NVT (Network Virtual Terminal) description

English version

Czech version

Communication with the TCP/IP device over the Ethernet network can be extended to more functions using NVT (Network Virtual Terminal) commands. It can be used according to the RFC2217 standard to change baudrate on remote Virtual Serial Port for example. The simple NVT control commands are included in the data stream with the character "FF" used as the command prefix. If the "FF" character occurs within the normal data stream, it is simply doubled.
You can find detailed NVT and a complete TELNET description in the last section of this article, we shall begin with a detailed manual of used commands and examples.

Article content :

Description of the NVT commands implemented in HW devices

The aim in our applications is to stay compatible with existing standards, but there is a need to implement several parts of different standards, because some things are not necessary for our applications and some implementation parts are very useful and necessary. Basically we use some extensions from RCF2217 standard (controlling the serial asynchronous channel properties) and some more extensions (controlling I/O pins and other peripherals).

You can download the whole RCF 2217 standard documentation here- rfc2217.txt

The size of implemented extensions is getting larger every day, so the following functions are just a base. The full list of implemented commands is available upon request. There may be some commands specific to certain devices.

The basic functions supported by NVT

The basic functions of the TELNET protocol are described in the RFC854 (rfc0854.txt) standard or in a shorter version within "The Telnet Protocol" chapter of this manual.

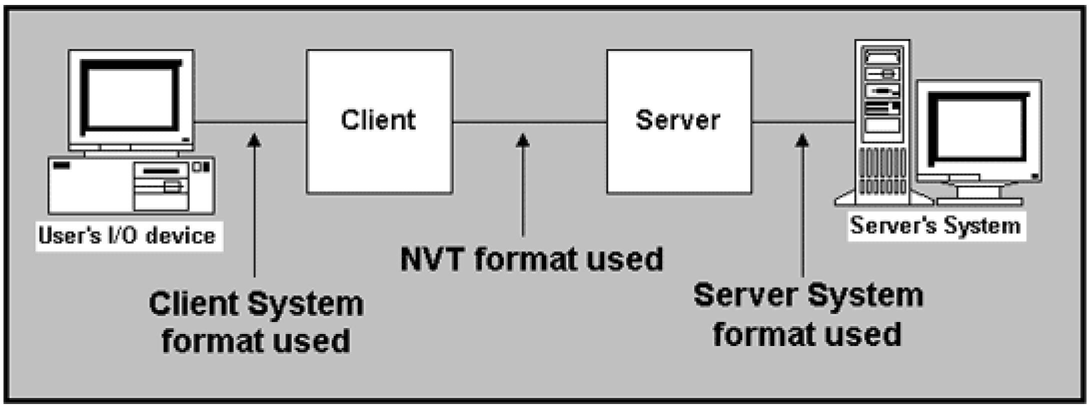
Brief NVT description :

- NVT comands are inserted to the data stream via TCP/IP before sending to the TCP/IP connection.
- Every NVT command is prefixed by character "0xFF".
- There are some basic commands with 2 byte interpretation only (EOF, ABORT, BRK, AYT, NOP, EC), and others with defined start (<IAC><SB> = 0xFF 0xFA) and defined end (<IAC><SE> = 0xFF 0xF0) commands.
- The TCP/IP device separates NVT commands and processes them without delay, while the data stream stores to the output stack. Hence NVT commands are asynchronous and independent on the data stream! You can't simply send "A", change parity and then send "B" to the TCP/IP connection. This doesn't result in character "A" having a different parity than character "B".
- The NVT commands can't be found in the serial port data, if the device is Serial / TCP/IP converter. The firmware of the TCP/IP device filter all NVT from the data stream. Because of this, you have to switch on/off the NVT support in the SETUP of the Converter device.
  If you are sending character "0xFF" (255), the PC will just double it, because in NVT "0xFFFF" means send character "0xFF" to the output.
- NVT uses a **negotiation** process. It's a way of testing if terminals on the opposite side use ECHO or not ar if there are specific terminals etc. We support easy negotiation with RFC2217 confirmation.

It is good to know, there are basic commands (EOF, ABORT, BRK, AYT, NOP, EC...), which might be represented by double characters that we do not need to use in practical applications using the embedded device (Unless NOP and AYT).

