PART 6: COMMUNICATIONS PROTOCOL

Introduction – Using The EVOLUTION Printer Control Language

To ensure that all features of EVOLUTION printers are used to their fullest, this section has been written as an aid in creating applications.
This section has been written with both the professional and advanced programmer in mind. It is assumed the reader understands concepts such as: ASCII codes, typical printer control languages, command structures, objects and various parametric programming. It is also assumed the reader can use a programming language like C, C/C++, Basic, or any other programming language capable of sending and receiving commands to and from the EVOLUTION printer via a serial communications port.

This communication protocol covers all EVOLUTION products. Some commands are not applicable to certain units, and care must be taken in determining what valid commands are for each specific printer. Commands that reference specific units are so noted.

Communications between a printer and the hand held controller, a host computer, or PLC are identical. The hand held controller limits the available features of the printer to simplify operation and minimize user data entry mistakes.

The communications protocol is via an RS485 data link operating in a master/slave environment where the printers are the slaves. There can only be one master such that both the hand held controller and a host device cannot coexist.

RS485 communications can be used effectively over long distances (up to 1000 feet) and in electrically noise environments as result of electromagnetic interference from motors and welding equipment. Also, multiple receivers (EVOLUTION Printers) -up to 32- may be connected to such a network in a linear multi-drop configuration in a master-slave topology.

If we are interested in writing a customized Windows software application to drive the EVOLUTION LX printer from a personal computer the standard RS232 serial port has to be used to control and drive the RS485 EVOLUTION LX Printer. So, the addition of a converter from RS232 to RS485 is necessary with its corresponding driver.

Since today the RS232 is being gradually replaced by USB for communications in personal computers as result of which most of the new computers desktops and laptops no longer have built-in standard serial (COM) ports, a USB to RS232 converter also is necessary, therefore a **“virtual serial port”** which is an emulation of the standard serial port has to be installed. This port is created by software which enables extra serial ports in an operating system without additional hardware installation.

The converter its driver and the virtual serial port along with the cables are provided by Digital Design. Note that it assigns COM3 to the virtual serial port.

Also, in writing Windows Applications is necessary to considerer the following:

Since, we are using the RS485 the biggest difference **related to programming** the RS232 is that we can have up to 32 EVOLUTION Printers and since communication is initiated by a master computer some form of arbitration is necessary therefore, the EVOLUTION COMMUNICATIONS PROTOCOL which is described next must be used accordingly. Here we must make sure **the first data byte to be sent** to the EVOLUTION Printer is the address of the recipient printer, further complicating this, is the fact that for compatibility with our legacy systems we are using 7 bits for transmitting data(which allow us to transmit 127 ASCII characters), so the byte that identify the printer address has to be split up into two nibbles(4 bits long) which means that the original printer's address byte has to be masked and shifted in order to have the Upper/Lower Address nibbles which are sent separately.

The same has to be done when a parameter (like printer speed) is being transmitted.

Finally, if we are planning to use MS Visual Basic as development tool for building applications we should use the MSComm serial communications control which allows direct control access to the standard or virtual serial port in a PC, which of course has to be initialized before being used, in this case also a timer has to be used to continually pool the serial port in order to receive data from the EVOLUTION Printer. (a full example of it you can find in the folder:\Printer Communications\EVCOMMTESTRELEASE\Setup.exe).

If, we are planning to use MS Visual C++ or MS Visual C#, the Serial Port class included in Microsoft's .NET Framework is the best tool to be used (a full example of it you can find in the folder: \Printer Communications\EVComunications\WindowsApplication14.exe).

There is also, another application to test comunications between a PC and the EV Printer, which is located in the folder:

\Printer Communications\Physical Connection Test\EV_TEST.exe).

ASCII CHARACTER CHART

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

DESCRIPTION

This communication protocol is based on Version 1.4, which was initially released NOV 2005 and is used with all EVOLUTION products. The communications option converses with a host computer via an RS485 data link.

NOTE: EACH REQUEST OR COMMAND SENT TO A PRINT STATION RECEIVES A RESPONSE FROM THAT PRINT STATION. COMMUNICATIONS SOFTWARE MUST WAIT FOR A RESPONSE TO DETERMINE IF THE PRINT STATION WAS READY TO ACCEPT THE COMMAND, AND THE DATA WAS VALID AND PROCESSED. NO RESPONSE COULD INDICATE THE DATA WAS LOST. IF AN ERROR WAS DETECTED IN PROCESSING A NAK WITH AN ERROR CODE IS RETURNED.

DATA WORD DEFINITION

Full Duplex
7 Data Bits
1 Even Parity Bit
1 Start Bit
1 Stop Bit

BAUD RATE

115,200 Bits per second

DEFINITIONS

Q=QUERY TO HEAD
R=RESPONSE FROM HEAD
D=DATA UPDATE TO HEAD
X=ACK FROM HEAD
'!'=ASCII CHARACTER OR CHARACTERS
0x21 HEX DATA EQUIVELENT
ADDRESS= TWO ASCII REPRESENTATIONS OF HEX CHARACTERS
\`x\`|\`y\` TWO ASCII CHARACTERS REPRESENTING THE UPPER AND LOWER
NIBBLE OF A HEXADECIMAL BYTE WHERE X IS THE UPPER NIBBLE AND
Y IS THE LOWER NIBBLE
FOR EXAMPLE:
TO SEND A SPEED OF 105 FEET PER MINUTE SEND
ASCII : (0x3a) AND ASCII 5 (0x35)
TO SEND A DELAY OF 30 SEND
ASCII 3 (0x33) AND ASCII 0 (0x30)

NOTE: THE \` CHARACTER AND | CHARACTER ARE NOT PART OF THE DATA STREAM AND ARE THERE FOR SEPARATION OF FIELDS ONLY.

CABLING FOR EVLINK ENVIRONMENT

C20552	RS232C to RS485 converter module
C20551	Cable from PC to RS485 converter module
C21008-xxxx	Cable (define length) from EVOLUTION units to RS485 data link
C21009	Termination plug for RS485 data link

HARDWARE INTERFACE

When connecting multiple printers via an RS485 link, input and output connectors are provided on the print station, which allows the cabling to be daisy chained. NOTE: It is important to remember to set each of the print stations to a unique address.

PHYSICAL CONNECTIONS RS485 PRINTER

Pin # 4	= Receive +
Pin # 5	= Receive -
Pin # 6	= Transmit +
Pin # 7	= Transmit -
Pin # 9	= Ground

Note: At the end of the data link a termination plug is installed to balance the RS485 data link-connecting pin 4 to pin 5 and pin 6 to pin 7 with120-ohm.

PROTOCOL FORMAT:

Host request for information;

```
    ESC|Command|SOH|EOT          (Single End Host to 1 printer)
   Or
   ESC|STX|Address|Command|SOH|EOT  (Multiple printers)
```

Host sending new information;

```
    ESC|Command|Data|EOT          (Single End Host to 1 printer)
   Or
   ESC|STX|Address|Command|Data|EOT  (Multiple printers)
```

EVOLUTION PRINTABLE CHARACTER SET

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9

Special Symbols:

ASCII Character	Hexadecimal	Prints As
Space	(0x20)	Space
!	(0x21)	Hour Glass
#	(0x23)	#
$	(0x24)	$
&	(0x26)	&
(	(0x28)	(
)	(0x29)	)
*	(0x2a)	*
+	(0x2b)	+
-	(0x2d)	-
.	(0x2e)	Period
=	(0x3d)	=
:	(0x3a)	:
/	(0x2f)	/
"	(0x22)	Cents
%	(0x25)	Solid block
;	(0x3b)	Ň
?	(0x3f)	Ě
@	(0x40)	Ó

Special Function Characters

`	(0x7b`	Large Logo 1
|	(0x7c)	Large Logo 2
`	(0x7d`	Large Logo 3
`	(0x7b`	Small Logo 1
|	(0x7c)	Small Logo 2
`	(0x7d`	Small Logo 3

NOTE: The same characters are used for a 2-line logo as is used for a 1-line logo. When the message is a 1**-line it accesses the logo from the** font table **memory map for a single line font, which is, where the large logos are stored. Conversely when the message is a 2-line then the logo is accessed from the memory map imbedded within the 2-line font table.**

SOFTWARE PROTOCOL

In the following pages, all references to characters or digits pertain to the standard ASCII character set. The bar (|) character is used as a field separator and it is not part of the transferred data. When data is shown in hexadecimal, it will consist of the hex number preceded by a 0x, for example (0x1B). Generally, all packets to and from a print station begin with an ESC (0x1B) and terminate with an EOT (0x04).

