

PCL XL Feature Reference

Protocol Class 2.0

Revision: p1.8
Revision Date: January 6, 1998
Author(s):
Word for Windows File: xl_ref20r18.doc
Word for Windows Version: Word97
FIGURE FILES: VISIO 4.0

Document Revision History

Rev	Revision Description	Date	Approval
1.0	First release of frozen features for Class 1.1	13Dec95	
1.1	Fixed many typo's in spec revision 1.0	15May96	
p1.2	First draft for Class 2.0 features	16May96	
p1.3	Second draft of Class 2.0 features	06Sep96	
P1.4	Final draft of Class 2.0 features	09Feb97	
P1.5	Redefinition of certain 2.0 features	01Mar97	
P1.6	Added newest Paper Handling information	03June1997	
P1.7	Additional 2.0 Features and General Editing	14July1997	
P1.8	Added JetASM format operator examples	06Jan1998	

Note: This document is for HP Personnel only. Please do not copy and distribute without notification to the PCL XL team.

***** NOTICE *****

HEWLETT-PACKARD COMPANY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 1	BLD 4600
-----------	-----------------	---------	--------	----------

Hewlett-Packard Company Confidential

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE OR TECHNICAL INFORMATION. HEWLETT-PACKARD COMPANY DOES NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE OR TECHNICAL INFORMATION IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. YOU ASSUME THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE OR TECHNICAL INFORMATION. The exclusion of implied warranties is not permitted by some jurisdictions. The above exclusion may not apply to you.

IN NO EVENT WILL HEWLETT-PACKARD COMPANY BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE OR TECHNICAL INFORMATION EVEN IF HEWLETT-PACKARD HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Because some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. Hewlett-Packard liability to you for actual damages from any cause whatsoever, and regardless of the form of the action (whether in contract, tort including negligence, product liability or otherwise), will be limited to US \$50.

Copyright © 1995 - 1997 Hewlett-Packard Company. All rights reserved.