Dec	HEX	Shortcut	Description
240	F0	SE	End of sub negotiation parameters
241	F1	NOP	No Operation
246	F6	AYT	Are You There
250	FA	SB	Indicates that what follows is sub negotiation of the indicated option.
255	FF	IAC	Data Byte 255

- **AYT** is useful to request identification of the device and check ahead for NVT compatible devices.
- **NOP** means NO OPERATION, but we can use it to "keep connection" when there is no data stream. You can send the command and over the TCP/IP layer confirmation, check there is still functionality and an opened TCP connection.



Supported RFC2217 functions

The RFC2217 is the standard defined in October 1997 by G. Clark from Cisco Systems, Inc. It proposes a protocol to allow greater use of modems attached to a network for outbound dialing purposes. In brief it describes how to control the remote asynchronous serial port connected over the TCP/IP network. It enables you to change remote serial port baudrate speeds, parity and other paramethers. You can download the whole RCF 2217 standard documentation here - rfc2217.txt

We support:

Com Port Control Client to Access Server constants
CAS_SIGNATURE, 0
CAS_SET_BAUDRATE, 1
CAS_SET_DATASIZE, 2
CAS_SET_PARITY, 3
CAS_SET_STOPSIZE, 4
CAS_SET_CONTROL, 5
CAS_NOTIFY_LINESTATE, 6
CAS_NOTIFY_MODEMSTATE, 7
CAS_FLOWCONTROL_SUSPEND,8
CAS_FLOWCONTROL_RESUME, 9
CAS_SET_LINESTATE_MASK, 10
CAS_SET_MODEMSTATE_MASK,11
CAS_PURGE_DATA, 12
CAS_OPT_GPIO, 50
CAS_SET_GPIO, 51

Com Port Control Access Server to Client constants
ASC_SIGNATURE, 100
ASC_SET_BAUDRATE, 101
ASC_SET_DATASIZE, 102
ASC_SET_PARITY, 103
ASC_SET_STOPSIZE, 104
ASC_SET_CONTROL, 105

ASC_NOTIFY_LINESTATE, 106
ASC_NOTIFY_MODEMSTATE, 107
ASC_FLOWCONTROL_SUSPEND,108
ASC_FLOWCONTROL_RESUME, 109
ASC_SET_LINESTATE_MASK, 110
ASC_SET_MODEMSTATE_MASK,111
ASC_PURGE_DATA, 112
ASC_OPT_GPIO, 150
ASC_SET_GPIO, 151

Supported General Purpose Input Output (GPIO) functions

We extended the RFC2217 standard to incorporate some GPIO (General Purpose Input Output) functions listed below. It's not standardised, but we didn't find any standards for GPIO functions during the year 2001.

COM-PORT-OPTION - 44 (2C hex)

Behind the sequence IAC SB, there might also be the enlargement COM-PORT-OPTION command (the command is ended by the IAC SE sequence of course) in RFC2217 standard. We are only describing some of the sub-commands. The whole description is in the RFC2217 standard.

The values **up to 100 dec** are valid in the **Client** >> **Server** mode.
The values **higher than 100 dec** are valid in **Server** >> **Client** mode

Dec	HEX	Description
0	00	CAS_SIGNATURE
1	01	CAS_SET_BAUDRATE
2	02	CAS_SET_DATASIZE
3	03	CAS_SET_PARITY
4	04	CAS_SET_STOPSIZE
5	05	CAS_SET_CONTROL
6	06	CAS_NOTIFY_LINESTATE
7	07	CAS_NOTIFY_MODEMSTATE
8	08	CAS_FLOWCONTROL_SUSPEND
9	09	CAS_FLOWCONTROL_RESUME
10	0A	CAS_SET_LINESTATE_MASK
11	0B	CAS_SET_MODEMSTATE_MASK
12	0C	CAS_PURGE_DATA
50	32	CAS_OPT_GPIO
51	33	CAS_SET_GPIO
:	:	
+100	**+64**	**ASC_**

150	96	ASC_OPT_GPIO
151	97	ASC_SET_GPIO
Supported NVT commands		

ASC_ is the device response to the CAS_ command. It means the PC will send a CAS_SET_PARITY command and the TCP/IP device will reply with the parity value confirmation ASC_SET_PARITY.

COM-PORT-GPIO SUBOPTION - 50 and 51 (32 and 33 hex)

For direct I/O pin control we use the double byte command GPIO-50 or 51 (which is behind the COM-PORT-OPTION 44 command) followed by the sub option sequence.