There are two types of commands:

Downloading information to the print station

Requesting information from the print station.

To distinguish the two types of commands, a SOH (0x01) is placed after the command byte in a request command string. The following illustrates this concept:

To download data to print station
ESC/GROUP ADDRESS/UNITADDRESS/COMMAND/DATA/EOT
To request data from the Print Station
ESC/GROUP ADDRESS/UNITADDRESS/COMMAND/SOH/EOT

NOTE: EACH REQUEST OR COMMAND SENT TO A PRINT STATION RECEIVES A RESPONSE FROM THAT PRINT STATION. COMMUNICATIONS SOFTWARE MUST WAIT FOR A RESPONSE TO DETERMINE IF THE PRINT STATION WAS READY TO ACCEPT THE COMMAND, AND THE DATA WAS VALID AND PROCESSED. NO RESPONSE COULD INDICATE THE DATA WAS LOST OR THE PRINTER WAS OCCUPIED PERFORMING A NON-INTERRUPTABLE TASK. IF AN ERROR WAS DETECTED DURING COMMUNICATIONS A NAK WITH AN ERROR CODE IS RETURNED. IN THE EVENT OF A NAK RESPONSE IT IS THE RESPONSIBILITY OF THE PROGRAMMER TO DETERMINE THE NATURE OF THE ERROR, CORRECT THE PROBLEM IF NECESSARY, AND RESEND THE COMMAND TO THE APPROPRIATE PRINTER. IT SHOULD NEVER BE ASSUMED THAT THE PRINTER RECEIVED THE DATA. VERIFICATION FROM THE PRINTER SHOULD ALWAYS BE TAKEN INT CONSIDERATION.

There is often confusion concerning how data is represented when transmitted within strings of text. As a general rule each character imbedded within a string is an ASCII character. Take for example the command for setting the printer address, which is the ASCII character B. The imbedded data requires two bytes of data they are ‘X’ and ‘Y’. When received by the printer these two bytes are concocted into an 8-bit byte. Therefore to set a printers address to 15 it is necessary to send two ASCII characters a HEX 31 (the number 1) and a HEX 35 (the number 5)

i.e. x = 0x31 & y = 0x35 yields unit address 15

ERROR CODES

Commands to a print station, if completed successfully, return a single byte response of an ASCII ACK (0x06). If the command was not successful, a two-byte response of an ASCII NAK (0x15) is returned, followed by an error code.
Below is a list of the returned error codes.
Both responses will be preceded with the printers address for further verification.

ACK command
ESC/GROUP ADDRESS/UNITADDRESS/ACK/EOT

NAK command
ESC/GROUP ADDRESS/UNITADDRESS/NAK/'ERROR CODE'/EOT
Where error code is a single byte ASCII 31 to 39

NAK 1	= PHYSICAL DATA ERROR
NAK 2	= ILLEGAL COMMAND BYTE
NAK 3	= MANUAL PRINT ATTEMPTED WHILE IN PRINTING MODE
NAK 4	= TRYING TO READ A WRITE ONLY FIELD
NAK 5	= TRYING TO WRITE A READ ONLY REGISTER
NAK 6	= PRINT STATION INPUT BUFFER FULL MUST PRINT BEFORE NEXT DOWNLOAD TO CLEAR INPUT BUFFER.
NAK 7	= SYSTEM BUSY – USER HAS SYSTEM THRU KEYBOARD
NAK 8	= SYSTEM BUSY – PRINTING FUNCTION
NAK 9	= BARCODE DOES NOT VERIFY

NOTE THE FOLLOWING COMMAND SET IS APPLICABLE TO ALL EVOLUTION MODELS EXCEPT WHERE NOTED. WHERE EV 1 OR EV 2 IS STATED IT REFERS TO BOTH EV 1 OR EV 2 AND LX 1 OR LX 2.

COMMANDS:

'!' 0x21 Software Version (read only) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`!`|SOH|EOT

R. ESC|STX|Address|`PRINTER fffffssss`|CR|EOT

Where:

PRINTER= ASCII string	PRINTER	for EVOLUTION I LX
	EV2	for EVOLUTION I LX
	EVSC	for EVOLUTION SC

fffff = Software and Firmware versions
(eg. 2.02H indicates version 2.02 with Firmware version H)
ssss = Optional Software loaded
Where: (for EV 1 only)
The first y indicates option pack 1 – OP1
The second y indicates option pack 2 – OP2
The third y indicates option pack 1.5 – OP1.5
The last y is reserved for option pack 3- OP 3
Where: (for EV 2 and EV SC)
Both units are standard with all options thus a ++++ will be returned

'#' 0x23 Printer Configuration (Read only) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`#`|SOH|EOT

R. ESC|STX|Address|`#`|`x`|`y`|EOT

Where Byte x Bits 3,2,1,0
Bit 3 = if 1 Cartridge Not Valid
Bit 2 = Not Used
Bits 1,0 = System Type
11 = EVOLUTION I LX
10 = Evolution 2
01 = Evolution 3
00 = Evolution Small Character

Where Byte y Bits 3,2,1,0
0000 = no options available
0001 = option1 enabled
0010 = option2 enabled
0100 = option1.5 enabled
1000 = option3 enabled

'\' 0x5c Unit Serial Number (Read only 6 digits) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`\`|SOH|EOT

R. ESC|STX|Address|`\`|`serial number`|CR|EOT

'l' 0x6c Special Field Flags (EV 2, EV SC AND EV 1 WITH OP1 AND ABOVE)

Q. ESC|STX|Address|`l`|SOH|EOT
R. ESC|STX|Address|`l`|`x`|`y'|EOT

Where: x defines bits 7,6,5,4
Bit 7 = don’t care
Bit 6 = dont care
Bit 5 = 1 = No guard bars
Bit 4 = 1 = Man read added to barcode

Where: y defines bits 3,2,1,0
Bit 3 = 1 = Bar checksum added to barcode
Bit 2 = 0 = Calendar will only change on 1st day of week
Bit 1 = 1 = Day of the week is alpha
Bit 0 = 1 = counting down

D. ESC|STX|Address|`l`|`x`|`y`|EOT
X. ESC|STX|Address|`l`|ACK|EOT

'8' 0x38 Control Flags (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`8`|SOH|EOT
R. ESC|STX|Address|`8`|`x`|`y`|EOT

Where: x defines bits 7,6,5,4
Bit 7 1 = Head busy printing message
Bit 6 1 = Print image inverted
Bit 5 1 = Head busy manual cycle
Bit 4 1 = Head busy purging

Where: y defines bits 3,2,1,0
Bit 3 1 = External Encoder
Bit 2 1 = External Product Detect
Bit 1 1 = Direction forward
Bit 0 1 = Enable PRINTING Mode