Table Of Contents

Figure Files: Visio 4.0 1

1.0 INTRODUCTION7

1.1 GLOSSARY8

1.2 IMAGING MODEL9

1.3 STREAMS10

1.4 SESSIONS12

1.5 PAGES13

1.6 USER COORDINATE SYSTEM14

1.7 COORDINATE TRANSFORMATION MATRIX15

1.8 PATHS16

1.9 CLIP PATHS17

1.10 BITMAPS18

1.11 PENS19

1.12 BRUSHES20

1.13 FONTS21

1.14 PAINT SOURCES22

 1.14.1 Color Objects22

 1.14.2 Raster Patterns22

1.15 RASTER OPERATIONS22

1.16 GRAPHICS STATE23

2.0 PCL XL OPERATORS, ATTRIBUTES AND STREAMS24

2.1 PROTOCOL OPERATORS24

2.2 SIMPLE OPERATOR EXAMPLES26

2.3 LEGAL PROTOCOL OPERATOR SEQUENCES27

2.4 ATTRIBUTE LIST SPECIFICATION FORMAT29

2.5 FORMAT OF OPERATOR SPECIFICATIONS30

2.6 PCL XL PROTOCOL CLASS AND REVISION NUMBERS31

2.7 ERROR AND WARNING REPORTS32

2.8 BINARY STREAM FORMAT33

2.8.1 BINARY STREAM FORMAT DOCUMENT CONVENTIONS33

2.8.2 BINARY STREAM HEADERS34

2.8.3 BINARY OPERATOR AND ATTRIBUTE LIST CONSTRUCTION35

2.8.4 BINARY STREAM OPERATOR SYNTAX38

2.8.5 BINARY STREAM OPERATOR EXAMPLE39

 Operator: *DemoOperator*39

2.9 PCL XL DESIGN PRINCIPLES41

3.0 SESSION OPERATORS42

3.1 BEGINNING AND ENDING A SESSION42

 Operator: *BeginSession*43

 Operator: *EndSession*45

3.2 BEGINNING AND ENDING A PAGE46

 Operator: *BeginPage*47

 Operator: *EndPage*54

3.3 ADDING A COMMENT56

 Operator: *Comment*57

3.4 OPENING AND CLOSING A DATA SOURCE58

 Operator: *OpenDataSource*59

 Operator: *CloseDataSource*61

4.0	FONT CONTROL OPERATORS	62
4.1	DEFINING FONTS AND CHARACTERS.....	62
	Operator: <i>BeginFontHeader</i>	63
	Operator: <i>ReadFontHeader</i>	64
	Operator: <i>EndFontHeader</i>	66
	Operator: <i>BeginChar</i>	67
	Operator: <i>ReadChar</i>	68
	Operator: <i>EndChar</i>	70
4.2	REMOVING FONTS	71
	Operator: <i>RemoveFont</i>	72
5.0	GRAPHICS STATE OPERATORS	73
5.1	SAVING AND RESTORING THE GRAPHICS STATE	75
	Operator: <i>PopGS</i>	77
	Operator: <i>PushGS</i>	78
5.2	SETTING AND CHANGING THE CURSOR LOCATION.....	79
	Operator: <i>SetCursor</i>	80
	Operator: <i>SetCursorRel</i>	81
5.3	SETTING COLOR SPACE AND PAINT SOURCES.....	82
	Operator: <i>SetColorSpace</i>	83
	Operator: <i>SetBrushSource</i>	86
	Operator: <i>SetPenSource</i>	89
5.4	SETTING FONT AND CHARACTER ATTRIBUTES	92
	Operator: <i>SetCharAngle</i>	93
	Operator: <i>SetCharScale</i>	94
	Operator: <i>SetCharShear</i>	95
	Operator: <i>SetCharBoldValue</i>	96
	Operator: <i>SetCharSubMode</i>	98
	Operator: <i>SetCharAttributes</i>	99
	Operator: <i>SetFont</i>	100
5.5	SETTING ATTRIBUTES OF THE CURRENT PATH.....	102
	Operator: <i>SetFillMode</i>	103
	Operator: <i>SetMiterLimit</i>	105
	Operator: <i>SetLineCap</i>	107
	Operator: <i>SetLineJoin</i>	108
	Operator: <i>SetLineDash</i>	109
	Operator: <i>SetPenWidth</i>	111
5.6	SETTING THE CURRENT CLIP PATH.....	112
	Operator: <i>SetClipReplace</i>	113
	Operator: <i>SetClipIntersect</i>	114
	Operator: <i>SetClipRectangle</i>	115
	Operator: <i>SetClipToPage</i>	116
	Operator: <i>SetPathToClip</i>	117
	Operator: <i>SetClipMode</i>	118
5.7	PCL XL LOGICAL OPERATIONS	119
5.7.3	THE ROP3 OPERANDS.....	122
5.7.4	TRANSPARENCY MODES	125
5.7.5	RASTER OPERATIONS AND TRANSPARENCY INTERACTIONS	127
5.7.6	ROP2 EQUIVALENTS IN THE ROP3 SET.....	129
	Operator: <i>SetPaintTxMode</i>	130
	Operator: <i>SetSourceTxMode</i>	131
	Operator: <i>SetROP</i>	132

5.8 Setting and Using Halftone Methods.....133
 Operator: *SetHalftoneMethod*.....135

5.9 SETTING PAGE COORDINATE SYSTEM ATTRIBUTES137
 Operator: *SetPageDefaultCTM*.....138
 Operator: *SetPageOrigin*.....139
 Operator: *SetPageRotation*.....140
 Operator: *SetPageScale*.....141

6.0 PAINTING OPERATORS143

6.1 PATH MANIPULATION OPERATORS143
 Operator: *CloseSubPath*.....144
 Operator: *NewPath*.....145
 Operator: *PaintPath*146

6.2 DEFINING AND PAINTING ARCS148
 Operator: *ArcPath*149

6.3 DEFINING AND PAINTING BEZIEFS150
 Operator: *BezierPath*.....151
 Operator: *BezierRelPath*.....153

6.4 DEFINING AND PAINTING A CHORD.....155
 Operator: *Chord*156
 Operator: *ChordPath*.....157

6.5 DEFINING AND PAINTING AN ELLIPSE158
 Operator: *Ellipse*.....159
 Operator: *EllipsePath*.....160

6.6 DEFINING AND PAINTING LINES161
 Operator: *LinePath*.....162
 Operator: *LineRelPath*.....164

6.7 DEFINING AND PAINTING PIES166
 Operator: *Pie*.....167
 Operator: *PiePath*.....168

6.8 DEFINING AND PAINTING RECTANGLES AND ROUND RECTANGLES169
 Operator: *Rectangle*.....170
 Operator: *RectanglePath*.....171
 Operator: *RoundRectangle*172
 Operator: *RoundRectanglePath*.....173

6.10 DEFINING AND PAINTING BITMAP IMAGES174
 Operator: *BeginImage*175
 Operator: *ReadImage*177
 Operator: *EndImage*179

6.11 DEFINING AND PAINTING RASTER PATTERNS.....180
 Operator: *BeginRastPattern*182
 Operator: *ReadRastPattern*185
 Operator: *EndRastPattern*187

6.12 DEFINING AND PAINTING SCAN LINES.....188
 Operator: *BeginScan*.....189
 Operator: *ScanLineRel*190
 Operator: *EndScan*191

6.13 PAINTING TEXT.....192
 Operator: *Text*.....193
 Operator: *TextPath*195

7.0 USER-DEFINED STREAMS.....197

7.1 DEFINING AND USING USER-DEFINED STREAMS197

7.2 STREAM HEADER FORMAT.....198

 Operator: *BeginStream*.....200

 Operator: *ReadStream*.....201

 Operator: *EndStream*.....202

 Operator: *ExecStream*.....203

 Operator: *RemoveStream*.....204

APPENDIX.....205

A. PCL XL AND PIXEL PLACEMENT205

B. BINARY STREAM TAG VALUES.....207

C. OPERATOR NAME TO TAG VALUE TABLE.....212

D. DATA TYPE TO TAG VALUE TABLE213

E. ATTRIBUTE ID NUMBER TO ATTRIBUTE NAME TABLE.....214

F. ATTRIBUTE NAME TO DATA TYPES TABLE.....216

G. ATTRIBUTE VALUE ENUMERATIONS TABLE220

H. HP LASERJET PRINTER PJL AND PCL XL STREAMS222

I. HP LASERJET PRINTER ENGINE/PAPER HANDLING FEATURES AND PCL XL224

J. PCL XL DEVICE ERROR REPORTING227

K. PCL XL ERROR AND WARNING CODES229

Generic Operator Errors.....229

Operator-Specific Errors231

L. PCL XL FONT FORMATS AND FONT DEFINITION OPERATORS.....234

M. TABLE OF ROP3 OPERATIONS.....244

N. MODIFIED BACKUS-NAUR FORM.....246

O. SYMBOL SET SELECTION247

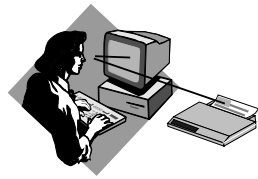
P. HP LASERJET PRINTER INTERNAL FONT SELECTION252

Q. COMPRESSION METHODS254

INDEX256

Overview**Introduction**

1.0 Introduction



Devices supporting PCL XL produce text and graphics pages. The vehicle by which a page imaging task is submitted to a PCL XL device is through the *PCL XL Imaging Protocol*.

