Chapter 4: Escape Command Syntax

Contents of this Chapter

This chapter contains information required to implement PCL command-level parsing. Text level processing, during which the device is processing printable characters, is described in Chapter 5.

4.1 Introduction to PCL Commands

There are three types of PCL commands:

- Control Codes
- Two-Character Escape Sequences
- Parameterized Escape Sequences

Control codes are discussed in Chapter 5. This chapter defines the syntax for two-character and parameterized escape sequences.

Escape Sequences begin with the ASCII escape character *Esc* (decimal 27), followed by at least one other character. A PCL device receiving *Esc* tries to interpret subsequent characters as a PCL command rather than as data to be printed. If the device does not recognize the command, it remains at the text processing level: syntactically correct but unrecognized escape sequences are ignored; syntactically incorrect sequences are ignored and printed.

4.2 Two-Character Sequences

Two-character escape sequences take the form *EscX*, where "X" is an ASCII character between 48-126 decimal (0 through ~) that defines the operation being performed. *EscE* (Reset Printer) and *Esc9* (Clear Margins) are examples of two-character escape sequences.

4.3 Parameterized Sequences

An escape sequence is parameterized if the character following *Esc* is between 33-47 decimal (! through /). Parameterized escape sequences have the form:

Esc X y # Z_n[Data]

or, when combining escape sequences with the same parameterized and group characters:

Esc X y # z_1 # z_2 # z_3 ... # Z_n[Data]

In parameterized escape sequences, **y**, **#**, **z_i** and **[Data]** are optional.

X ***Parameterized Character*** **-** An ASCII character between 33-47 decimal (! through /) identifying the sequence as parameterized.

y ***Group Character*** **-** An ASCII character between 96-126 decimal (` through ~) identifying the type of control being performed.

***Value Field*** **-** A group of ASCII characters specifying an unsigned numeric value between 0 and 2^{32}-1, or a signed numeric value between -2^{31} and 2^{31}-1. If no value field is present, zero is assumed.

z_i ***Parameter Character*** **-** An ASCII character between 96-126 decimal (` through ~) identifying the function that interprets the previous value field. This lower-case character is used instead of Z_n (which is z_i + 32) when combining escape sequences.

Z_n ***Terminating Character*** **-** An ASCII character between 64-94 decimal (@ through ^) identifying the function that interprets the previous value field. Z_n terminates an escape sequence or a string of combined escape sequences.

[Data] ***Binary Data*** **-** Eight-bit data (e.g., graphics data, downloaded fonts, etc.) that immediately follows the terminating character. The number of data bytes is specified by the value field.

The following are examples of valid PCL commands:

Command	Description
EscE	Printer Reset (two-character escape sequence).
Esc(U	Select USASCII as primary character set (no group character or value field).
Esc(8U	Select Roman-8 as primary character set (no group character).
Esc(sB	Select medium stroke weight as primary (no value field).
Esc(s3B	Select bold stroke weight as primary.
*Esc*b5W12345*	Send 5 bytes of binary raster data.

Parameterized Character (X)

This ASCII character between 33-47 decimal (! through /) identifies the escape sequence as parameterized. It should be chosen in accordance with ANSI X3.41. Although some exceptions exist, the following parameterized characters are recommended for feature categories.

%	Personality mode change.
&	Device feature control.
(	Designates a font as primary.
)	Designates a font as secondary.
*	Graphics control.

Group Character (y)

This optional ASCII character between 96-126 decimal (` through ~) identifies the family of functions being performed. Within HP a large number of usages have been documented, and these guidelines should be followed if possible.

Value Field (#)

This is a group of characters specifying a numeric value. A value is represented as an ASCII string of decimal digits (0 through 9) specifying a numeric value between -2^{31} and $2^{31} - 1$. Zero is assigned if a value field is expected but not present. Leading zeros and blanks are ignored.