D. ESC|STX|Address|`8`|`x`|`y`|EOT
X. ESC|STX|Address|`8`|ACK|EOT

Where: x defines bits 7,6,5,4
Bit 7 Don't Care
Bit 6 1 = Print image inverted
Bit 5 Don't Care
Bit 4 Don't Care

Where: y defines bits 3,2,1,0
Bit 3 1 = External Encoder
Bit 2 1 = External Product Detect
Bit 1 1 = Direction forward
Bit 0 1 = Enable PRINTING Mode

‘G’ 0x47 Errors (note: error codes must be reset) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`G`|SOH||EOT

R. ESC|STX|Address|`G`|'x`|`y'|EOT

Where: x defines bits 7,6,5,4

Bit 7 = UART Overrun Error
Bit 6 = Communication Overrun Error
Bit 5 = UART Framing Error
Bit 4 = UART Parity Error

Where: y defines bits 3,2,1,0

Bit 3 = Font checksum error loading from card to chip
Bit 2 = Font 1 checksum error in Ram
Bit 1 = Font 0 checksum error in Ram
Bit 0 = Real Time Clock Memory error

TO RESET ERROR CODES

D. ESC|STX|Address|`G`|'x`|`y'|EOT

same bit positions as above
use only as a mask to clear error bits.
i.e. x = 0001 and y = 0001 clears real time clock memory error and UART parity error.

X. ESC|STX|Address|`G`|ACK|EOT

'R' 0x52 Head Status (read only) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`R`|SOH|EOT

R. ESC|STX|Address|`R`|`x`|`y`|EOT

Where: x defines bits 7,6,5,4

Bit 7 = Not Used
Bit 6 = Latched eye active
Bit 5 = Unfiltered eye active
Bit 4 = Product being printed

Where y defines bits 3,2,1,0

Bit 3 = auto repeat print gap active
Bit 2 = Not Used
Bit 1 = Input buffer Line 2 full
Bit 0 = Input buffer Line 1 full

'U' 0x55 General purpose flags (read only) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`U`|SOH|EOT
R. ESC|STX|Address|`U`|`y`|EOT
Where y defines bits 3,2,1,0
Bit 3 = Not Used
Bit 2 = Not Used
Bit 1 = Ink Cartridge Empty
Bit 0 = Mixed Raster Enabled

'B' 0x42 Set Unit Address (Write Only) (EV 1, EV 2, EV SC)

D. ESC|STX|Address|`B`|`x`|`y`|EOT
X. ESC|STX|Address|`B`|ACK|EOT
Where x y = 8 bit unit address
i.e. x = 0x31 & y = 0x35 yields unit address 15

'1' 0x31 Auto Repeat Inter-print delay (Range 0 - 255) (EV 2, EV SC AND EV 1 with any option pack)

Q. ESC|STX|Address|`1`|SOH|EOT
R. ESC|STX|Address|`1`|`x`|`y`|EOT

D. ESC|STX|Address|`1`|`x`|`y`|EOT
X. ESC|STX|Address|`1`|ACK|EOT
0 = Auto Repeat Disabled
Each count provides a delay equal to 16 columns for EV 1 and EV 2.
Each count provides a delay equal to 2 columns for EV SC.

'&' 0x26 Line Speed (RANGE 10-200) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`&`|SOH|EOT
R. ESC|STX|Address|`&`|`x`|`y`|EOT

D. ESC|STX|Address|`&`|`x`|`y`|EOT
X. ESC|STX|Address|`&`|ACK|EOT

'd' 0x64 Encoder Divider (Range 0-7) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`d`|SOH|EOT
R. ESC|STX|Address|`d`|`x`|`y`|EOT

D. ESC|STX|Address|`d`|`x`|`y`|EOT
X. ESC|STX|Address|`d`|ACK|EOT

'’' 0x27 Product Delay (RANGE 1-255) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`0x27`|SOH|EOT
R. ESC|STX|Address|`0x27`|`x`|`y`|EOT

D. ESC|STX|Address|`0x27`|`x`|`y'|EOT
X. ESC|STX|Address|`0x27`|ACK|EOT

')' 0x29 Inter-Character spaces (RANGE 1-25) (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`)`|SOH|EOT
R. ESC|STX|Address|`)`|`x`|`y`|EOT

D. ESC|STX|Address|`)`|`x`|`y'|EOT
X. ESC|STX|Address|`)`|ACK|EOT

'>' 0x3E Head Align (Range 0 - 16) 'O' on keyboard (EV 2 only)

Q. ESC|STX|Address|`>`|SOH|EOT
R. ESC|STX|Address|`>`|`x`|`y'|EOT

D. ESC|STX|Address|`>`|`x`|`y`|EOT
X. ESC|STX|Address|`>`|ACK|EOT

'4' 0x34 Sequence Number Rollover Value (EV 2, EV SC AND EV 1 with version 2.09 and OP2 or 3)

Q. ESC|STX|Address|`4`|SOH|EOT
R. ESC|STX|Address|`4`|`#########`|CR|EOT
where ######### = rollover value in ascii (max 9 digits)

D. ESC|STX|Address|`4`|`#########`|CR|EOT
X. ESC|STX|Address|`4`|ACK|EOT

'^' 0x5E Lot Counter Limit Count (EV 2, EV SC AND EV 1 with version 2.09 and OP2 or 3)

Q. ESC|STX|Address|`^`|SOH|EOT
R. ESC|STX|Address|`^`|`###`|CR|EOT
where ### = rollover value in ascii (max 4 digits)

D. ESC|STX|Address|`^`|`####`|CR|EOT
X. ESC|STX|Address|`^`|ACK|EOT

'_' 0x5F Lot Counter Value (read only) (EV 2, EV SC AND EV 1 with version 2.09 and OP2 or 3)

Q. ESC|STX|Address|`_`|SOH|EOT
R. ESC|STX|Address|`_`|`###`|CR|EOT
where ### = current count value in ascii (max 4 digits)

D. ESC|STX|Address|`_`|`####`|CR|EOT
X. ESC|STX|Address|`_`|ACK|EOT

'[' 0x5b DATE_ROLLOVER

(EV 2, EV SC AND EV 1 with version 2.09 and OP2 or 3)

Q. ESC|STX|Address|`[`|SOH|EOT
R. ESC|STX|Address|`[`|`x`|`y`|`x1`|`y1`|EOT
Where:
|`x`|`y`| = Time of Day Hours
|`x1`|`y1`| = Time of Day Minutes

D. ESC|STX|Address|`[`|`x`|`y`|`x1`|`y1`|EOT
X. ESC|STX|Address|`[`|ACK|EOT

‘3’ 0X31 Expiration Days 1 (max 999)
or
‘@’ 0X31 Expiration Days 2 (max 999)

(EV 2, EV SC AND EV 1 WITH OP3)

Q. ESC|STX|Address|`3`|SOH|EOT
R. ESC|STX|Address|`3`|`aaaa`|EOT
Where: each set of 2 ASCII characters represent the upper and lower nibble of a packed BCD byte

D. ESC|STX|Address|`3`|aaaa`|EOT
Where: each set of 2 ASCII characters represent the upper and lower nibble of a packed BCD byte
X. ESC|STX|Address|`3`|ACK|EOT

'r' 0x72 Remaining Ink (0 to 99%)

(EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`r` |SOH|EOT
R. ESC|STX|Address|`r`|`x`|`y`|EOT

‘0’ 0x30 Shift Code (max 6 shift codes)
(EV 2, EV SC AND EV 1 WITH OP3)

Q. ESC|STX|Address|`0`|SOH||EOT
R. ESC|STX|Address|`0`|`hh mm`|`zz`|……|CR|EOT
Where: each set of 2 ASCII characters represent the upper and lower nibble of a packed BCD byte