The PCL XL imaging protocol is a set of imaging operators for communicating electronic images from one device to another. The imaging protocol may be communicated among devices via traditional byte streams or other communication mechanisms such as remote procedure calls. Devices that use PCL XL for imaging pages may include printers, fax machines, copiers, software on-screen viewers, etc.

This is an introduction to PCL XL imaging protocol concepts. All PCL XL devices conform to the general concepts outlined in this section.

Readers of this document are assumed to have basic knowledge in areas of imaging, firmware, software and page description languages.

1.1 Glossary



The following is a table of definitions to help understand the terminology used in this section.

Contone	A multi-bit per pixel raster definition commonly used to achieve device independence.
DPI	Dots Per Inch – A common measurement of a device’s resolution.
Graphic State	A firmware repository that groups many imaging states into a single container.
Imaging Operators	Device operators that mark the page (i.e. drawing a line) or effect the nature of marking (i.e. setting color, setting line width, etc.).
Intelligent Printer	An intelligent printer accepts page descriptions in the form of lines, character codes, etc. This form is called “symbolic” form. Each symbol is rendered into bits at device-resolution. Intelligent printers perform this rendering inside the printer. Intelligent printers perform very well for mainstream documents in high page-per-minute, color, and shared environments.
Painting	The act of activating or placing dots on an output surface.
Protocol Class	A method of identifying a specific version of PCL XL, usually specifying the imaging and page handling characteristics.
Raster Graphics	Symbols destined for the pages that are pixel-oriented, such as scanned images, art from the Windows Paintbrush package, etc.
Vector Graphics	Symbols destined for the pages that are line-oriented, such as straight lines, rectangles, circles, etc.

Overview**Introduction**

1.2 *Imaging Model*



The foundation of any text and graphics imaging device is its *imaging model*. An imaging model defines a class of operators with which graphical objects are painted on a page or display. The model also defines each tool's attributes and capabilities.

Graphical User Interface environments such as Microsoft-Windows, OS/2, X-Windows (Motif), and Macintosh each have well-defined imaging models embodied in the capabilities of their individual graphics device interfaces. The speed and accuracy with which an application or driver in these environments may image an arbitrary page is often dependent on the existence of matching imaging capabilities in the target imaging device.

PCL XL imaging operations are based on an imaging model that matches today's graphical user interface (GUI) environments and is built to be extensible for the future. This section provides an introduction to the PCL XL imaging model and related key concepts.

1.3 Streams

Stream Header

Stream Body

Certain devices such as printers and fax machines accept a sequence of bytes over a network, parallel or serial port to describe a unit of imaging work to be performed. PCL XL imaging is accomplished on byte-oriented devices through an encapsulated sequence of operations called a *stream*. A PCL XL stream is a self-describing object that contains a sequence of bytes that may instruct PCL XL to image anything from a simple graphical object (e.g. a string of text, an ellipse, or a logo) to entire documents. For example, a stream may instruct a PCL XL device to draw one or more of the following:

- ◆ A piece of clip-art
- ◆ A form
- ◆ A watermark
- ◆ A signature
- ◆ An entire page or entire document
- ◆ A series of Documents

The overall set of text, vector, and raster imaging operators available to PCL XL streams provide the fastest and most efficient methods of imaging pages in a given Hewlett-Packard printing or viewing device. The format of streams will be described later in this document.

All streams belong to a specific protocol class. The protocol class specifies the operations and capability of the operators in the stream and the set of operators that are allowed during a session inside that stream. The protocol class and revision number for the classes are defined in the stream header, preceding the stream body containing PCL XL operators and data.

Each protocol class is defined according to capabilities required for specific device types (i.e. raster-only printers, intelligent printers, fax machines, etc.). For example, there may be a protocol class for low-cost (i.e. raster-only) printers that contains fewer capabilities and is less expensive to implement than a protocol class for a high-volume printing devices with collators and staplers.

Each protocol class places boundaries on the imaging operations and behavior of the device during the session. Some PCL XL devices may be designed to accept multiple protocol classes. For example, one could envision a high-volume printer accepting a redirected stream from a fax machine or a stream produced by a scanner.

In a PCL XL device where the operator usage is via programming language APIs or remote procedure calls, the protocol class is implied by the set of PCL XL function calls available to the user.

Overview

Introduction

For more information regarding PCL XL operator streams, see the section entitled “**User-Defined Streams.**”

1.4 Sessions



A *session* is the context in which a user interacts with a PCL XL device. The user instructs PCL XL to begin a new session to start imaging on the device. The user may only cause imaging to occur on a PCL XL device during an active session.

Figure 1-1 shows the relationship between sessions and page descriptions. The user may only describe pages between the time a session begins and ends. The concept of a PCL XL session may differ from the traditional concept of a job. A job is a high-level printing environment concept usually intertwined with spooling systems. A session is a PCL XL-specific environment the user sets up to image a collection of pages. The user may wish to alter the session environment during a traditional job. A stream within a job may contain more than one session for certain classes of devices.

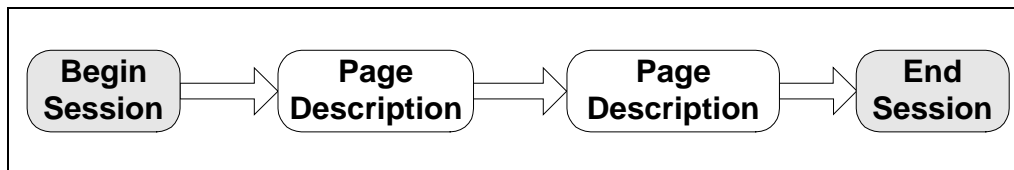


Figure 1-1. Relationship Between Sessions and Page Operations.

