Network Working Group   Internet Engineering Task Force
Request for Comments: 1116   Telnet Linemode Working Group
D. Borman, Editor
Cray Research, Inc.
August 1989

Telnet Linemode Option

Status of this Memo

This RFC describes a proposed elective standard for the Internet community. Hosts on the Internet that support Linemode within the Telnet protocol are expected to adopt and implement this standard. Distribution of this memo is unlimited.

Overview

Linemode Telnet is a way of doing terminal character processing on the client side of a Telnet connection. While in Linemode with editing enabled for the local side, network traffic is reduced to a couple of packets per command line, rather than a couple of packets per character typed. This is very useful for long delay networks, because the user has local response time while typing the command line, and only incurs the network delays after the command is typed. It is also useful to reduce costs on networks that charge on a per packet basis.

Table of Contents

1. Command Names and Codes

```
       LINEMODE        34
           MODE             1
               EDIT             1
               TRAPSIG          2
               MODE_ACK         4
           FORWARDMASK      2
           SLC              3
               SLC_SYNCH        1
               SLC_BRK          2
               SLC_IP           3
               SLC_AO           4
               SLC_AYT          5
               SLC_EOR          6
               SLC_ABORT        7
               SLC_EOF          8
               SLC_SUSP         9
               SLC_EC          10
               SLC_EL          11
               SLC_EW          12
               SLC_RP          13
               SLC_LNEXT       14
               SLC_XON         15
               SLC_XOFF        16
               SLC_FORW1       17
               SLC_FORW2       18

               SLC_DEFAULT      3
               SLC_VALUE        2
               SLC_CANTCHANGE   1
               SLC_NOSUPPORT    0
               SLC_LEVELBITS    3

               SLC_ACK        128
               SLC_FLUSHIN     64
               SLC_FLUSHOUT    32
       EOF            236
       SUSP           237
```

ABORT 238

2. Command Meanings

2.1 The LINEMODE function

IAC WILL LINEMODE

The sender of this command REQUESTS permission to begin sub-negotiation of the editing/signaling status. This should only be sent by the client side of the connection.

IAC WONT LINEMODE

The sender of this command DEMANDS that sub-negotiation of the editing/signaling status not be allowed.

IAC DO LINEMODE

The sender of this command REQUESTS that the remote side begin subnegotiation of the editing/signaling status. This should only be sent by the server side of the connection.

IAC DONT LINEMODE

The sender of this command DEMANDS that the remote side not begin subnegotiation of the editing/signaling status.

2.2 LINEMODE suboption MODE

IAC SB LINEMODE MODE mask IAC SE

The sender of this command CONFIRMS, or REQUESTS permission for, a switch to the mode defined by "mask".

The "mask" is a bit mask of various modes that the connection can be in. Under normal operation, the server side of the connection will initiate mode changes, and the client will confirm the mode changes.

The currently defined modes are:

EDIT   When set, the client side of the connection should process all input lines, performing any editing functions, and only send completed lines to the remote side. When unset, client side should not process any input from the user, and the server side should take care of all character processing that needs to be done.

TRAPSIG When set, the client side should translate appropriate interrupts/signals to their Telnet equivalent. (These would be IP, BRK, AYT, ABORT, EOF, and SUSP.)

When unset, the client should pass interrupts/signals as their normal ASCII values.

FLOW   Logically, this belongs in the "mask". However, this would overlap the Telnet TOGGLE-FLOW-CONTROL option, so the Telnet TOGGLE-FLOW-CONTROL option is used instead. When DO/WILL LINEMODE is negotiated, DO/WILL TOGGLE-FLOW-CONTROL should also be negotiated. See RFC 1080, "Telnet Remote Flow Control", for correct usage.

ECHO   Logically, this belongs in the "mask". However, this would overlap the Telnet ECHO option, so the Telnet ECHO option is used instead. The client side should never negotiate "WILL ECHO". When the server has negotiated "WILL ECHO", the client should not echo data typed by the user back to the user. When the server has negotiated "WONT ECHO", the client is responsible for echoing data typed by the user back to the user. See RFC 857, "Telnet ECHO OPTION" for a complete discussion on the use of the Telnet ECHO option.