……	= pattern repeat for each shift code programmed
hh	= shift start hours
mm	= shift start minutes
zz	= shift code to print

D. ESC|STX|Address|`0`|`hhmm`|`z`|CR|EOT
Where: each set of 2 ASCII characters represent the upper and lower nibble of a packed BCD byte
hh = shift start hours
mm = shift start minutes
zz = shift code to print
X. ESC|STX|Address|`0`|ACK|EOT

‘/’ 0x2f Product Counter (6 Digits Max) (EV 2, EV SC AND EV 1 WITH OP3)

Q. ESC|STX|Address|`/`|SOH|EOT

R. ESC|STX|Address|`/`|`HH MM hh mm`|`cccccc`|CR|EOT

Where: each set of 2 ASCII characters represent the upper and lower nibble of a packed BCD byte

HH = Product counter start hours
MM = Product counter start minutes
hh = Product counter stop hours
mm = Product counter stop minutes
cccccc = counter (6 Digits Max)

D. ESC|STX|Address|`/`|`ww xx yy zz`|`cccccc`|CR|EOT

Where: each set of 2 ASCII characters represent the upper and lower nibble of a packed BCD byte

HH = Product counter start hours
MM = Product counter start minutes
hh = Product counter stop hours
mm = Product counter stop minutes
cccccc = counter

X. ESC|STX|Address|`/`|ACK|EOT

'6' 0x36 Cycle Head (Write Only)

(EV 1, EV 2, EV SC)

D. ESC|STX|Address|`6` |SOH|EOT

R. ESC|STX|Address|`6`|ACK|EOT

'`' 0x60 Print Column Configuration (All Lexmark Models)

Q. ESC|STX|Address|```|SOH|EOT

R. ESC|STX|Address|```|`#`|CR|EOT

where 1 = Column 1
where 2 = Column 2
where 3 = Column 3
where 4 = Column 4
where 5 = Columns 1&2 (600 DPI)
where 6 = Columns 3&4 (600 DPI)
where 7 = Columns 1,2 ,3 ,4 sequencing each print cycle
where 7 = Column 1&2, 3&4 sequencing each print cycle (600 DPI)

D. ESC|STX|Address|```|`#`|CR|EOT

X. ESC|STX|Address|```|ACK|EOT

SPECIAL FIELD OBJECTS

Message Objects define special characteristics about the messages contained in line 1 or line 2. These may define for example font size, sequence number, date code, etc. There may be up to 15 Objects (special fields) for each line in a message with the limitation that there can only be 1 sequence number imbedded in a message.

'P' 0x50 Message Objects (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`P`|SOH|aabb|EOT

R. ESC|STX|Address|`P`|`aa bb cc dd ee ff gggg hhhh`|EOT

Where: each set of 2 ASCII characters represent the upper and lower nibble of a byte
aa = objects for which line 0 or 1
bb = number of objects transmitted. (Max 15)

Each object as defined by bb: (repeat the for each object)
cc = Position within message string
dd = Number of characters in object
ee = Attribute of the object
Where:

ee= 00 Normal Alpha/Numeric character
ee= 01 Time Hours
ee= 02 Time Minutes
ee= 03 Time Seconds
ee= 04 Date Month
ee= 05 Date Day
ee= 06 Date Year
ee= 07 Date Julian
ee= 08 Sequence Number (1 per message)
ee= 09 Barcode
ee= 0A Shift Code
ee= 0B Expiration Date Month
ee= 0C Alpha Date Code
conflict shows lot code ee= 0D Expiration Date Year
ee= 0E Expiration Date Julian
ee= 0F Expiration Date Day
ee= 10 Day of Week (1-7)
ee= 12 Expiration 2 Date Month in Alpha
ee= 13 Expiration 2 Date Year
ee= 14 Expiration 2 Date Month
ee= 15 Expiration 2 Date Julian
ee= 16 Expiration 2 Date Day
ee= 40 Valid Bar Code (EV 2 only) OR’d with other Attributes
ee= 80 Bar Code Attribute (EV 2 only) OR’d with other Attributes

'P' 0x50 Message Objects (continued)

ff = font of object

Where: for EV 1 AND EV 2

- ff= 00 for 2 Line Font
- ff= 01 for 1 Line Font
- ff= 02 for 3 Line Font (EV 2 only)
- ff= 03 for 4 Line Font (EV 2 only)

Where: for EVSC ONLY

- ff= 00 for S5 Font
- ff= 01 for S7 Font
- ff= 02 for B7 Font
- ff= 03 for S12 Font
- ff= 04 for B12 Font

gggg = starting column of object in printed image (reserved)

hhhh = starting row of object in printed image (reserved)

D. ESC|STX|Address|`P`|`aa bb cc dd ee ff gggg hhhh`|EOT

X. ESC|STX|Address|`P`|ACK|EOT

Even though there up to 24 characters (48 characters for LX1 with OP 1.5 and above or LX 2) permitted per line data entry will be inhibited when the 15th object is entered, although the last field, if it is an alpha/numeric object, may contain enough characters to meet the max character limit.

Barcodes are also an object field and must be considered when entering a message. Thus a barcode with imbedded variable field data would be counted as two or mode objects.

NOTE: PRINTER MAX CHARACTERS PER LINE
(EV 1 max 24 characters – 48 characters OP1.5, 2 or 3)
(EV 2 max 48 characters)
(EV SC max 96 characters)

'$' 0x24 Line 1 Message

Q. ESC|STX|Address|`$`|SOH|EOT
R. ESC|STX|Address|`$`|`message`|CR|EOT

D. ESC|STX|Address|`$`|`message`|CR|EOT
X. ESC|STX|Address|`$`|ACK|EOT

'%' 0x25 Line 2 Message

Q. ESC|STX|Address|`%`|SOH|EOT
R. ESC|STX|Address|`%`|`message`|CR|EOT

D. ESC|STX|Address|`%`|`message`|CR|EOT
X. ESC|STX|Address|`%`|ACK|EOT

'w' 0x77 Line 3 Message (EV 2 only max 48 characters)

Q. ESC|STX|Address|`$`|SOH|EOT
R. ESC|STX|Address|`$`|`message`|CR|EOT

D. ESC|STX|Address|`$`|`message`|CR|EOT
X. ESC|STX|Address|`$`|ACK|EOT

'z' 0x7a Line 4 Message (EV 2 only max 48 characters)

Q. ESC|STX|Address|`$`|SOH|EOT
R. ESC|STX|Address|`$`|`message`|CR|EOT

D. ESC|STX|Address|`$`|`message`|CR|EOT
X. ESC|STX|Address|`$`|ACK|EOT

'E' 0x45 Line 5 Message (Prefix line)

Q. ESC|STX|Address|`$`|SOH|EOT
R. ESC|STX|Address|`$`|`message`|CR|EOT

D. ESC|STX|Address|`$`|`message`|CR|EOT
X. ESC|STX|Address|`$`|ACK|EOT

NOTE: TO ENTER A LOGO CALLOUT INTO A MESSAGE USE THE ACSII CHARACTERS 0x7B FOR LOGO1 0x7C FOR LOGO 2 AND 0x7D FOR LOGO 3

':' 0x3A Logo1 Name (read only - max 9 characters) (EV 1, EV 2)

Q. ESC|STX|Address|`:`|SOH|`x`|`y`|EOT
R. ESC|STX|Address|`:`|`logo name`|CR|EOT

```
Where: x = don’t care
       y = Bit 0 =   0 = Logo Name in Font 0
                     1 = Logo Name in Font 1
           Bit 1 =   0 = Get Name from on board data flash chip
                     1 = Get Name fro Data Flash card
```