User interactions during a session are typically from the user to the device (i.e. all graphics imaging operations). Interactions may also be initiated from the device to the user for error reporting.

The operator to start a session has both *required* and *optional* attributes. PCL XL devices require a user resolution attribute to be set when a new session is started. This attribute identifies the coordinate system units in which the user prefers to describe pages during the session (e.g. 600 units-per-inch). The user units resolution for a session may be different from the internal device resolution. This is because the coordinate system of the session is device-independent. See the *Coordinate System* and *CTM* sections below.

Overview	Introduction
-----------------	---------------------


1.5 Pages

Memo to HP: Your Products Are Great! Sincerely, Your Customers
--

A *page* in a PCL XL device is a surface upon which painting operations may be performed. This surface is an abstraction of the physical media (printed page or display) upon which objects are painted. The user must begin a new page before any painting operations may begin. Page imaging operations may paint text, vector, and raster objects on a physical page.

Pages have attributes that define and effect the real page received or viewed by a person. For example, the size, type, source, and destination for the physical page media are attributes that may be associated with a page.

1.6 User Coordinate System

 PCL XL devices have a two-dimensional user coordinate system to specify the location at which graphical objects are placed and painted.

The user coordinate system defaults are illustrated in Figure 1-3. These defaults are specified as follows:

- ◆ The origin ($x = 0, y = 0$) is the “physical” upper left hand corner of page. Note: PCL 5e used the upper left corner of the printable page.
- ◆ The x coordinate increases horizontally from left to right
- ◆ The y coordinate increases vertically from top to bottom
- ◆ The default scale of the x and y axis is set according to the session resolution attribute given by the user when the session begins

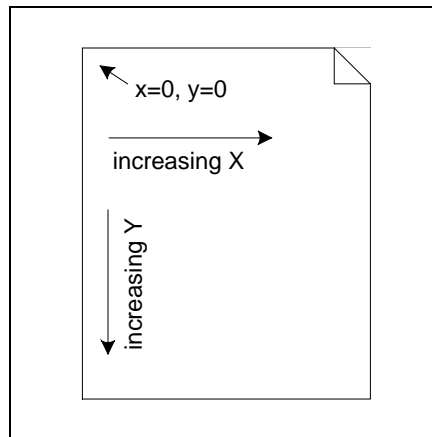
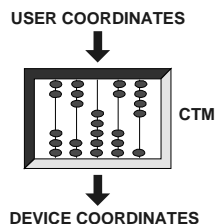


Figure 1-3. User Coordinate System Defaults for a Page

1.7 Coordinate Transformation Matrix



The component used to map user coordinate space to the page surface coordinate space is called a coordinate transformation matrix (CTM). The user may manipulate values in the CTM to transform the way in which text, vector objects, and raster bits are painted on the page surface.

Initially, the CTM is set such that the origin of a page is the top left corner. Figure 1-4 illustrates a case where the user has changed the CTM to translate the user origin to 4 inches to the right of the original origin and 5 inches down from the original origin. On the right of Figure 1-4, clip-art now painted at user coordinates of $x = 0, y = 0$ will be printed on the physical page at 4 inches to the right and 5 inches down from the *original* origin.

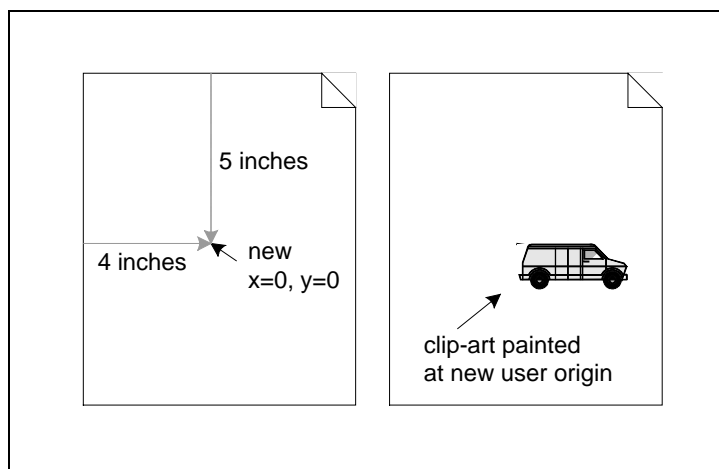


Figure 1-4. Changing the User Coordinate System by Modifying the CTM.

Setting the session user resolution attribute also uses the CTM to accomplish device-independence. For example, the user may prefer to work in 72 units-per-inch coordinates on a device that happens to be 600 pixels-per-inch internally. The user may accomplish this by setting the user resolution attribute to 72 units-per-inch for the session. As in this and every case the CTM takes care of translating all the user coordinates (e.g. 72 units-per-inch) to the internal device coordinates (e.g. 600 units-per-inch). The user may work during the entire session in 72 units-per-inch space and automatically achieve the device-best resolution on the page for lines edges, text, and raster objects.

1.8 Paths



Nearly every application has the need to print or display vector regions that are comprised of one or more arbitrary lines and/or curves. Vector regions are described using *paths*. Paths are used to define both arbitrary vector regions and common regions like circles and rectangles.