Sub option 50 (32 hex)
- **0 (00 hex)** – request for input state reading - the answer contains the value of the input port (The CPU’s pins or input shift register)
- **16 .. 23 (10 .. 17 hex)** – set the output bit 0..7 as 1
- **32 .. 39 (20 .. 27 hex)** – set the output bit 0..7 as 0
- **48 (30 hex)** – request for output state reading - tha answer contains the value of the output port

Sub option 51 (33 hex)
- sets the sent XX value to the output port, see example. It sends the same answer back, because it reads it from the inner pseudoregister.

We practically control the GPIO port by sending the "**FF FA 2C 32 XX FF F0**“ sequence with XX port value. (the XX value is sent to the output port). For example the 0x11 (0x11 = 11 hex) value sets P1.1 to 1, other P1 pins stay unchanged.

Sub option 52 (34 hex)
Sends the **outputs value, <u>if there has been a change in any state</u>**, or if the device was powered on. This command does not expect any answer, therefore there is no value for 152 (98 HEX) in the table (a single report is basically an unrequested answer).

This command **can be preceded** with a "**FF FA 2C 32 00 FF F0**", sequence. This command is used to synchronize binary inputs and outputs of two devices connected to each other.

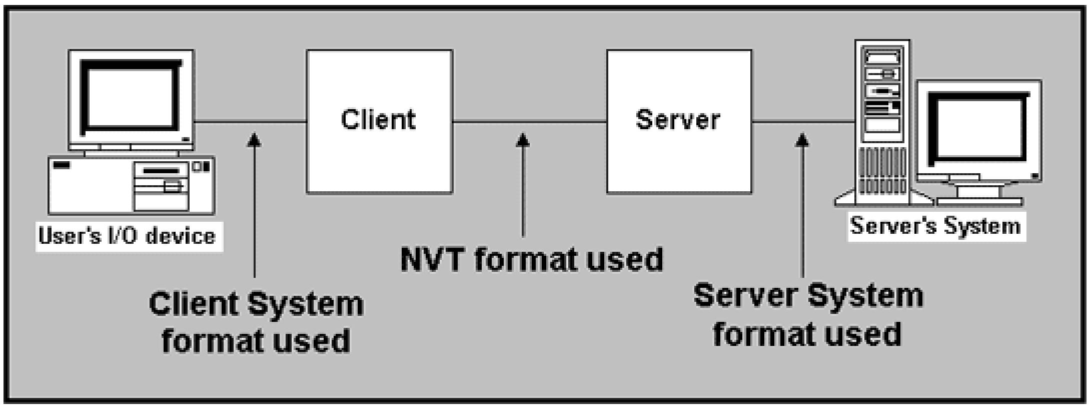
If we receive the "**FF FA 2C 34 XX FF F0**" in the data flow, we know that an input port has been changed.

Why two sequencies ?
In the appropriate communication between two I/O controller devices, the first device sends the "**FF FA 2C 32 00 FF F0**" sequence only once, because the other I/O Controller responds

with **"FF FA 2C 97 XX FF F0"** (where XX is its inputs state). From now on, both sides send only the "**FF FA 2C 34 XX FF F0**" if there has been an input state change.

Note: *To activate the function that informs about the input changes, you need to set the monitored input range with the "#T: Trigger AND mask"command.*

- To transmit all the inputs set: #T=255
- To turn off inputs transmitting set #T=0



How to use NVT correctly

Almost all the NVT commands have a set number of characters. If the value is set in 4 Byte format and we would like to read the actual value from the device by the "0 setting", we have to send this "0" as a 00 00 00 00 hex sequence.

Setting up the output of the GPIO port

The following command sets the output GPIO port to hex value AA (10101010 bin)

```
<IAC><SB><COM_PORT_OPTION><CAS_SET_GPIO><byte to
output)><IAC><SE>
 FF  FA      2C             33           AA          FF
  F0
```

```
This sequence is the answer from the device :
<IAC><SB><COM_PORT_OPTION><ACS_SET_GPIO><byte to
output)><IAC><SE>
 FF  FA      2C             97           AA          FF
  F0
```

You can see that the answer from the device contains +100 DEC (+64 HEX because 0x33 + 0x64 = 0x97) value for the NVT command confirmation.

Reading the GPIO inputs

By sending the **FF FA 2C 32 00 FF F0** sequence >> the embedded server responds with: **FF FA 2C 96 XX FF F0**
- where XX is the actual value of the input pins.

Reading the output value

By sending the **FF FA 2C 32 30 FF F0** sequence >> the embedded server responds with : **FF FA 2C 97 XX FF F0**
- where XX is the actual value of the register, which is used to set the outputs.