';' 0x3B Logo2 Name (read only - max 9 characters) (EV 1, EV 2)

Q. ESC|STX|Address|`;`|SOH|`x`|`y`|EOT
R. ESC|STX|Address|`;`|`logo name`|CR|EOT

```
Where: x = don’t care
       y = Bit 0 =   0 = Logo Name in Font 0
                     1 = Logo Name in Font 1
           Bit 1 =   0 = Get Name from on board data flash chip
                     1 = Get Name fro Data Flash card
```

'<' 0x3C Logo3 Name (read only - max 9 characters) (EV 1, EV 2)

Q. ESC|STX|Address|`<`|SOH|`x`|`y`|EOT
R. ESC|STX|Address|`<`|`logo name`|CR|EOT

```
Where: x = don’t care
       y = Bit 0 =   0 = Logo Name in Font 0
                     1 = Logo Name in Font 1
           Bit 1 =   0 = Get Name from on board data flash chip
                     1 = Get Name fro Data Flash card
```

'Q' 0x51 Starting Sequence Number (max. length 9 digits) (EV 2, EV SC AND EV1 with version 2.09 and after)

Q. ESC|STX|Address|`Q`|SOH|EOT
R. ESC|STX|Address|`Q`|`zzzzzzzzz`|CR|EOT

Where:

zzzzzzzzz = ASCII string which is the starting sequence number to print.

D. ESC|STX|Address|`Q`|`zzzzzzzzz`|CR|EOT
X. ESC|STX|Address|`Q`|ACK|EOT

'2' 0x32 Date and Time Setting / Reading (EV 1, EV 2, EV SC)

Q. ESC|STX|Address|`2`|SOH|EOT
R. ESC|STX|Address|`2`|`aa bb cc dd ee ff gg`|EOT

Where: each set of 2 ASCII characters represent the upper and lower nibble of a packed BCD byte
aa= Time of Day Seconds (not used)
bb= Time of Day Minutes
cc= Time of Day Hours
dd= Day of Week
ee= Date Day
ff = Date Month
gg= Date Year

D. ESC|STX|Address|`2`|`aa bb cc dd ee ff gg`|CR|EOT
X. ESC|STX|Address|`2`|ACK|EOT

'u' 0x75 Store message in non-volatile memory (Write only) (EV 1, EV 2, and EV SC)

D. ESC|STX|Address|`u`| EOT
X. ESC|STX|Address|`u`|ACK|EOT

NOTE: THE FOLLOWING CODES ARE SPECIFIC TO EV 2

' " ' 0x22 Minimum Bar Width (Range 3-15 Data matrix 2-15)

Default 5

Q. ESC|STX|Address|`"`|SOH|EOT
R. ESC|STX|Address|`"`|`x`|`y`|EOT

D. ESC|STX|Address|`"`|`x`|`y`|EOT
X. ESC|STX|Address|`"`|ACK|EOT

' .' 0x2e Bleed Compensation (Range 0 - 3) Default 0

Q. ESC|STX|Address|`.`|SOH|EOT
R. ESC|STX|Address|`.`|`x`|`y`|EOT

D. ESC|STX|Address|`.`|`x`|`y`|EOT
X. ESC|STX|Address|`.`|ACK|EOT

' * ' 0x28 Quiet Zone (Range 0 - 150) Default 75

Q. ESC|STX|Address|`*`|SOH|EOT
R. ESC|STX|Address|`*`|`x`|`y`|EOT

D. ESC|STX|Address|`*`|`x`|`y`|EOT
X. ESC|STX|Address|`*`|ACK|EOT

'n' 0x6e Type of Barcode (read only)

Q. ESC|STX|Address|`n`|SOH|EOT
R. ESC|STX|Address|`n`|`x`|`y`|EOT

where

x = number of available barcodes
y = type of barcode

0= CODE39
1= TWO OF FIVE
2= CODE 128B
3= CODE 128C
4= UPCA
5= UPCE
6= EAN8
7= EAN13
8= DATAMATRIX

'?' 0x3F Barcode Name(read only)

Q. ESC|STX|Address|`?`|SOH|`x`|`y`|`x1`|`y1`|EOT

Where:

`x``y` = Barcode type as in 'n' command

`x1`|`y1` = don't care

R. ESC|STX|Address|`?`|`BARCODENAME`|CR|EOT

where BARCODENAME = Ascii name of type of barcode

'=' 0x3d Barcode Verify

D. ESC|STX|Address|`=`|`x`|`y`|`BARCODESTRING`|CR|EOT

x = don't care

y = type of barcode (same as 'n' command)

BARCODESTRING = Barcode Ascii data

X. ESC|STX|Address|`=`|`xy`|EOT

where

if barcode verifies

ESC|STX|Address|`=`|ACK|EOT

if barcode doesn't verify

ESC|STX|Address|`=`|NAK|`9`|EOT

EXAMPLE WRITTEN IN C

to query a print station to determine the line speed.

INITIALIZE AND OPEN A SERIAL CHANNEL
EXECUTE THE FOLLOWING CODE

```
// Query Print Station Address 7 for Line Speed
        putchar(0x1b);          // Send out ESC
        putchar(0x02);          // Send out STX
        putchar(0x30);          // Send out upper nibble of address 07
        putchar(0x37);          // Send out lower nibble of address 07
        putchar(0x26);          // Send out a '&' command
        putchar(0x01);          // Send out SOH
        putchar(0x04);          // Send out EOT

        // Get results from print station
        `
        unsigned char dummy,speed;

                dummy = getchar();                      // Get ESC
                dummy = getchar();                      // Get STX
                dummy = getchar() << 4;                 // Get upper nibble of address
                dummy |= getchar() & 0x0f;              // Get lower nibble of address
                if(dummy == our_address)
                `
                        dummy = getchar();              // Get command
                        speed = getchar() << 4;         // Get upper nibble of speed
                        speed |= getchar() & 0x0f;      // Get lower nibble of speed
                        dummy = getchar();              // Get EOT
                ` else `
                        // error handler (not our address)
                `
        `
```

Example written in C to send a line speed to a print station

INITIALIZE AND OPEN A SERIAL CHANNEL
EXECUTE THE FOLLOWING CODE

```
// Send Print Head Address 2 Line Speed of 100 feet per minute.
        putchar(0x1b);          // Send out ESC
        putchar(0x02);          // Send out STX
        putchar(0x30);          // Send out upper nibble of address
        putchar(0x32);          // Send out lower nibble of address
        putchar(0x26);          // Send out '&' command
        putchar(0x36);          // Send out upper nibble for Line Speed 100
        putchar(0x34);          // Send out lower nibble for Line Speed 100
        putchar(0x04);          // Send out EOT

        // Get results from print station
        `
        unsigned char dummy;

                dummy = getchar();                      // Get ESC
                dummy = getchar();                      // Get STX
                dummy = getchar() << 4;                 // Get upper nibble of address
                dummy |= getchar() & 0x0f;              // Get lower nibble of address
                if(dummy == our_address)
                `
                        dummy = getchar();              // Get command
                        dummy = getchar();              // Get ACK for print station
                        if(!dummy == ACK)
                        `
                                // error handler (didn't get acknowledgement from printer)
                        ` else `
                                dummy = getchar();      // Get EOT
                        `
                ` else `
                        // error handler (not our address)
                `
        `
```

EXAMPLE WRITTEN IN VB

to send a new message to a print station.