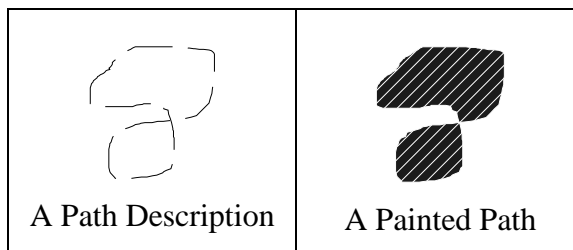


Figure 1-5. A Path Description (left) and a Painted Path (right).

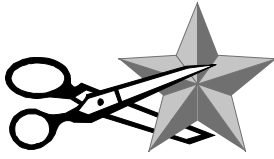
The dashed-line symbol on the left in Figure 1-5 depicts a path description of a vector region. Through the PCL XL imaging protocol, the user describes a series of lines, arcs, and/or Beziers to form a path. Paths are not visible on a physical page or display unless they are painted.

The symbol on the right in Figure 1-5 depicts a painted path where a brush was defined to fill the path with a pattern. When a path is painted, the device takes the line and curve descriptions of the path and fills inside the path with user-selected paint and/or strokes paint along the center of the edges of the path.

Filling and outlining is controlled by pen and brush settings in the current graphics state (pens, brushes and the graphics state are introduced in later sections). The lines, arcs, and Beziers in the path may be connected end-to-end or they may be left disconnected.

Connected sets of lines and curves form a *sub-path*. A path contains zero or more sub-paths. Paths provide a very efficient way to accurately describe any vector region on a page.

1.9 Clip Paths



In addition to drawing, a path is used to define the current clip path. Clip paths allow the application or driver to constrain areas in which marks may be placed on a page for vector, text, and raster objects. All painting operations are confined to the inside region(s) defined by the current clip path.

Figure 1-6 depicts how defining a clip path effects painting operations. In this case, a rectangular clipping region is defined by a path. After the clip path is defined, a star object is painted to the page near the origin. The only elements of the star actually painted on the page are those lying within the clip path. The clip path may be formed from any combination of lines and curves. Flexible clip path definition allows the driver or application to efficiently image complex graphics without using expensive bit-mapped operations like ROP's or scanlines.

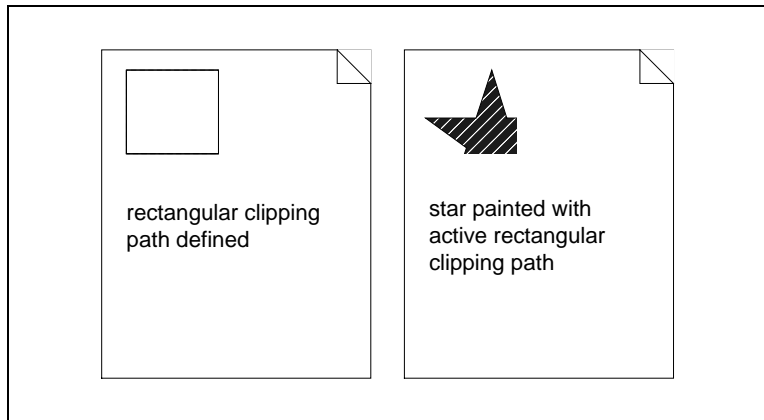


Figure 1-6. Effect of a Clip Path on Painting Operations.

Overview	Introduction
-----------------	---------------------

1.10 Bitmaps



Bitmaps are rectangular raster regions and include scanned images. Bitmaps may be single- or multi-bit-per-pixel, including color and contone formats and are device independent with respect to resolution. Bitmaps may be scaled and half-toned in a PCL XL device.

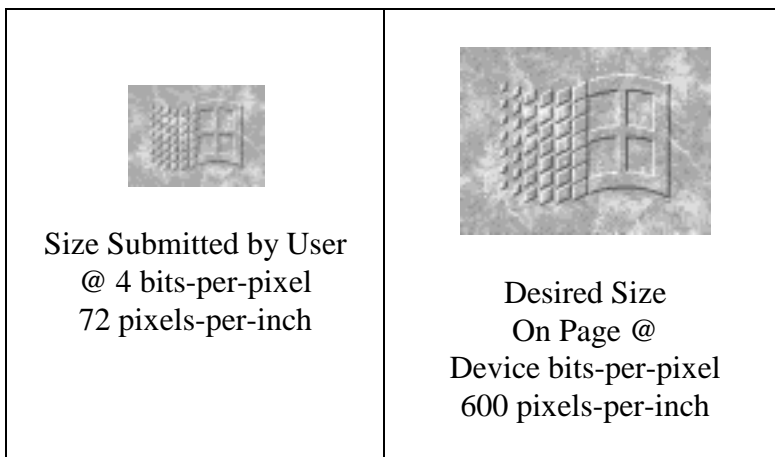
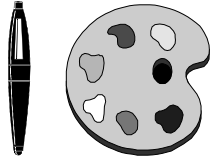


Figure 1-7. Device Half-toning and Scaling of Raster (Scanned) Images

Figure 1-7 depicts an example where a 72 pixels-per-inch image is submitted to a 600 DPI PCL XL device. Assume the user also wants the image to be enlarged to fit the desired area on the physical page. On a monochrome device, two actions must transpire: (1) the image must be scaled-up to fit a larger area on the page and (2) the image must be halftoned to simulate continuous levels of gray on the device. On color devices, the halftoning step may or may not be skipped. Image scaling and halftoning in a device allows bitmap images that are different size and resolution in their original form to be scaled to the appropriate size with good visual quality. Scaling and halftoning bitmaps in a device assures device independence and often leads to file size and transmission time efficiencies.

Overview	Introduction
-----------------	---------------------

1.11 Pens



A pen may be defined for stroking paint along a path object to achieve an outlining effect. When a paint operation is invoked for paths, the edges are stroked with the paint source associated with the current pen (see the Paint Sources section).