When the client side of a connection receives a MODE command, it MUST agree with at least the state of the EDIT and TRAPSIG bits. If a MODE command is received with a mode mask that is currently in use (ignoring the MODE_ACK bit), the MODE command is ignored. If a MODE command is received that is different from the current mode mask, then a reply is sent with either the new mode mask and the MODE_ACK bit set, or a subset of the new mode mask. The only exception is that if the server receives a MODE with either the EDIT or TRAPSIG bits not set, it may set the EDIT and TRAPSIG bits in the response, and if the client receives a MODE with the EDIT or TRAPSIG bits set, it may not clear them in the response.

When a MODE command is received with the MODE_ACK bit set, and the mode is different that what the current mode is, the client will ignore the new mode, and the server will switch to the new mode. This ensures that both sides of the connection will resolve to the same mode. In all cases, a response is never generated to a MODE command that has the MODE_ACK bit set.

2.3 LINEMODE suboption FORWARDMASK

IAC SB LINEMODE DO FORWARDMASK mask0 mask1 ... mask31 IAC SE

The sender of this command request that the other side send any buffered data when any of the ASCII characters defined by the bit

mask are received. Only the side of the connection that sent DO LINEMODE (the server side) may negotiate this. The mask is up to 32 octets long. Each octet represents 8 ASCII character codes. The high order bit of mask0 corresponds to an ASCII code of 0. The low order bit of mask0 corresponds to an ASCII code of 7. The high order bit of mask1 corresponds to an ASCII code of 8. The low order bit of mask1 corresponds to an ASCII code of 15, and so on. The mask list may be terminated before the end of the list, in which case all the rest of the mask octets are assumed to be reset (equal to zero). When the server side is in DONT TRANSMIT-BINARY mode, then only the first 16 octets of the mask (ASCII codes 0 through 127) are used. If any individual octet of the mask is equal to IAC, it must be sent as a double IAC.

IAC SB LINEMODE DONT FORWARDMASK IAC SE

The sender of this command requests that the other side stop using the forward mask to determine when to send buffered data.

IAC SB LINEMODE WILL FORWARDMASK IAC SE

This command is sent in response to a DO FORWARDMASK command. It indicates that the forward mask will be used to determine when to send buffered data.

IAC SB LINEMODE WONT FORWARDMASK IAC SE

This command is sent in response to a DO FORWARDMASK command. It indicates that the forward mask will not be used to determine when to send buffered data.

2.4 LINEMODE suboption SLC, Set Local Characters

The SLC suboption uses a list of octet triplets. The first octet specifies the function, the second octet specifies modifiers to the function, and the third octet specifies the ASCII character for the function.

IAC SB LINEMODE SLC <list of octet triplets> IAC SE

The sender of this command REQUESTS that the list of octet triplets be used to set the local character to be used to send to perform the specified function.

There are four levels that a function may be set to. SLC_NOSUPPORT is the lowest, SLC_CANTCHANGE is the next higher level, SLC_VALUE is above that, and SLC_DEFAULT is the highest level.

If the SLC_LEVELBITS in the second octet are equal to SLC_DEFAULT, then this particular function should use the system default on the other side of the connection.

If the SLC_LEVELBITS in the second octet are equal to SLC_VALUE, then this function is supported, and the current value is specified by the third octet.

If the SLC_LEVELBITS in the second octet are equal to SLC_CANTCHANGE, then this is a function that is supported, but the value for this function, specified in the third octet, cannot be changed.

If the SLC_LEVELBITS in the second octet are equal to SLC_NOSUPPORT, then this particular function is not supported and should be disabled by the other side.

If this is a response to a previous request to change a special character, and we are agreeing to the change, then the SLC_ACK bit must be set in the second octet.

If the SLC_FLUSHIN bit is set in the second octet, then whenever this function is sent, a Telnet "sync" should be sent at the same time to flush the input stream.

If the SLC_FLUSHOUT bit is set in the second octet, then whenever this function is sent, output data should be flushed.

Only the client may send an octet triplet with the first octet equal to zero. In this case, the SLC_LEVELBITS may only be set to SLC_DEFAULT or SLC_VALUE, and the third octet does not matter. When the server receives 0 SLC_DEFAULT 0, it should switch to its system default special character settings, and send all those special characters to the client. When the server receives 0 SLC_VALUE 0, it should just send its current special character settings. Note that if the server does not support some of the editing functions, they should be sent as XXX SLC_DEFAULT 0, rather than as XXX SLC_NOSUPPORT 0, so that the client may choose to use its own values for those functions, rather than have to disable those functions even if it supports them.