Note: *The output value here is the output REGISTER value. This can be very useful if you have to combine bit and byte oriented commands.*

Clearing the P1.5 output pin (or D5 of the data output)

By sending the **FF FA 2C 32 25 FF F0** sequence >> the embedded server responds with : **FF FA 2C 97 DF FF F0**
- where DF is the actual value of the output port (it also depends on the previous port state). The embedded server only changes one bit, but returns the whole port value.

Reporting the input changes

If you do not send ani request from your PC and the input state changes, you receive the "**FF FA 2C 32 00 FF F0** " and the "**FF FA 2C 34 XX FF F0**" sequence

Where the **XX** is the actual value of the binary input.

The function works similar to the incoming data on the serial port. If your device works in the TCP Client/Server mode and has NVT enabled, then when you receive 1 byte from the serial port (or there is any binary input change), teh device tries to establish a TCP connection and send the data. When the TCP connection is established, only the actual inputs status is sent, not the whole history of changes.

Note: *To activate the function that informs about the input changes, you need to set the monitored input range with the "#T: Trigger AND mask"command.*

- To transmit all the inputs set: #T=255
- To turn off inputs transmitting set #T=0

How to change the RS-232 line baudrate speed

You can check the actual Port speed by sending the **00 00 00 00** sequence. If you send any other value, the Port speed will be changed by the server. The dec value corresponds with the port speed in Bd.

By sending the **FF FA 2C 01 00 00 00 00 FF F0** sequence >> The server returns : **FF FA 2C 65 00 00 25 80 FF F0**
After the 00 00 25 80 HEX->DEC conversion, we can read the port speed directly, which is 9600Bd.

Keep Connection

If data is not transferred, the server terminates the connection. To keep the connection active all the time, use the "`K: Keep connection`" command in the setup mode of the Embedded device.
The connection is active for an unlimited time, because the NOP command sends (FF F1 sequence) from the Embedded device side every 5seconds. The TCP connection will close if the TCP layer finds it can't deliver packets for too long a time.

Note : *Keep connection function works only if NVT support is switched on!*

How to solve the 9th bit problem?

The "*space mark*" NVT feature has been implemented from version 2.3. This feature can be used for the 9th bit settings, which was commonly used by older applications during 90's.

Note, the parity change is asynchronous, it is not buffered, but it is completed after the character is received. The synchronous functions are available as our proprietary solution by sending the 0xFE'P' sequence, which reserves the parity. You have to enable the "`Variable Parity`" support in the SETUP.

Are You There ?

If you would like to check whether the device is available via the network, there is a special "**Are you there**" command in telnet application. Usually, the response from a standard Unix device is "Yes". We have extended this response as follows: If you send the **FF F6** sequence, our Embedded server will answer the response in the following format:

<WEB51 HW 4.5 SW 2.3 SN 01A03B #01>

Which means :
<WEB51 HW XXX SW XXX SN 1035EE #0F *OvErr *ParErr *FlErr>

There is the device name, HW version, firmware version and “S/N”, which is the last 3 Bytes of the MAC address. The sequence behind the * character is just for the status response and is not necessary.

The Telnet Protocol

The Telnet protocol is often thought of as simply providing a facility for remote logins to a computer via the Internet. This was its original purpose although it can be used for many other functions.
It is best understood in the context of a user with a simple terminal using the local telnet program (known as the client program) to run a login session on a remote computer where his communication needs are handled by a telnet server program. It should be emphasised that the telnet server can pass on the data it has received from the client to many other types of process including a remote login server. It is described in the RFC854 standard, first published in 1983.

The Network Virtual Terminal

Communication is established using the TCP/IP protocols and communication is based on a set of facilities known as a Network Virtual Terminal (NVT). At the user or client end the telnet client program is responsible for mapping incoming NVT codes to the actual codes needed to operate the user's display device and is also responsible for mapping user generated keyboard sequences into NVT sequences.

The NVT uses 7 bit codes for characters, the display device, referred to as a printer in the RFC, is only required to display the "standard" printing ASCII characters represented by 7 bit codes and to recognise and process certain control codes. The 7 bit characters are transmitted as 8 bit bytes with most significant bit set to zero. An end-of-line is transmitted as the character sequence CR (carriage return) followed by LF (line feed). If it is desired to transmit an actual carriage return this is transmitted as a carriage return followed by a NUL (all bits zero) character.

NVT ASCII is used by many other Internet protocols.
The following control codes are required to be understood by the Network Virtual Terminal.

Name	code	Decimal Value	Function
NULL	NUL	0	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.

The following control codes are optional but should have the indicated effect on the display.