Figure 1-8 depicts the results of painting a path object with the pen set to a moderately wide line width and a black paint source.

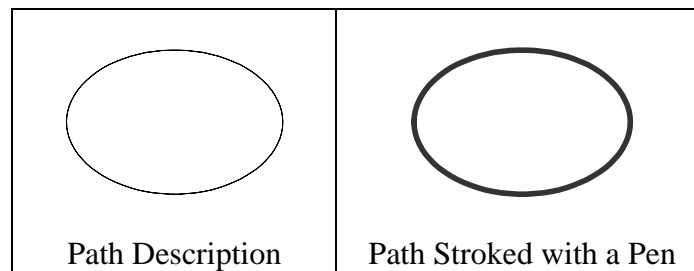
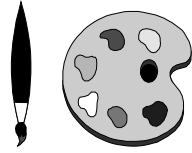


Figure 1-8. Outlining an Object with the Current Pen

- ◆ The following is a summary of the key concepts behind pens:
- ◆ Pens can outline any path object, including text outlines and paths
- ◆ There is one pen defined to outline both vector and text objects
- ◆ The color or pattern stroked along the edges of the object depends upon the *paint source* currently associated with the pen (see section 1.14 on Paint Sources for more detail)
- ◆ If the pen is not associated with a paint source when an object is painted, no outlining is performed
- ◆ The current path is *not* destroyed when outlined, allowing the path be reused for later filling or clipping operations

Overview
Introduction

1.12 Brushes



Brushes may be defined for filling paint within the edges of path objects or text. Brushes are also used to color raster images. When a paint operation is invoked on paths or text, the region inside the edges of the object are brushed with the paint source associated with the active brush (see section 1.14 on Paint Sources for more detail).

Figure 1-9 depicts the results of filling a path object with the brush set to a diagonal pattern paint source. Note that in this example no pen was defined to stroke the edges of the object.

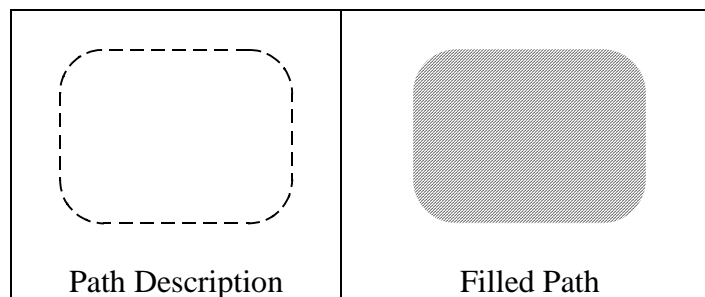


Figure 1-9. Filling Inside an Object's Edges with the Current Brush

The following is a summary of the key concepts behind brushes:

- ◆ There is one brush defined to fill both vector, text, and color raster objects
- ◆ The color or pattern filled or colored within the object depends upon the *paint source* currently associated with the brush
- ◆ If the brush is not associated with a paint source when an object is painted, no filling is performed
- ◆ The current path is *not* destroyed when filled, allowing the path to be reused for later stroking operations

Overview	Introduction
-----------------	---------------------

1.13 Fonts

ABC	<p>A PCL XL device allows the user to place characters of a font anywhere on the page. The actual font technology available in the device (TrueType, Bitmap, etc.) to render each character is implementation-dependent. Each character is treated as an independent graphical object. Each character is placed at the current cursor location prior to painting. The current cursor location is defined or changed by graphics state operations, path construction operations, and/or intermediate text placement operations.</p> <p>PCL XL imaging protocol allows one or more characters to be placed on the page by a single text operator. The first character is placed at the current cursor. The remaining characters in a multi-character operation are placed at corresponding escapements (character spacings) provided by a parameter to the text operator. Each escapement for a character tells PCL XL where the current cursor should be relocated for placement of the next character in succession.</p> <p>Character sizes are always specified in user units. Painted characters are transformed, scaled and rotated according to page coordinate transformation matrix (CTM) manipulations by the user. Characters may also be rotated, scaled, and sheared (skewed) in an additive manner to the current page CTM by setting a separate character CTM. Painting direction of characters may also be specified for international fonts. Both 8- and 16-bit character codes are supported.</p>
------------	--

Overview	Introduction
-----------------	---------------------

1.14 Paint Sources



A paint source defines the color or graphical pattern used to paint a vector or text object. A paint source associated with a pen is the source of a color object or pattern for a stroking operation. A paint source associated with a brush is the source of a color object or pattern for a filling or raster-coloring operation.

1.14.1 COLOR OBJECTS



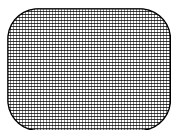
Color Object

Color objects are conceptually a specific color in a designated color space (i.e. RGB, CIELAB, Gray, etc.) defined by one or more color components. In a PCL XL device, color objects are single values or an ordered set of values that map to a specific color in the current color space. These values may be associated with a pen or brush to define the color with which a graphical object is painted.

The components of a color object must be compatible with the current (active) color space. For example, a color object intended for use in an RGB color space must contain three ordered components, each representing a red, green, and blue intensity value. In an RGB color space, a color object with the ordered values 0, 0.9, 0 would produce green when used as a paint source. Color objects may also represent gray-scaled and mono-toned colors.

A user-defined mapping into a color space may also be constructed using palettes. The method and use of palettes will be discussed later in this document.

1.14.2 RASTER PATTERNS



Raster patterns are rectangular NxM source pixel regions used for constructing primitive patterns. The pattern may be specified as a device-independent bitmap. Once created, raster patterns may be associated with a pen or a brush. A raster pattern associated with the brush is tiled across the page and then used to fill graphical objects. A raster pattern associated with a pen is tiled across the page and then used to outline or stroke along the edges of a path object.