If any of the octets in the list of octet triplets is equal to IAC, it must be sent as a double IAC.

When a connection is established, it is the responsibility of the client to either request the remote default values for the special characters, or to send across what all the special characters should be set to.

The function values can be put into two groups; functions that are to be translated to their Telnet equivalents before being sent across the Telnet connection, and functions that are to be recognized and processed locally.

First, we have those characters that are to be mapped into their Telnet equivalents:

SLC_SYNCH Synch. See RFC 854, "TELNET PROTOCOL SPECIFICATION", for a complete description.

SLC_BRK Break. See RFC 854, "TELNET PROTOCOL SPECIFICATION", for a complete description.

SLC_IP Interrupt Process. See RFC 854, "TELNET PROTOCOL SPECIFICATION", for a complete description.

SLC_AO Abort Output. See RFC 854, "TELNET PROTOCOL SPECIFICATION", for a complete description.

SLC_AYT Are You There. See RFC 854, "TELNET PROTOCOL SPECIFICATION", for a complete description.

SLC_EOR End of Record. See RFC 885, "TELNET END OF RECORD OPTION" for a complete description.

SLC_ABORT Abort. See section 2.5 for a complete description.

SLC_EOF End of File. See section 2.5 for a complete description.

SLC_SUSP Suspend. See section 2.5 for a complete description.

Next, we have the locally interpreted functions:

SLC_EC Erase Character. This is the character that is typed to erase one character from the input stream. See RFC 854, "TELNET PROTOCOL SPECIFICATION", for a complete description.

SLC_EL Erase Line. This is the character that is typed to erase the entire contents of the current line of input. See RFC 854, "TELNET PROTOCOL SPECIFICATION", for a complete description.

SLC_EW Erase Word. This is the character that is typed to erase one word from the input stream. When backing up in the input stream, a word is defined

to be (optionally) whitespace (tab or space characters), and a string of characters up to, but not including, whitespace or line delimiters.

SLC_RP	Reprint Line. This is the character that is typed to cause the current line of input to be reprinted, leaving the cursor at the end of the line.

SLC_LNEXT	Literal Next. This is the character that is typed to indicate that the next character is to be taken literally, no character processing should be done with it, and if it is a special character that would normally get mapped into a Telnet option, that mapping should not be done.

SLC_XON	Start Output. This is the character that is sent to resume output to the users terminal.

SLC_XOFF	Stop Output. This is the character that is sent to stop output to the users terminal.

SLC_FORW1	Forwarding character. This is a character that should cause all data currently being buffered, and this character, to be sent immediately.

SLC_FORW2	Forwarding character. This is another character that is to be treated in the same manner as SLC_FORW1.

2.5 New control characters

IAC ABORT

Abort. Similar to "IAC IP", but means only to abort or terminate the process to which the NVT is connected. (The Telnet spec says IP may "suspend, interrupt, abort or terminate" the process.) If a system does not have two methods of interrupting a process, then ABORT and IP should have the same effect.

IAC SUSP

Suspend the execution of the current process attached to the NVT in such a way that another process will take over control of the NVT, and the suspended process can be resumed at a later time. If the receiving system does not support this functionality, it should be ignored.

IAC EOF

End Of File. The recipient should notify the process connected to the NVT that an end of file has been reached. This is intended for systems that support the ability for the user to type in an EOF character at the keyboard.

3. Default Specification

The default specification for this option is:

```
WONT LINEMODE
DONT LINEMODE
```

meaning there will not be any subnegotiation of the mode of the connection.

If WILL LINEMODE is negotiated, the defaults are:

```
IAC SB LINEMODE MODE 0 IAC SE
IAC SB LINEMODE WONT FORWARDMASK IAC SE
```

If DO LINEMODE is negotiated, the defaults are:

```
IAC SB LINEMODE MODE 0 IAC SE
IAC SB LINEMODE DONT FORWARDMASK IAC SE
```

Character values for SLC default to SLC_NOSUPPORT.