INITIALIZE AND OPEN A SERIAL CHANNEL
EXECUTE THE FOLLOWING CODE

```
Public Sub DoMessage()
DATA$ = "800": GETINFODATA: Rem DISABLE PRINTING MODE
DATA$ = "&32": GETINFODATA: Rem SET LINE SPEED TO 50
DATA$ = "P01010010000100000000" & Chr$(&HD): GETINFODATA: Rem SET OBJECTs
DATA$ = "%ABCDEFGHIJ" & Chr$(&HD): GETINFODATA: Rem SEND MESSAGE
End Sub

Public Sub GETINFODATA() : :  Rem SENDS A COMMAND AND GETS A RESPONSE
RESPONSE$ = "": COMM.InBufferCount = 0
COMM.Output = ESC & STX & "01" & DATA$ & EOT
Timer.Enabled = True: TIMERFLAG = False
GETINFO:
   Do
      DoEvents
      If TIMERFLAG = True Then GoTo TCOMMERROR
   Loop Until COMM.InBufferCount >= 1
      RESPONSE$ = RESPONSE$ & COMM.Input
      If InStr(RESPONSE$, Chr$(&H15)) > 0 Then GoTo GETDATAERROR:
Rem A NAK WAS RECEIVED
      If InStr(RESPONSE$, Chr$(&H4)) = 0 Then GoTo GETINFO
Rem AN EOT WAS RECEIVED
      RESPONSE$ = Mid$(RESPONSE$, 6, Len(RESPONSE$))
Rem DELETE ADDRESS HEADER
      Timer.Enabled = False
Rem WE NOW HAVE A VALID RESPONSE
      Exit Sub
GETDATAERROR:
   Timer.Enabled = False: TIMERFLAG = False
   GoTo PROCESSERROR
   Exit Sub
TCOMMERROR:
   Timer.Enabled = False: TIMERFLAG = False
PROCESSERROR:
If RESPONSE$ = "" Then RESPONSE$ = "0" Else RESPONSE$ = Right$(RESPONSE$, 1):
Rem GET THE ERROR CODE
Select Case (RESPONSE$)
   Case 0
      MSG$ = "NO RESPONSE FROM UNIT"
   Case 1
      MSG$ = "TRANSMISSION ERROR"
   Case 2
      MSG$ = "ILLEGAL COMMAND"
   Case 3
      MSG$ = "TRYING TO PRINT WHILE IN COMMAND MODE"
   Case 4
      MSG$ = "TRYING TO READ A WRITE ONLY REGISTER"
   Case 5
      MSG$ = "TRYING TO WRITE A READ ONLY REGISTER"
   Case 6
      MSG$ = "UNIT INPUT BUFFER FULL"
```

```
 Case 7
    MSG$ = "UNIT IN EDIT MODE"
 Case 8
    MSG$ = "PRINT STATION BUSY TRY AGAIN"
 End Select
 MsgBox MSG$
 COMM.InBufferCount = 0: Rem FLUSHES THE INPUT BUFFER
End Sub
```

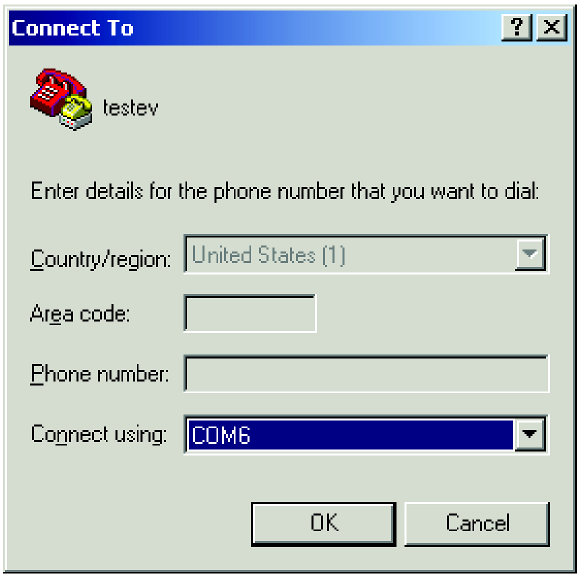
THE ABOVE VB ROUTINES DEMONSTRATE THE ENTIRE SEQUENCE OF:

PREPARING DATA TO SEND TO THE HEAD
SENDING THE DATA TO THE HEAD
WAIT FOR A RESPONSE
DETERMINE IF THE DATA WAS ACCEPTED OR REJECTED

EXAMPLE USING HYPER TERMINAL

Preliminary test of the device data link can be performed using the standard HYPER TERMINAL supplied with windows.

NOTE: it is assumed that the user has already installed a RS485 adapter and has verified the device address (COMM PORT) that device is attached to.


Connect To

testev

Enter details for the phone number that you want to dial:

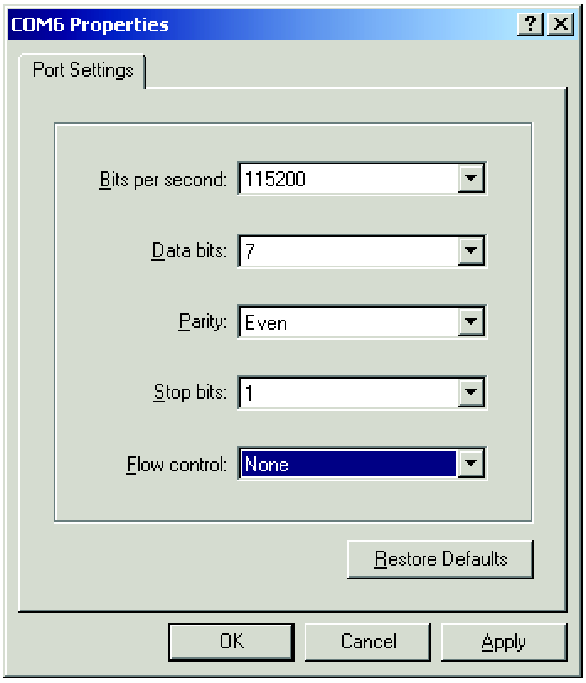
Country/region: United States (1)

Area code:

Phone number:

Connect using: COM6

Select the appropriate COM port


COM6 Properties

Port Settings

Bits per second: 115200

Data bits: 7

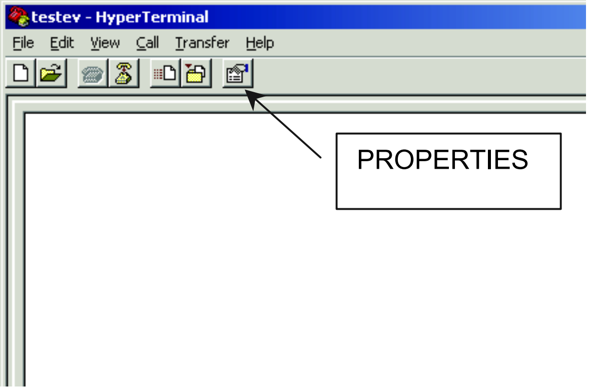
Parity: Even

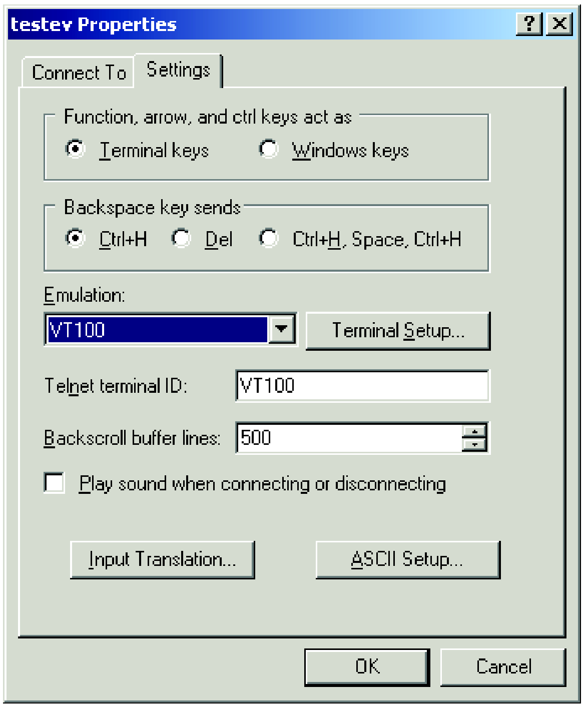
Stop bits: 1

Flow control: None

Set the baud rate, data bits, parity, Stop bits and flow control.