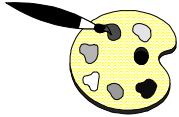
1.15 Raster Operations

10000001
OR 10101110
= 10101111

Some protocol classes allow raster operations (ROP's). ROP's are set to cause bit-wise operations (AND, OR, XOR, & NOT) on painted graphical objects in conjunction with the images already on the page and the paint source. A protocol class that supports ROP's supports at least all standard ROP3 operations. ROP's are particularly useful for blending and adding transparency to objects. Note: the document description size and overhead of ROP clipping operations is often large compared to using paths and clipping and therefore should be avoided when possible.

Overview**Introduction**

1.16 Graphics State

**BRUSH PAINT
SOURCE = RED**

All painting operations in PCL XL rely on settings of state attributes related to painting operations. The object containing state attributes for all painting operations is called the *graphics state*.

Graphics state attributes are set by the user to obtain a specific result during painting. For example, the graphics state contains the current paint source (color) associated with the brush. When an object is painted, the device retrieves the brush's current paint source from the Graphics State. If the brush is associated with a valid paint source the device fills the object with the corresponding color or pattern specified. The Graphics State can be viewed as a last-in first-out stack of fixed size. The Graphics State and its attributes and behavior will be explained further later in this document.

2.0 PCL XL Operators, Attributes and Streams

2.1 Protocol Operators

The PCL XL imaging protocol is specifically designed to have a minimum number of operators for ease-of-use, supportability, efficiency, and best performance. The protocol design allows an operator to be used with various data types and a variable number of parameters during the imaging session.

A simple example of this type of flexibility is the plus (“+”) operator in programming languages. The plus operator may be used to add two integer numbers one time it is used and two fractional numbers in another use case. The operator is always known to the user as plus (“+”) even though different data types may be used. This principle of operator parameter flexibility is called “operator overloading” in programming languages and “polymorphism” in object-oriented systems. All PCL XL operators employ this re-use principle.

All operators create, modify, or paint PCL XL protocol objects. Examples of protocol objects are sessions, pages, the current path, bitmap images, and characters. Values associated operators are used to set a protocol object’s attributes.

Specific examples of attribute values are the placement points for an ellipse and the media size selection for a page. The mechanism by which the imaging protocol achieves reuse is the *attribute-list*. Every operator requiring parameters supplied by the user obtains the data from an attribute-list.

An attribute-list contains a set of one or more attribute/value pairs. Each pair in the list contains an attribute identifier and a set of one or more values for the attribute. Below are examples of attribute/value pairs:

2-1(a)	600 600 1300 2500 (attribute value)	BoundingBox (attribute id)
2-1(b)	eLetterPaper (attribute value)	MediaSize (attribute id)
	ePortraitOrientation (attribute value)	Orientation (attribute id)

Figure 2-1. Attribute/Value Pairs.

Three attribute/value pairs are shown in Figure 2-1. An operator knows the valid data type(s) and number of values expected in each value-set by the attribute identifier. The attribute identifiers in Figure 2-1 are represented by a name. The actual attribute identifier is a unique number from a set of attribute identifiers defined for the protocol.

Overview**PCL XL Operators and Attributes**

Attribute lists may hold one or more attribute/value pairs. Many operators need only one attribute/value pair each time they are executed such as the attribute/value pair in Figure 2-1(a). This attribute/value pair is used to define a bounding box with two points (two x, y values). In this case, the bounding box could be used to define the size of an ellipse. Fixed-length, single attribute/value pairs are preferred in serial byte streams to minimize data overhead.

Variable-length attribute lists are used when the exact number and type of attributes required are not predictable. Figure 2-1(b) is an example where the number of attribute/value pairs for the corresponding operator may be variable. One or both of these two attribute/value pairs may be provided with an operator like **BeginPage** to start a new page description. The type and number of attribute/value pairs needed to setup a new page may differ based on user requirements and are therefore variable. The valid construction of an attribute list from attribute-value pairs is discussed later in this document.

When an operator is to be executed, a specific attribute list is associated with the operator. The method of association is implementation-specific. For example, if the device interface is a serial byte-stream, the attribute list may simply be data preceding the operator. If the imaging protocol is communicated to the device via a programming-language API, the attribute list may be a parameter in a function call.

The attribute list mechanism allows the protocol to be easily extended (or reduced) for each target imaging environment. Attribute lists simplify the device protocol interface (parsers and function-based APIs) and enhance error checking prior to operator execution.

The specific operation and behavior of all PCL XL imaging protocol operators is found in this volume.

2.2 Simple Operator Examples

Each operator description in this document includes examples using the JetASM PCL-XL assembler tool syntax. JetASM uses the same operator, attribute, value and enumeration labels that are used throughout this document and its appendices.

This example shows how to paint an ellipse that fits to a one-inch square bounding-box (600 user-units-per-inch assumed):

Step 1: Build Attribute List

```
600 600 1200 1200 BoundingBox // defines a one-inch square bounding box
// attribute id with the corresponding values
```

Step 2: Perform the Operation

```
Ellipse // defines an ellipse path and paints it
```

The following PCL XL sample session paints a single two-inch line on a page:

```
eInch Measure // attribute: basic measure for the session is inches
600 600 UnitsPerMeasure // attribute: 600 units in both X and Y direction
BeginSession // operator: begin the imaging session
```

```
ePortraitOrientation Orientation // attribute: page orientation is portrait
eLetterPaper MediaSize // attribute: size of media for page is letter
BeginPage // operator: begin the page description
```

```
1200 800 Point // attribute: point a which to set the current cursor
SetCursor // operator: set the cursor
```

```
2400 800 EndPoint // attribute: endpoint of a 2 inch line
LinePath // operator: add the line to the current path
```

```
PaintPath // operator: paint the current path
```

```
EndPage // operator: end the page description
```

```
EndSession // operator: end the imaging session
```

The attribute values that begin with “e” such as “eInch” are actually names for enumerated integer values. For example, eInch is a name for the value 0 for the Measure attribute. Values that correspond to enumerated names are listed in **Appendix G**.