4. Motivation

With increasing Telnet usage, it has become apparent that the ability to do command line processing on the local machine and send completed lines to the remote machine is a feature necesary in several environments. First, in the case of a connection over long delay equipment, it is very frustrating to the user to have the echoing of his data take several seconds. Second, some supercomputers, due to their nature, are not good at handling and processing single character input. For these machines, it is better to have the front end computer do the character processing, and leave the supercomputer's cycles available for doing vectorized number crunching.

There have been attempts to make local line editing work within the existing Telnet specs. Indeed, the 4.3 BSD tape includes a version of Telnet that attempts to do this through recognition of the state of the ECHO and SUPRESS-GO-AHEAD options; other implementations do this recognition purely through the ECHO option.

There are problems with both of these methods. Using just the ECHO provides no mechanism to have ECHO to the user turned off, and leave local character processing on, for example, when a user is typing a password.

The usage of the SUPRESS-GO-AHEAD comes from reading into RFC 858, where it states:

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

The reverse reading of this is that without the ECHO option or the SUPPRESS-GO-AHEAD option, you are in line at a time mode, implying local line editing. This has the obvious problem that that is not what the SUPPRESS-GO-AHEAD option is supposed to mean.

Other shortcomings are that the Telnet specification is not rich enough to handle all of the special characters that some of the current operating systems support. For example, the ECHO/SGA implementation supports two ways of interrupting a process, by borrowing the BRK option for the second interrupt. Some implementations have taken the EOR option to send an End-Of-File. Obviously, this is using things for which they were not intended, and the correct solution would be to define new options.

Another problem is that some implementations of line mode buffer up the input until the end of the line, and then send the whole line across, editing characters and all. No local editing of the line has been done.

After examining several implementations, it has become clear that the correct thing to do is to implement new options to enhance the current Telnet specification so that it can support local line editing in a reasonable, reliable, and consistent manner.

There are three states that are of interest:

1) Local line editing and local signal trapping

2) Remote line editing, local signal trapping

3) Remote line editing, remote signal trapping

The case of local line editing and remote signal trapping is not a

very interesting case, because you don't recognize the signals, and cannot send them to the remote side for it to recognize until the line has been completed. Also, special signals usually will have an effect on the line editing function, and if they are not being trapped locally the desired action will not happen.

Local line editing means that all normal command line character processing, like "Erase Character" and "Erase Line", happen on the local system, and only when "CR LF" (or some other special character) is encountered is the edited data sent to the remote system.

Signal trapping means, for example, that if the user types the character associated with the IP function, then the "IAC IP" function is sent to the remote side instead of the character typed. Remote signal trapping means, for example, that if the user types the character associated with the IP function, then the "IAC IP" function is not sent to the remote side, but rather the actual character typed is sent to the remote side.

5. Implementation Rules

It is expected that any implementation that supports the Telnet LINEMODE option will support all of this specification.

5.1 User Interface

Normally, the entire user interface is left up to the implementor. However, there is functionality that the user should be able to specify on the client side of the connection. During a Telnet session, the client side should allow some mechanism for the user to give commands to the local Telnet process. These commands should at least allow the user to:

1) Change the mode of the connection. The user should be able to attempt to turn EDIT, FLOW, TRAPSIG, and ECHO on and off. The server may refuse to change the state of the EDIT and TRAPSIG bits.

2) Import or export SLC. The user should be able to tell the local Telnet process whether he wants to use the local or the current or default remote definitions of the special characters.

3) Manual sending of options. The user should be able to tell the local Telnet process to explicitly send any of the Telnet options (like IP, ABORT, AYT, etc.).

5.2 End of line terminators

When LINEMODE is turned on, and when in EDIT mode, when any normal line terminator on the client side operating system is typed, the line should be transmitted with "CR LF" as the line terminator. When EDIT mode is turned off, a carriage return should be sent as "CR NUL", a line feed should be sent as LF, and any other key that cannot be mapped into an ASCII character, but means the line is complete (like a DOIT or ENTER key), should be sent as "CR LF".

5.3 Output processing

Regardless of what mode has been negotiated, the server side is responsible for doing all output processing. Specifically, it should send "CR LF" when it wants the "newline" function, "CR NUL" when it wants just a carriage return, and "LF" when it wants just a linefeed.

5.4 A terminal driver in Telnet?

Conforming implementations need not do all the line editing themselves. There is nothing wrong with letting the system terminal driver handle the line editing, and have it hand to the Telnet application the completed and edited line, which is then sent to the remote system.