A plus "+" or minus "-" sign may start a value field; and spaces between the sign and the numeric value are stripped. Following the sign, or in its absence, a value field can be started by a decimal point "." or the decimal digits "0" through "9". Once started, a value field is closed by any character in the ranges 32-45 (SPACE through -), 47 (/), 58-63 (: through ?), or the second occurrence of a decimal point (46). Examples of value commands with signed value fields are Simple Color (*Esc*r#U*), Mechanical Print Quality (*Esc*o#Q*), and the Move CAP commands.

Value fields are assumed to be integer unless otherwise specified. The first decimal point indicates the start of a fractional portion. The number of decimal places after the decimal point is device dependent. Fractional portions of the value field are ignored if not used.

If multiple value fields are received when only one is required, the first value field should be used. Presently, no commands use more than one value field.

Commands not using the plus "+" or minus "-" sign as part of their definition should parse and use the sign to compute magnitude. Binary transfers can ignore the sign.

DEVICE NOTE: LaserJets prior to LJIII use two decimal places for decipoint commands and four decimal places for CMI and LMI commands. These were the limits within which these products could maintain accuracy when converting to their internal units of measure.

DEVICE NOTE: DeskJets below DJ1200C use two decimal places for all fractional values.

The following are examples of value fields and the resulting data passed to the command. The brackets are used to distinguish leading or trailing spaces in the examples.

Value Field	**Resulting Data**
{9}	9 — no spaces, signs, or fractional value.
{ 009 }	9 — leading 0's and spaces and trailing spaces are stripped.
{+ 007}	Positive signed 7 — spaces after the sign are stripped.
{-7}	Negative signed 7 — no spaces or zeros after the negative sign.
{} or { }	0 — since the value field is blank or no value field is present.
{42187}	32767 — magnitude is limited to 32767 (with noted exceptions)
{4./25}	4 — The "/" terminates the value field.
{4.75}	4.75
{4.75}	4 if a decimal portion is not expected.

Parameter Character (z_i)

This ASCII character between 96-126 decimal (` through ~) is used to combine escape sequences having the same parameterized and group characters. After executing the function identified by z_i, the parser interprets succeeding data as a value field. Z_n, the upshifted (z_i - 32) equivalent, signifies escape sequence termination when used instead of z_i.

Terminating Character (Z_n)

This ASCII character between 64-94 decimal (@ through ^) signifies escape sequence termination when it is used instead of z_i, the downshifted (Z_n + 32) equivalent.

The terminating character Z_n appears in two forms:

1. If at least one parameter is specified, the last parameter character in an escape sequence or string of combined escape sequences is upshifted from its column 6 or 7 position in the ASCII table to the equivalent position in column 4 or 5. That is, z_i becomes the terminating character z_i - 32. For example, *Esc&l6d...* and *Esc&l6D* both set line spacing to 6 lines per inch.
2. If no parameters (and therefore no values) are specified, the group character is the terminator. That is, y becomes Z_n. For example, *Esc(@* is a legal escape sequence with no group character.

The parameter and terminating characters have the same function; but the lower-case parameter character (Z_n + 32) allows escape sequences from the same group to be combined. When the parser encounters z_i, it expects a value field. On the other hand, Z_n places a device back at the text level of processing.

Binary Data Transfers [Data]

Some parameterized escape sequences transfer binary data (graphics, transparent print data, font downloading, etc.). These sequences contain a data field immediately following the terminating or parameter character. The data field contains 8-bit binary data that is not interpreted by escape sequence parsing. The value field of the escape sequence must specify the correct number of data bytes. Extra bytes are ignored by the command and may be considered printable data. Insufficient bytes may cause the device to consider characters following the command as part of the data field.

To ensure that products not supporting one or more of these commands can interpret this data in a non-disruptive fashion, the following guidelines should be used for binary transfer escape sequences:

- The format is *EscXy#w/W [binary data]*. The binary data string contains "#" bytes.
- The value field must describe the number of bytes to be transferred.
- Binary transfer commands should use lowercase "w" as the parameter character or uppercase "W" as the terminating character.
- The "w" parameter character or "W" terminating character should be followed immediately by the binary data.
- Unsupported binary escape sequences denoted by the "w" or "W" terminating character should be handled as an unrecognized command and the binary data ignored.
- When illegal or unsupported escape sequences are discarded, the binary data associated with the command must also be discarded.

