followed by a byte containing a count of the number of blank lines to insert. The cursor is unmoved. The line the cursor is on and all lines below it move down; lines moved off the bottom of the screen are lost.
%TDDLp	224	Delete lines at the cursor; followed by a count. The cursor is unmoved. The first line deleted is the one the cursor is on. Lines below those deleted move up. Newly-created lines at the bottom of the screen are blank.

[illegible]

[illegible]

%TD code	Value	Meaning
%TDICP	225	Insert blank character positions at the cursor; followed by a count. The cursor is unmoved. The character the cursor is on and all characters to the right on the current line move to the right; characters moved off the end of the line are lost.
%TDDCP	226	Delete characters at the cursor; followed by a count. The cursor is unmoved. The first character deleted is the one the cursor is on. Newly-created characters at the end of the line are blank.
%TDBOW	227	Display black characters on white screen. HIGHLY OPTIONAL.

[illegible]

[illegible]

[illegible]

[illegible]

\000\000\000

[illegible]