5.5 Setting of Local Characters

When this RFC was being developed, the original thought was that both sides of the connection would use their own defaults for the special characters, even if they were not the same on both sides of the connection. If this scheme is used, though, the view that the user has is that the local special characters are being used, and the remote character settings don't matter. It was decided that the client side of the connection should be in control of the character settings.

When LINEMODE is negotiated, the client must either export the local character settings to the server, or send a request (SLC 0 SLC_DEFAULT 0) to import the servers special characters. The usual action would be that a client running on a full fledged computer would export the special characters, and a client running where there are no local defaults (like on some terminal servers) would import the special characters.

When an SLC command is received, the action taken should be:

1) Ignore it if it is the same as the current settings.

2) If the SLC_LEVELBITS are the same as the current level bits, but the value is different and the SLC_ACK bit is set, no reply is generated. On the server side, the command is ignored, and on the client side, a switch is made to the new value. This is so that if a request to change the same character is generated by both the server and the client, they will both settle on the clients requested value.

3) If we agree with the new setting, we switch to it and reply with the same value, but also set the SLC_ACK bit.

4) If we don't agree, we send a response with what we think the value should be. The SLC_ACK bit is NOT set in this case. You may only disagree with a value by sending a different value at a lower level.

If the remote system doesn't support some of the line editing characters, but the front end does, then the front end may use the local definitions for those characters when in line mode. In this case, the server should send "SLC xxx SLC_DEFAULT 0" in response to a "SLC 0 SLC_DEFAULT 0" request, and just ack whatever value the client requests to set the function to.

The SLC_FORW2 character should only be used if SLC_FORW1 is already in use.

5.6 FORWARDMASK and SLC_FORW1 and SLC_FORW2

To help ease the amount of work needed to implement the client side, two methods of setting forwarding characters are provided. The SLC_FORW1 and SLC_FORW2 allow for the setting of two additional characters on which to forward buffered input data. Since many terminal drivers have the ability to set one or more line delimiters, it is fairly easy to support these without having to implement through the local terminal driver, rather than putting a terminal driver into Telnet. If the local terminal driver has functionality that maps easily into the FORWARDMASK, then it can also be easily supported. If the local terminal driver does not support that, then it would require more work to support FORWARDMASK.

Also note that the client side is required to forward data when it sees one of SLC_FORW1, SLC_FORW2, or FORWARDMASK characters, or when any normal line termination or special signal is encountered. The client side is also free to forward on other characters that it chooses. For example, if the server side sent a FORWARDMASK that asked for data to be forwarded on the first 20 control characters (ASCII codes 1 through 024), and the client side cannot have its local terminal driver forward on just the first 20 control

characters, but it can have the local terminal driver forward on any control character (ASCII codes 1 through 039), then the client side could validly accept the FORWARDMASK, and forward on any control character. When in EDIT mode, care should be taken to not forward at random times, since once that data is forwarded, no more editing on the forwarded part of the line can be done. The only time (other than the normal times) that data should be forwarded when in EDIT mode would be if a single input line is too long to handle locally.

5.7 Valid and invalid modes and values

At no time should "DO LINEMODE" be negotiated in both directions of the Telnet connection. The side that is the "DO LINEMODE" is considered to be the server side, and the side that is "WILL LINEMODE" is the client side.

At no time should "SB LINEMODE DO/DONT FORWARDMASK", be sent unless "DO LINEMODE" has been previously negotiated. At no time should "SB LINEMODE WILL/WONT FORWARDMASK", be sent unless "WILL LINEMODE" has been previously negotiated.

If an ABORT, EOF or SUSP, is received and the system does not support that functionality, it may just be ignored.

5.8 Flushing input and output

When an IP, BRK or ABORT is sent, it is usually desirable to be able to flush the input stream, and to flush output to the user until the IP, BRK, or ABORT is processed. The SLC_FLUSHIN and SLC_FLUSHOUT bits are used to indicate what action should be done. These bits are advisory only, but should be honored if possible. The standard method for processing the SLC_FLUSHIN is to use the Telnet "Synch" signal, and the SLC_FLUSHOUT is processed using the TIMING-MARK option. If both are to be sent, the IAC DM is sent before the DO TIMING-MARK. Thus, the sender would send "IAC XXX IAC DM IAC DO TIMING-MARK", where XXX may be IP, BRK or ABORT, or any other special character. The IAC DM is sent as TCP urgent data with the DM as the last (or only) data octet; this is used to flush the input stream. The "IAC DO TIMING-MARK" is used to tell when to stop flushing output; once it is sent, all data is discarded until an "IAC WILL TIMING-MARK" or an "IAC WONT TIMING-MARK" is received.