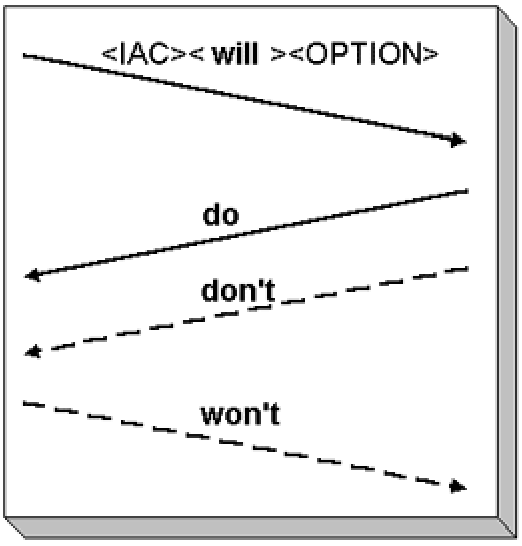
Name	code	Decimal Value	Function
BELL	BEL	7	Produces an audible or visible signal (which does NOT move the print head.
Back Space	BS	8	Moves the print head one character position towards the left margin. [On a printing devices this mechanism was commonly used to form composite characters by printing two basic characters on top of each other.]
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. [On visual displays this commonly clears the screen and moves the cursor to the top left corner.]

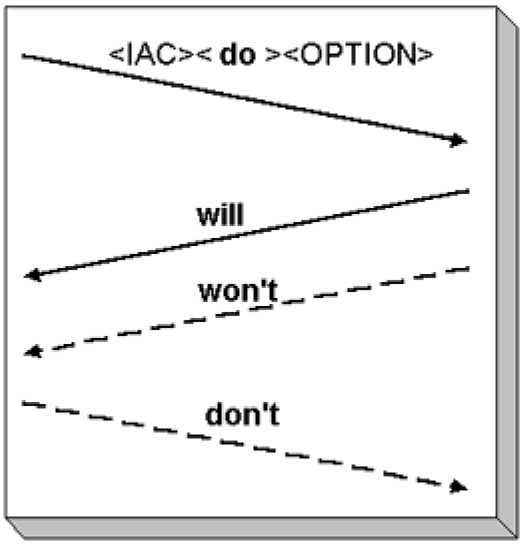
The NVT keyboard is specified as being capable of generating all 128 ASCII codes by using keys, key combinations or key sequences.

Commands

The telnet protocol also specifies various commands that control the method and various details of the interaction between the client and server. These commands are incorporated within the data stream. The commands are distinguished by the use of various characters with the most significant bit set. Commands are always introduced by a character with the decimal code 255 known as an Interpret as command (IAC) character. The complete set of special characters is

Name	Decimal Code	Meaning
SE	240	End of subnegotiation parameters.
NOP	241	No operation
DM	242	Data mark. Indicates the position of a Synch event within the data stream. This should always be accompanied by a TCP urgent notification.
BRK	243	Break. Indicates that the "break" or "attention" key was hit.
IP	244	Suspend, interrupt or abort the process to which the NVT is connected.
AO	245	Abort output. Allows the current process to run to completion but do not send its output to the user.
AYT	246	Are you there. Send back to the NVT some visible evidence that the AYT was received.
EC	247	Erase character. The receiver should delete the last preceding undeleted character from the data stream.
EL	248	Erase line. Delete characters from the data stream back to but not including the previous CRLF.
GA	249	Go ahead. Used, under certain circumstances, to tell the other end that it can transmit.
SB	250	Subnegotiation of the indicated option follows.
WILL	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option.
WONT	252	Indicates the refusal to perform, or continue performing, the indicated option.
DO	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the indicated option.
DONT	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	Interpret as command





There are a variety of options that can be negotiated between a telnet client and server using commands at any stage during the connection. They are described in detail in separate RFCs. The following are the most important.

Decimal code	Name	RFC
1	echo	857
3	suppress go ahead	858
5	status	859
6	timing mark	860
24	terminal type	1091
31	window size	1073
32	terminal speed	1079
33	remote flow control	1372
34	linemode	1184
36	environment variables	1408

Options are agreed by a process of negotiation which results in the client and server having a common view of various extra capabilities that affect the interchange and the operation of applications.
Either end of a telnet dialogue can enable or disable an option either locally or remotely. The initiator sends a 3 byte command of the form

IAC,<type of operation>,<option>

The response is of the same form.