Select properties and enter the following settings.


testev - HyperTerminal



testev Properties

Connect To | Settings

Function, arrow, and ctrl keys act as

● Terminal keys ○ Windows keys

Backspace key sends

● Ctrl+H ○ Del ○ Ctrl+H, Space, Ctrl+H

Emulation: VT100

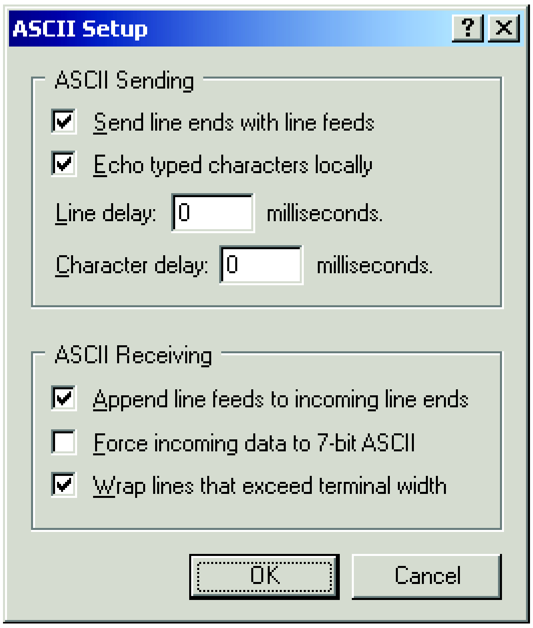
Telnet terminal ID: VT100

Backscroll buffer lines: 500

□ Play sound when connecting or disconnecting

Set terminal keys checked. Select VT100 for the emulation mode as pictured on the left.

You can save your setup for future usage.
There are several keys that are required as control characters. They are as follows:
(The ^ character represents the control key in conjunction with the key shown)
^A THE SOH CHARACTER
^B THE STX CHARACTER
^D THE EOT CHARACTER
ESC the actual key on the keyboard



ASCII Setup

ASCII Sending

☑ Send line ends with line feeds

☑ Echo typed characters locally

Line delay: 0 milliseconds.

Character delay: 0 milliseconds.

ASCII Receiving

☑ Append line feeds to incoming line ends

□ Force incoming data to 7-bit ASCII

☑ Wrap lines that exceed terminal width

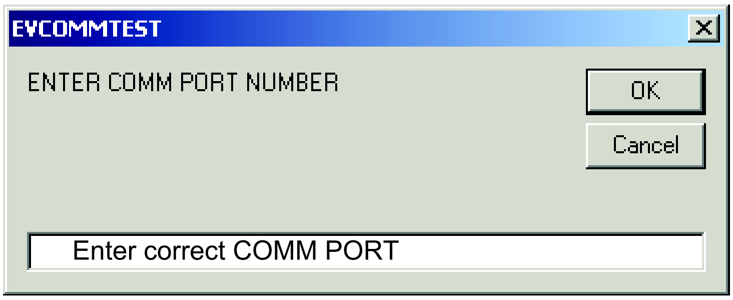
Select ASCII setup and set the appropriate check boxes.

Test the link by typing: spaces are shown for clarification and are not included.
ESC ^B 01 ! ^A ^D
The printer will respond with the printer's software/firmware information.
NOTE: NOT ALL CHARACTERS ARE DISPLAYED ON THE SCREEN.

EXAMPLE USING EVCOMMTEST

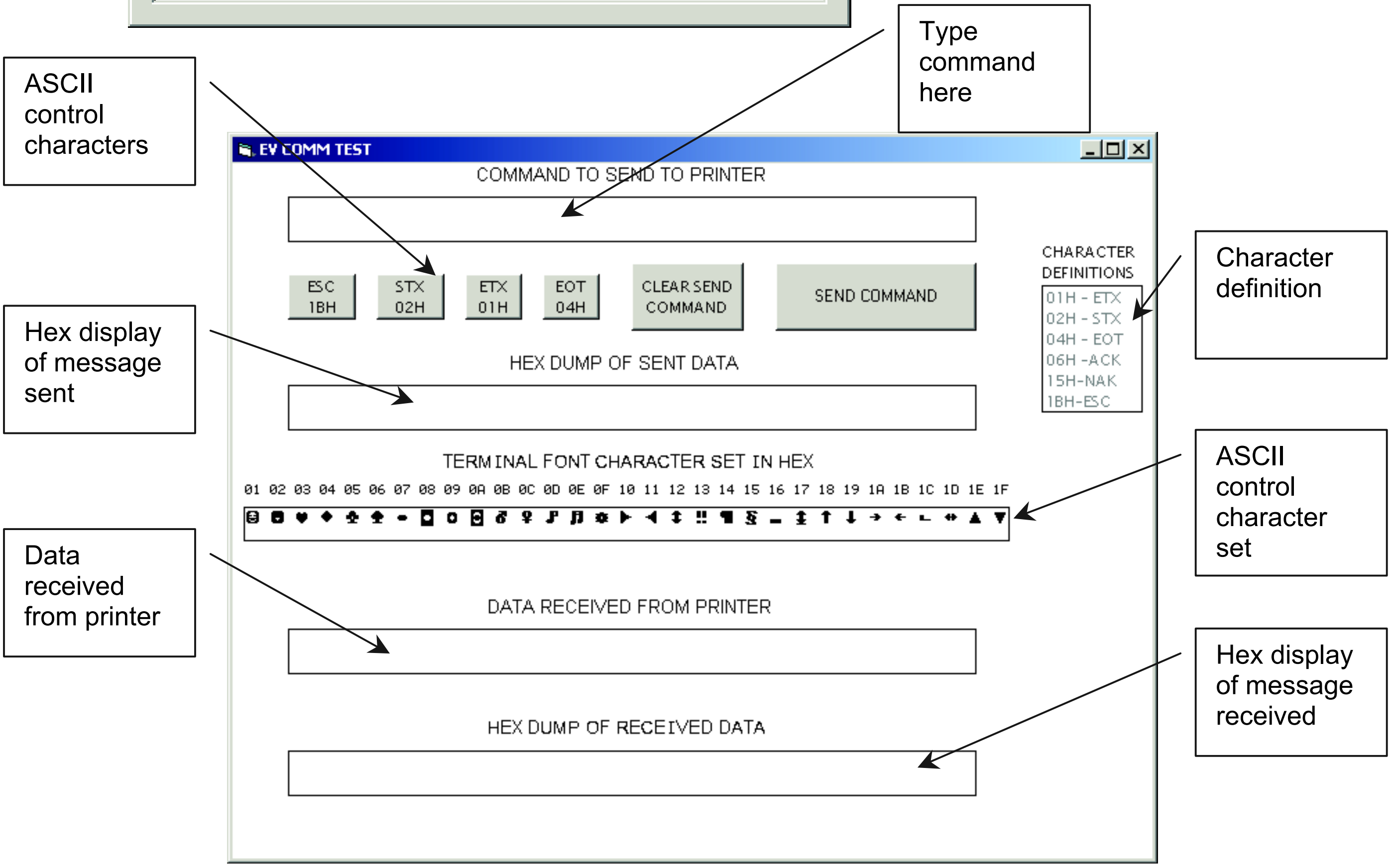
Hyper Terminal has a drawback in that it does not normally display the ASCII control character set, specifically those characters between a hex 01 through a hex 1F. The following describes a simple program written in visual basic (VB5) that provides the programmer with a clearer definition of the ASCII control character sequence. The program is included on this CD manual and may be found in the sub-directory "EVCOMMTEST".
The user should find the correct location and install the program by invoking "SETUP.EXE".