Since the SLC_FLUSHIN and SLC_FLUSHOUT bit are only advisory, the user interface should provide a method so that the user can override the sending (or not sending) of the "Synch" and TIMING-MARK, but the default action should be to send them according to the SLC_FLUSHIN and SLC_FLUSHOUT bits.

Whenever an IAC AO is received, a Synch must be returned. Whenever a Synch is being processed, (by the TCP connection going into Urgent mode), all data must be discarded (but not Telnet commands!) until an IAC DM is found, and the connection goes out of Urgent mode. See RFC

854, "TELNET PROTOCOL SPECIFICATION", for a complete description of the Synch signal.

5.9 State diagram for SLC

```
   +---------------------------------------------------------------+
   |                                IDLE                           |
   +----------------------+------+------+-------+-------+---------++
    ^      ^     ^        |      | ^    | ^     | ^     |       ^ |
    |      |     |        v      v |    | |     | |     v       | |
    |      |     |    +------+ +---+--+ | |     | | ########### | |
    |      |     |    | Get  | | Send | | |     | | #   Get   # | |
    |      |     |    | SPC0 | | SPC0 | | |     | | # 0,DEF,0 # | |
    |      |     |    +---+--+ +------+ | |     | | ########### | |
    |      |     |        |       ^     | |     | |     |       | |
    |      |     |        v       |     v |     | |     v       | |
    |      |     |       / \      | *********** | | ########### | |
    |      |     |     /     \    | *  Send   * | | # Switch  # | |
    | ********** |Yes/ Same as \  | * 0,VAL,0 * | | # to       # | |
    | * Change * +--<  current? > | *********** | | # default # | |
    | * to new *     \         /  |             v | ########### | |
    | * value  *       \     /    |     ***********     |       | |
    | **********         \ /      |     *  Send   *     v       | |
    |      ^              |No     |     * 0,DEF,0 *  #########  | |
    |      |Yes           v       |     ***********  # Send   #--+ |
    |     / \            / \      |                  # SPC-A #    |
    |   /     \        /     \    |                  #########    |
    | / Is ACK  \ Yes/  Same   \  |                     ^         |
    |< bit set?  ><-<  level as > |                     |         |
    | \         /    \ current?/  |                 ###########   |
    |   \     /        \     /    |                 #   Get   #<--+
    |     \ /            \ /    +-+---+             # 0,VAL,0 #
    |      |No            |No   | Set |             ###########
    |      +--------------+     | ACK |
    |                     v     | bit |      * - Client side only
    |                    / \    +-----+      # - Server side only
    |   +------+       /     \      ^
    |   | Send |  No /  Do we  \ Yes|
    +---| SPC1 |<---<   agree?  >---+
        +------+     \         /
                       \     /
                         \ /

           SPC0    Initial setting for a special character
           SPC1    A changed special character < SPC0
           SPC-A   All current special character settings
           VAL     SLC_VALUE level
           DEF     SLC_DEFAULT level
```

```
   Levels: DEFAULT, VALUE, CANT_CHANGE, NOSUPPORT
   Flags: ACK

           Receive                  Response
           --------                 ---------
           f,SLC_DEFAULT,x          f,SLC_VALUE,v
                                    f,SLC_CANTCHANGE,v
                                    f,SLC_NOSUPPORT,x

           f,SLC_VALUE,v            f,SLC_ACK|SLC_VALUE,v
                                    f,SLC_CANTCHANGE,w
                                    f,SLC_NOSUPPORT,x

           f,SLC_CANTCHANGE,v       f,SLC_ACK|SLC_CANTCHANGE,v
                                    f,SLC_NOSUPPORT,x

           f,SLC_NOSUPPORT,x        f,SLC_ACK|SLC_NOSUPPORT,x

           x,SLC_ACK|x,x            no response
```