Operation is one of

Description	Decimal Code	Action

WILL	251	Sender wants to do something.
DO	252	Sender wants the other end to do something.
WONT	253	Sender doesn't want to do something.
DONT	254	Sender doesn't want the other end to do anything.

Associated with each of the these there are various possible responses

Sender Sent	**Receiver Responds**	**Implication**
WILL	DO	The sender would like to use a certain facility if the receiver can handle it. Option is now in effect
WILL	DONT	Receiver says it cannot support the option. Option is not in effect.
DO	WILL	The sender says it can handle traffic from the sender if the sender wishes to use a certain option. Option is now in effect.
DO	WONT	Receiver says it cannot support the option. Option is not in effect.
WONT	DONT	Option disabled. DONT is only valid response.
DONT	WONT	Option disabled. WONT is only valid response.

For example if the sender wants the other end to suppress go-ahead it would send the byte sequence
255(IAC),251(WILL),3
The final byte of the three byte sequence identifies the required action.
For some of the negotiable options values need to be communicated once support of the option has been agreed. This is done using sub-option negotiation. Values are communicated via an exchange of value query commands and responses in the following form.
IAC,SB,<option code number>,1,IAC,SE
and
IAC,SB,<option code>,0,<value>,IAC,SE

For example if the client wishes to identify the terminal type to the server the following exchange might take place
Client 255(IAC),251(WILL),24
Server 255(IAC),253(DO),24

Server 255(IAC),250(SB),24,1,255(IAC),240(SE)
Client 255(IAC),250(SB),24,0,'V','T','2','2','0',255(IAC),240(SE)

The first exchange establishes that the terminal type (option number 24) will be handled, the server then enquires of the client what value it wishes to associate with the terminal type. The sequence SB,24,1 implies **sub-option negotiation** for option type 24, value required (1). The IAC,SE sequence indicates the end of this request. The response IAC,SB,24,0,'V'... implies sub-option negotiation for option type 24, value supplied (0), the IAC,SE sequence indicates the end of the response (and the supplied value).
The encoding of the value is specific to the option but a sequence of characters, as shown above, is common.

Telnet Negotiable Options

Many of those listed here are self-evident, but some call for more comments.

- **Suppress Go Ahead**
  The original telnet implementation defaulted to "half duplex" operation. This means that data traffic could only go in one direction at a time and specific action is required to indicate the end of traffic in one direction and that traffic may now start in the other direction. [This similar to the use of "roger" and "over" by amateur and CB radio operators.] The specific action is the inclusion of a GA character in the data stream. Modern links normally allow bi-directional operation and the "suppress go ahead" option is enabled.

- **echo**
  The echo option is enabled, usually by the server, to indicate that the server will echo every character it receives. A combination of "suppress go ahead" and "echo" is called 'character at a time mode' meaning that each character is separately transmitted and echoed.
  There is an understanding known as kludge line mode which means that if either "suppress go ahead" or "echo" is enabled but not both then telnet operates in line at a time mode meaning that complete lines are assembled at each end and transmitted in one "go".

- **linemode**
  This option replaces and supersedes the line mode kludge.

- **remote flow control**
  This option controls where the special flow control effects of Ctrl-S/Ctrl-Q are implemented.

Telnet control functions

The telnet protocol includes a number of control functions. These are initiated in response to conditions detected by the client (usually certain special keys or key combinations) or server. The detected condition causes a special character to be incorporated in the data stream.

- **Interrupt Process**
  This is used by the client to cause the suspension or termination of the server process. Typically the user types Ctrl-C on the keyboard. An IP (244) character is included in the data stream.
- **Abort Output**
  This is used to suppress the transmission of remote process output. An AO (238) character is included in the data stream.
- **Are You There**
  This is used to trigger a visible response from the other end to confirm the operation of the link and the remote process. An AYT (246) character is incorporated in the data stream.
- **Erase character**
  Sent to the display to tell it to delete the immediately preceding character from the display. An EC (247) character is incorporated in the data stream.
- **Erase line**
  Causes the deletion of the current line of input. An EL (248) character is incorporated in the data stream.
- **Data Mark**
  Some control functions such as AO and IP require immediate action and this may cause difficulties if data is held in buffers awaiting input requests from a (possibly misbehaving) remote process. To overcome this problem a DM (242) character is sent in a TCP Urgent segment, this tells the receiver to examine the data stream for "interesting" characters such as IP, AO and AYT. This is known as the telnet synch mechanism.

Related Datasheets & Links

- Originally whole RCF 2217 standard documentation - rfc2217.txt