NOTE: it is assumed that the user has already installed a RS485 adapter and has verified the device address (COMM PORT) that device is attached to.



Enter correct COMM PORT

Program initiation displays the COMM PORT selection screen. The user should enter the correct COMM PORT and select "OK".


EV COMM TEST
COMMAND TO SEND TO PRINTER
ESC 1BH
STX 02H
ETX 01H
EOT 04H
CLEAR SEND COMMAND
SEND COMMAND
HEX DUMP OF SENT DATA
TERMINAL FONT CHARACTER SET IN HEX
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
DATA RECEIVED FROM PRINTER
HEX DUMP OF RECEIVED DATA

CHARACTER DEFINITIONS
01H - ETX
02H - STX
04H - EOT
06H -ACK
15H-NAK
1BH-ESC

Type command here

ASCII control characters

Character definition

Hex display of message sent

ASCII control character set

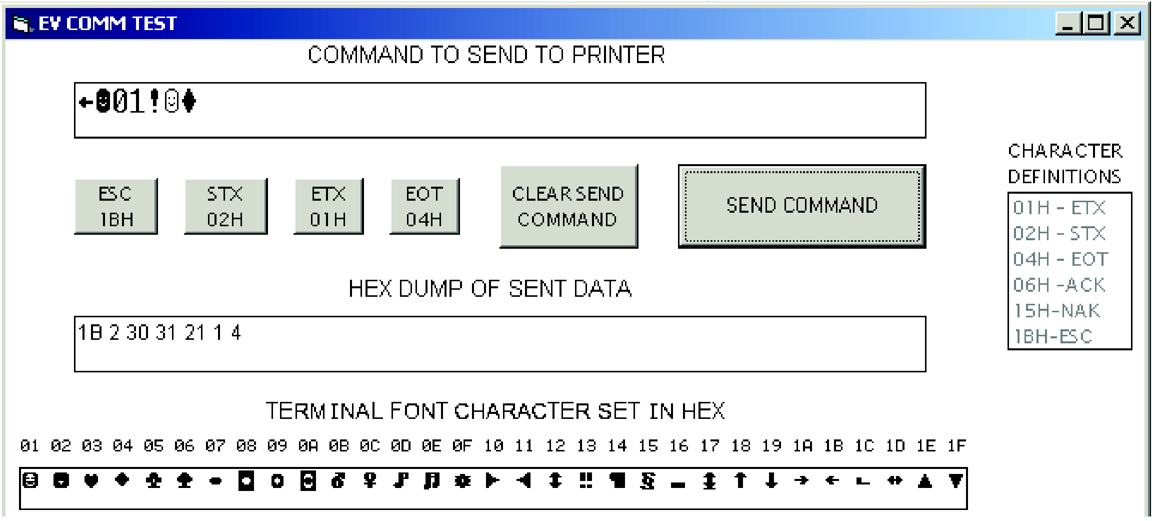
Data received from printer

Hex display of message received

After a comm. Port is selected the above screen is displayed. This program allows for simple command entry and provides the programmer a detailed view of both the characters sent and the data received from the printer.

Commands to be sent to the printer are typed in first text box labeled “COMMAND TO BE SENT TO PRINTER”. Since windows does not display ASCII control characters, and often interprets these control characters to have special meaning data entry for the special characters are in the form of icons on the screen. The second line of the display shows 6 gray-scaled boxes. The first 4 boxes represent the ESC, STX, ETX and EOT ASCII control characters. The “CLEAR SEND COMMAND’ is intended to clear the first line text box. Of course any other windows erasing method will work. Finally the last box “SEND COMMAND” actually sends the command typed into the first line to the printer.

To enter a command string the user must follow the correct sequence. All command structures sent to a printer must start with the ESC character so select the ESC gray scaled box. Next select the STX box and then place the cursor in the command text box following the second character, which was the STX.
After the cursor is placed in the command text box it is necessary to enter the unit ID (device address) of the printer. For initial test purposes use the factory default setting. This default address is 01 and should be typed in the command box following the STX. The required command follows the address. Selecting the ETX box followed by the EOT box ends the command string. Select the “SEND COMMAND” button to send the command to the printer.



COMMAND TO SEND TO PRINTER

←☻01!☺♦

HEX DUMP OF SENT DATA

1B 2 30 31 21 1 4

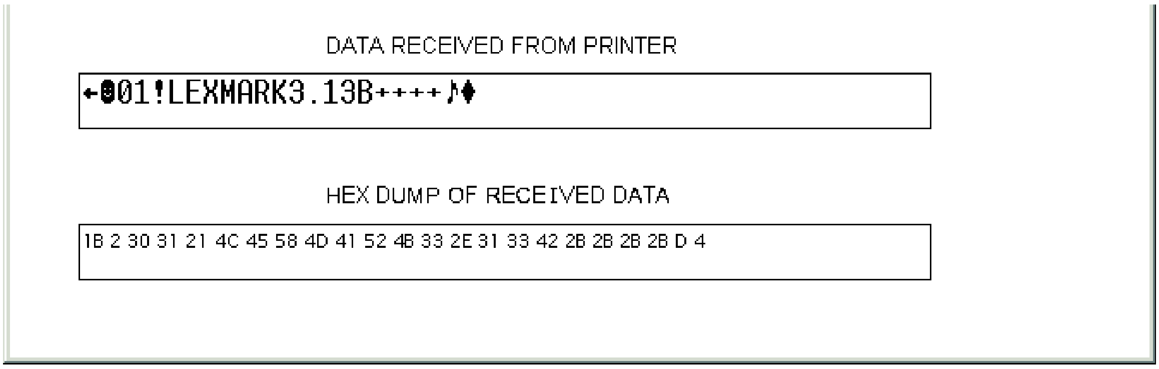
TERMINAL FONT CHARACTER SET IN HEX

```
01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
☺  ☻  ♥  ♦  ♣  ♠  •  ◘  ○  ◙  ♂  ♀  ♪  ♫  ☼  ►  ◄  ↕  ‼  ¶  §  ▬  ↨  ↑  ↓  →  ←  ∟  ↔  ▲  ▼
```

CHARACTER DEFINITIONS

01H - ETX
02H - STX
04H - EOT
06H -ACK
15H-NAK
1BH-ESC

When the “SEND COMMAND” button is selected the second text line displays the hex values of the message sent to the printer. In this case the ASCII character ! (Exclamation Point) was sent to the printer. This command requests the printer to respond with the software version of the system.
For convenience the third text line displays all the ASCII control codes with the hex values above them.



DATA RECEIVED FROM PRINTER

```
←☻01!LEXMARK3.13B++++♪♦
```

HEX DUMP OF RECEIVED DATA

```
1B 2 30 31 21 4C 45 58 4D 41 52 4B 33 2E 31 33 42 2B 2B 2B 2B D 4
```

The bottom half of the display shows the ASCII data received from the printer and the hex values for the received data displayed in the last text box.
The above response shows the printer is a LEXMARK with software version 3.13, a “B” firmware rev, and all options installed.

The programmer should try several commands to become familiar with the typical responses received from the printer. It should also be noted that the actual returned data might be in packed BCD format, or a hex value that needs to be translated.