DEVICE NOTE: LaserJets do not currently support the lowercase "w" as indicating that binary data follows and will continue parsing the binary data, looking for value field or terminating character.

NOTE: Not all binary transfer escape sequences are defined by the "w" parameter character or "W" terminating character. However, this is the preferred parameter for standardization. The following are some sequences that do not end with a "w" or "W"; these commands should be parsed and any binary data discarded if the device does not support the command.

Esc & p # x/X [Transparent data]	Transparent print data
*Esc * b # v/V [Raster data]*	Raster graphics transfer by plane

NOTE: The following commands should not be interpreted as binary data commands:

Esc & k # W	Device Specific Control
Esc (W	Select Primary Font Character Set
Esc) W	Select Secondary Font Character Set
Esc & d W	Underline

4.4 Rules for PCL Commands

PCL Commands as Modes

Most PCL commands are defined as modes: once a feature has been selected, it remains in effect until a device reset. However, other commands may interact and change the feature.

Order of Execution

Functions should be executed as they are decoded. For example, changing line spacing from 6 to 8 lpi and then setting text length to 80 lines has a completely different meaning than setting text length to 80 lines and then changing line spacing from 6 to 8 lpi. Another example might involve page length and top margin: the top margin will move if set after page length.

Unrecognized Sequences

A syntactically correct escape sequence that is not recognized is ignored.

An escape sequence that cannot be fully implemented should not necessarily be ignored, since **some** action may be more appropriate than no action. For example, attempting to set the right margin past the physical limits of the printer should cause the right margin to be set at the printer's physical limit.

The Simplex/Duplex command (*Esc&l#S*) is an example of a command that is not completely ignored by some non-duplex printers; it causes a conditional page eject in LaserJets and DeskJets above 1000.

Illegal Syntax

If an illegal character is received within an escape sequence, processing of the sequence terminates. For data preservation, the illegal character should be interpreted at the text level independently of previous characters.

4.5 Combining Commands

Parameterized (**but not two-character**) escape sequences can be combined, eliminating intermediate escape characters. To combine escape sequences, use the parameter character zi, instead of its upper-case equivalent Z_n (z_i - 32), for every escape sequence except the last.

Use the following three rules to combine parameterized sequences into one string:

1. The first two characters after *Esc* (the parameterized and group character) must be the same for all the commands being combined. In the example below, these are "*&*" and "*l*".
2. All alphabetic characters within the combined command must be lower-case except the final letter, which is upper-case to notify the device when the command is complete. In the example below, the terminating character "*E*" in the first sequence is changed to "*e*" in the combined sequence.
3. Combined commands are executed as they are parsed (from left to right). That is, if several commands are concatenated and the last command has a syntax error, all the commands up to the last one are executed.

EXAMPLE: To set the top margin to line 10 and the text length to 70 lines, either of the following could be sent:

Esc&l10E and *Esc&l70F*
or
Esc&l10e70F

4.6 Backus-Naur Form (BNF)

PCL escape sequence syntax can be described by a rule system derived from Backus-Naur Form (BNF). The following symbols are used:

bold	indicates syntactic categories
::=	"to be rewritten as"
|	separates choices
| ... |	specifies the range of choices
XXH	character code in hex
$\{\}_1$	choose 1 of the enclosed items
$\{\}_{0+}$	the enclosed items occur 0 or more times
$\{\}_{1+}$	the enclosed items occur 1 or more times
$\sum$	Start symbol

Other items are terminal symbols of the language.