5.10 Examples of a connection

In these examples, the symbolic names are used rather than the actual values, to make them readable. When two or more symbolic names are joined by a |, it means that the actual value will be the logical "or" of the values of the symbolic names. In the interest of clarity, for these examples the leading IAC and IAC SB sequences, and the trailing IAC SE sequences have been omitted. Also, the SLC_ prefix has been left off where ever it would normally occur.

```
       CLIENT                        SERVER
       ------                        ------
                                     DO TOGGLE-FLOW-CONTROL
                                     DO LINEMODE
       WILL TOGGLE-FLOW-CONTROL
       WILL LINEMODE
       [ Subnegotiation may now proceed in both directions.  The client
         sends of the list of special characters.  ]
       LINEMODE SLC SYNCH DEFAULT 0
       IP VALUE|FLUSHIN|FLUSHOUT 3 AO
```

VALUE 15 AYT DEFAULT 0 ABORT
VALUE|FLUSHIN|FLUSHOUT 28 EOF
VALUE 4 SUSP VALUE|FLUSHIN 26
EC VALUE 127 EL VALUE 21 EW
VALUE 23 RP VALUE 18 LNEXT
VALUE 22 XON VALUE 17 XOFF
VALUE 19
[Now that linemode is enabled, the server sets the initial

mode, and acknowledges the special characters.]
LINEMODE MODE EDIT

LINEMODE SLC SYNCH NOSUPPORT 0 IP
VALUE|FLUSHIN|FLUSHOUT|ACK 3 AO
NOSUPPORT 0 AYT NOSUPPORT 0 ABORT
VALUE|FLUSHIN|FLUSHOUT|ACK 28 EOF
VALUE|ACK 4 SUSP NOSUPPORT 0 EC
VALUE|ACK 127 EL VALUE|ACK 21 EW
VALUE|ACK 23 RP VALUE|ACK 18 LNEXT
VALUE|ACK 22 XON VALUE|ACK 17 XOFF
VALUE|ACK 19
[The client gets the mode and ack of the special characters, and acks the mode and any special characters that the server changed.]
LINEMODE MODE EDIT|MODE_ACK

LINEMODE SLC SYNCH
NOSUPPORT|ACK 0 AO
NOSUPPORT|ACK 0 AYT|ACK NOSUP-
PORT 0 SUSP NOSUPPORT|ACK 0
"Login:"
"my_account"
[Turn off echo to the user.]
WILL ECHO
DO ECHO
"Password:"
"my_password"
[Turn back on echo to the user.]
WONT ECHO
DONT ECHO
[User does some stuff, and then runs an application that wants to use single character mode, doing its own echoing of

```
  characters, but keep signal trapping on.  ]
                                             WILL ECHO
DO ECHO
                                             LINEMODE MODE TRAPSIG
LINEMODE MODE TRAPSIG|MODE_ACK
[ Application finishes.  ]
                                             WONT ECHO
DONT ECHO
                                             LINEMODE MODE EDIT|TRAPSIG
LINEMODE MODE
EDIT|TRAPSIG|MODE_ACK
[ Another application, that wants full control of everything.  ]
                                             WILL ECHO
DO ECHO
                                             LINEMODE MODE 0
```



```
LINEMODE MODE 0|MODE_ACK
[ Application finishes.  ]
                                             WONT ECHO
DONT ECHO
                                             LINEMODE MODE EDIT|TRAPSIG
LINEMODE MODE
EDIT|TRAPSIG|MODE_ACK
[ The user changes his erase character to ^H.  ]
                                             LINEMODE SLC EC VALUE 8
LINEMODE SLC EC VALUE|ACK 8
[ The user decides to revert to all the original client side
  special characters.  ]
LINEMODE SLC SYNCH DEFAULT 0
IP VALUE|FLUSHIN|FLUSHOUT 3 AO
VALUE 15 AYT DEFAULT 0 ABORT
VALUE|FLUSHIN|FLUSHOUT 28 EOF
VALUE 4 SUSP VALUE|FLUSHIN 26
EC VALUE 127 EL VALUE 21 EW
VALUE 23 RP VALUE 18 LNEXT
VALUE 22 XON VALUE 17 XOFF
VALUE 19
                                             LINEMODE SLC SYNCH NOSUPPORT 0 AO
                                             NOSUPPORT 15 AYT NOSUPPORT 0 SUSP
                                             NOSUPPORT|FLUSHIN 26 EC VALUE|ACK
                                             127 EW VALUE|ACK 23 RP VALUE|ACK
                                             18 LNEXT VALUE|ACK 22 XON
```

```
                                       VALUE|ACK 17 XOFF VALUE|ACK 19
LINEMODE SLC SYNCH
NOSUPPORT|ACK 0 AO
NOSUPPORT|ACK 15 AYT
NOSUPPORT|ACK 0 SUSP
NOSUPPORT|ACK|FLUSHIN 26
[ The user decides to import the remote sides default special
  characters.  ]
LINEMODE SLC 0 DEFAULT 0
                                       LINEMODE SLC IP
                                       VALUE|FLUSHIN|FLUSHOUT 3 ABORT
                                       VALUE|FLUSHIN|FLUSHOUT 28 EOF
                                       VALUE 4 EC VALUE 127 EL VALUE 21
[ Since these are the same as the current local settings, no
  response is generated.  ]
[ This next example is what would happen if an editor was fired
  up, that wanted to let the client side do the echoing and
  buffering of characters, but did not want it to do any line
  editing, and only forward the data when got a control
  character.  Note that we have preceded all the the 0377s in the
  forward mask with an IAC.  ]
                                       LINEMODE MODE 0
```



```
                                       LINEMODE DO FORWARDMASK IAC 0377
                                       IAC 0377 IAC 0377 IAC 0377 0 0 0 0
                                       0 0 0 0 0 0 0 01
LINEMODE MODE 0
LINEMODE WILL FORWARDMASK
[ Application runs to completion, and then things are to be set
  back to what they were before.  ]
                                       LINEMODE MODE EDIT|TRAPSIG
                                       LINEMODE DONT FORWARDMASK
LINEMODE MODE EDIT|TRAPSIG
LINEMODE WONT FORWARDMASK
```

