PART 6: COMMUNICATIONS PROTOCOL

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 a specific unit. Commands that reference specific units are so noted.

ASCII CHARACTER CHART

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

DESCRIPTION

This communication protocol is based on Version 1.4, which was 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 165 FEET PER MINUTE SEND – ASCII : AND ASCII 5 WHICH WOULD BE 0x3A AND 0x35 HEXADECIMAL
TO SEND A DELAY OF 30 SEND – ASCII 3 AND ASCII 0 WHICH WOULD BE 0x33 AND 0x30 HEXADECIMAL

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 print carriages 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 PRINT CARRIAGE

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
{	(0x7b}	Logo 1
|	(0x7c)	Logo 2
}	(0x7d}	Logo 3

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. IF AN ERROR WAS DETECTED IN PROCESSING A NAK WITH AN ERROR CODE IS RETURNED.

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.

NAK 1	= NOT USED
NAK 2	= Illegal Command Byte
NAK 3	= NOT USED
NAK 4	= NOT USED
NAK 5	= Trying to write a read only register

NAK 6	= Print station buffer full must print before next download to clear buffer.
NAK 7	= NOT USED
NAK 8	= NOT USED

COMMANDS:

'!' 0x21 Software Version (read only) (EV I, EV II, EV SC)

Q. ESC|STX|Address|`!`|SOH|EOT
R. ESC|STX|Address|{PRINTER fffffssss}|CR|EOT

Where:

PRINTER= ASCII string PRINTER for EVOLUTION I
EV2 for EVOLUTION II
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 I only)

The first y indicates option pack 1
The second y indicates option pack 2
The third y indicates option pack 1.5
The last y is reserved for future expansion

Where: (for EV II and EV SC)

Both units are standard with all options thus a ++++ will be returned

'#' 0x23 Printer Configuration (Read only) (EV I, EV II, 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 1
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 = option3 enabled

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

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

'l' 0x6c Special Field Flags (EV II, EV SC AND EVI 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 I, EV II, 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 print 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 print mode

‘G’ 0x47 Errors (note: error codes must be reset) (EV I, EV II, 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 I, EV II, 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

'B' 0x42 Set Unit Address (Write Only) (EV I, EV II, 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 II, EV SC AND EVI 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 I and EV II.
Each count provides a delay equal to 2 columns for EV SC.

'&' 0x26 Line Speed (RANGE 10-200) (EV I, EV II, 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 I, EV II, 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 I, EV II, 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 I, EV II, 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 II 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 II, EV SC AND EV1 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

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

'[' 0x5b DATE_ROLLOVER (EV II, EV CS AND EV1 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 Days until Expiration (max 999) (EV II, EV SC AND EVI 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' 0x52 Remaining Ink (0 to 99%) (EV I, EV II, 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 II, EV SC AND EVI 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 II, EV SC AND EVI 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

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 I, EV II, 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
ee= 0D Expiration Date Year
ee= 0E Expiration Date Julian
ee= 0F Expiration Date Day
ee= 10 Day of Week (1-7)
ee= 80 Bar Code Attribute (EV II only)

The above constitutes 10 object fields. Even though there are 48 characters 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 48-character limit.

Barcodes are also an object field and must be considered when entering a message. Thus a barcode with an imbedded sequence number is counted as two objects.

'P' 0x50 Message Objects (continued)

ff = font of object

Where: for EV I AND EV II

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

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

'$' 0x24 Line 1 Message
(EV I max 24 characters – 48 characters OP1.5, 2 or 3)
(EV II max 48 characters)
(EV SC max 96 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

'%' 0x25 Line 2 Message
(EV I max 24 characters – 48 characters OP1.5, 2 or 3)
(EV II max 48 characters)
(EV SC max 96 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

'w' 0x77 Line 3 Message (max 24 characters) (EV II 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 (max 24 characters) (EV II 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

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

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 I, EV II)

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 I, EV II)

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 II, 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 I, EV II, 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 I, EV II, and EV SC)

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

NOTE: THE FOLLOWING CODES ARE SPECIFIC TO EVOLUTION II ONLY

' " '　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.

```
// 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

```
// 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.

```
Public Sub DoMessage()
DATA$ = "800": GETINFODATA: Rem DISABLE PRINT 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 FLUSH 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**

Trend Marking Systems
POSTAL: PO Box 1311 Castle Hill NSW 2154
TEL: 61-2-96299535 FAX: 61-2-96297535
EMAIL: trend@trendmarking.com.au
www.trendmarking.com.au