BNF Rewriting Rules (Productions)

$\sum$ ::= **two character sequence** | **parameterized_form1** | **parameterized_form2**
esc ::= 1BH
space ::= 20H
right_brace ::= 7BH
w ::= 88H
data ::= 0 ... 255
two_character_sequence ::= **esc** 0 | 1 | 2 | **...** | **right_brace** | ~
parameterized_character ::= ! | " | # | **...** | - | . | /
X ::= **parameterized character**
group_character ::= ' | a | b | **...** | **right_brace** | ~
y ::= **group_character**
parameter character ::= ' | a | b | **...** | **right_brace** | ~
$\mathbf{z_i}$ ::= **parameter_character**
terminating_character ::= @ | A | B | **...** |] | ^
$\mathbf{Z_n}$ ::= **terminating_character**
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
integer ::= $\mathbf{\{ digit \}_{1+}}$
fraction ::= $\mathbf{. \{ digit \}_{0+}}$
float ::= **integer fraction**
number ::= **integer** | **fraction** | **float**
unsigned_value ::= $\mathbf{\{ space \}_{0+}}$ **number**
signed_value ::= $\mathbf{\{ space \}_{0+} \{ + | - \}_1 \{ space \}_{0+}}$ **unsigned_value**
value_field ::= $\mathbf{\{ unsigned\_value | signed\_value \}_{0+}}$
::= **value_field**
parameterized_form1 ::= $\mathbf{esc\ X\ y \{ \#\ z_i \}_{0+}\ \#\ Z_n}$
parameterized_form2 ::= $\mathbf{esc\ X\ \#\ Z_n}$
parameterized_form3 ::= $\mathbf{esc\ X\ y \{ \#\ w \{ data \}_{0+} \}_{0+}\ \#\ W \{ data \}_{0+}}$

4.7 Converting a Floating Point Number

Some of the fields within the structure of a binary transfer may be in IEEE floating point. The following formula converts a floating point number into an IEEE 754-1985 4-byte encoded floating point number.

Variable	Meaning
X	Floating point number in base 10
Sign	X's sign (either 1 or -1)
S	Sign bit (1 if Sign is -1, 0 if Sign is 1)
E	X's biased base 2 exponent in base 10
F	X's base 2 fraction in base 10

Formula to convert IEEE 754-1985 number into native machine format

```
If E = 0 And F = 0 Then
    X = 0
Else
    X = Sign * ( 1 + ( F / 2^23 ) ) * 2^( E - 127 )
End If
```

Formula to convert native machine number into IEEE 754-1985 format

```
If X = 0 Then
    S = 0
    E = 0
    F = 0
Else
    Sign = X / ABS(X)
    S = ( 1 - Sign ) / 2
    E = INT( LOG( ABS(X) ) / LOG(2) ) + 127
    F = INT ( ( ( X / 2^(E-127) ) - 1 ) * 2^23 )
End If
```

Example:

X = 201.187
Sign = 201.187 / ABS(201.187)
Sign = 201.187 / 201.187
Sign = 1
S = (1 - Sign) / 2
S = (1 - 1) / 2
S = 0
E = INT(LOG(ABS(201.187)) / LOG(2)) + 127
E = INT(LOG(201.187) / LOG(2)) + 127
E = INT(2.303599914701 / .301029995664) + 127
E = INT(7.652393276024) + 127
E = 7 + 127
E = 134 in base 10 representation, which is 10000110 in base 2
F = INT (((201.187 / 2^(134-127)) - 1) * 2^23)
F = INT (((201.187 / 2^7) - 1) * 2^23)
F = INT ((1.5717734375 - 1) * 2^23)
F = INT (0.5717734375 * 2^23)
F = INT (0.5717734375 * 8388608)
F = INT (4796383.232)
F = 4796383 in base 10 representation, which is
10010010010111111011111 in base 2

To check:

X = Sign * (1 + (F / 2^23)) * 2^(E - 127)
X = 1 * (1 + (4796383 / 2^23)) * 2^(134 - 127)
X = (1 + 0.5717734375) * 2^7
X = 1.5717734375 * 128
X = 201.18699646