6. Other Telnet options and RFCs

The following is a list of RFCs for various Telnet options that should be supported along with LINEMODE.

1. Postel, J. and J. Reynolds, "TELNET PROTOCOL SPECIFICATION", RFC 854, USC/Information Sciences Institute, May 1983.

2. Postel, J. and J. Reynolds, "TELNET OPTION SPECIFICATIONS", RFC 855, USC/Information Sciences Institute, May 1983.

3. Postel, J. and J. Reynolds, "TELNET BINARY TRANSMISSION", RFC 856, USC/Information Sciences Institute, May 1983.

4. Postel, J. and J. Reynolds, "TELNET ECHO OPTION", RFC 857, USC/Information Sciences Institute, May 1983.

5. Postel, J. and J. Reynolds, "TELNET SUPRESS GO AHEAD OPTION", RFC 858, USC/Information Sciences Institute, May 1983.

6. Postel, J. and J. Reynolds, "TELNET TIMING MARK OPTION", RFC 860, USC/Information Sciences Institute, May 1983.

7. VanBokkeln, J., "Telnet Terminal-Type Option", RFC 1091, FTP Software, Inc., February 1989.

8. Waitzman, D., "Telnet Window Size Option", RFC 1073, BBN STC, October 1988.

9. Hedrick, C., "Telnet Remote Flow Control Option", RFC 1080, Rutgers University, November, 1988.

10. Hedrick, C., "Telnet Terminal Speed Option", RFC 1079, Rutgers University, December, 1988.

The following is a list of RFCs that need not be supported for

LINEMODE, but which would enhance any TELNET implementation.

11. Postel, J. and J. Reynolds, "TELNET STATUS OPTION", RFC 859, USC/Information Sciences Institute, May 1983.

12. Postel, J. and J. Reynolds, "TELNET END OF RECORD OPTION", RFC 885, USC/Information Sciences Institute, December 1983.

13. Silverman, S., "OUTPUT MARKING TELNET OPTION", RFC 933, MITRE-Washington, January 1985.

14. Marcy, G., "Telnet X Display Location Option", RFC 1096, Carnegie

Mellon University, March 1989.

Author's Address

Dave Borman
Cray Research Inc.
1440 Northland Drive
Mendota Heights, MN 55120

Phone: (612) 681-3398

EMail: dab@CRAY.COM