2.3 Legal Protocol Operator Sequences

The legal sequence in which PCL XL operators may be executed is specified here in a modified Backus-Naur form (BNF). Specifics of this modified BNF are explained in the appendix. Further operator sequencing constraints are described by the pre- and post-conditions in individual operator descriptions found later in this document. The operator sequencing is identical for all PCL XL implementations of a protocol class regardless of actual protocol implementation. Non-terminals start with a lower-case letter and are bold. Terminals start with an upper-case letter and represent operators. The actual operator association with the attribute list is implementation-specific. The productions for **attributeList** are listed in section 2.3.

```

session ::= { attributeList BeginSession {sessionOperations}0+ attributeList EndSession }
sessionOperations ::= { pageDef | userControlOperation }
pageDef ::= { attributeList BeginPage {imagingOperation}0+ attributeList EndPage }
userControlOperation ::= { {attributeList Comment} | dataSourceOperation | fontControlOperation
    | streamDef | {attributeList ExecStream} | attributeList RemoveStream }
imagingOperation ::= { graphicsStateOperation | paintOperation |
    rasterPatternDef | userControlOperation }
dataSourceOperation ::= { {attributeList OpenDataSource} |
    {attributeList CloseDataSource} }
fontControlOperation ::= { fontHeaderDef | charDef | {attributeList RemoveFont} }
streamDef ::= { {attributeList BeginStream} {attributeList ReadStream}1+
    {attributeList EndStream} }
graphicsStateOperation ::= { attributeList { PushGS | PopGS | SetBrushSource | SetCharAngle |
    SetCharScale | SetCharShear | SetCharAttributes | SetClipIntersect | SetClipMode |
    SetClipRectangle | SetClipReplace | SetClipToPage | SetColorSpace | SetCursor |
    SetCursorRel | SetHalftoneMethod | SetFillMode | SetFont | SetLineCap | SetLineDash |
    SetLineJoin | SetMiterLimit | SetPageDefaultCTM | SetPageOrigin | SetPageRotation |
    SetPageScale | SetPathToClip | SetPaintTxMode | SetPenSource | SetPenWidth | SetROP |
    SetSourceTxMode} }
paintOperation ::= { imagePaintOperation | pathOperation | scanOperation |
    {attributeList Text} {attributeList TextPath} }
rasterPatternDef ::= { {attributeList BeginRastPattern} {attributeList ReadRastPattern}1+
    {attributeList EndRastPattern} }
charDef ::= { {attributeList BeginChar} {attributeList ReadChar}1+
    {attributeList EndChar} }
fontHeaderDef ::= { {attributeList BeginFontHeader} {attributeList ReadFontHeader}1+
    {attributeList EndFontHeader} }
pathOperation ::= { attributeList { ArcPath | BezierPath | BezierRelPath | Chord | ChordPath |
    CloseSubPath | Ellipse | EllipsePath | LinePath | LineRelPath | NewPath | PaintPath | Pie |
    PiePath | Rectangle | RectanglePath | RoundRectangle | RoundRectanglePath } }
imagePaintOperation ::= { {attributeList BeginImage} {attributeList ReadImage}1+
    {attributeList EndImage} }

```

Overview**PCL XL Operators and Attributes**

```
scanOperation ::= { {attributeList BeginScan} {attributeList ScanLineRel }1+  
                  {attributeList EndScan} }
```

2.4 Attribute List Specification Format

This section describes the legal format for operator attribute lists. The format is presented in modified Backus-Naur form (BNF) which is explained in the appendix of this document. The attribute list format is identical for all imaging protocol implementations whether the list is applied in a byte stream or the list is passed as a parameter to a programming language function. All attribute lists are stored and retrieved with a last-in, first-out mechanism (a stack). The right-most attribute/value pair in the list is the last attribute/value pair added to the list. The right-most pair is the first taken out of the list when attributes are read by an operation.

The device internal format of the list and data types for attribute values are implementation-dependent. However, all enumerated values for attribute identifiers and attribute values are the same for all PCL XL devices.

```

attributeList ::= { singleAttributePair | multiAttributeList | nullAttributeList }
singleAttributePair ::= { attributeValue attributeID }
multiAttributeList ::= { singleAttributePair }1+
nullAttributeList ::= { “An implementation-dependent method of specifying that an empty
attribute list is being supplied to the operator” }
attributeValue ::= { value | array }
attributeID ::= { “One in a sequence of enumerated positive integer values representing a unique
attribute identifier” }
value ::= { number | xyValue | boxValue }
array ::= { numberArray }
number ::= { integer | +integer | realNumber }
integer ::= { “an implementation-specific representation of an integer value where the range is
determined by the implementation” }
+integer ::= { “an implementation-specific representation of an unsigned integer value where the range is
determined by the implementation” }
realNumber ::= { “an implementation-specific representation of a real number where the range is
determined by the implementation” }
xyValue ::= { “an implementation-specific representation of two integers or two real numbers, the first
placed in the list represents an x location the second a y location” }
boxValue ::= { “an implementation-specific representation of two integer or real points, the first point
placed in the list represents the top left corner of the box, the second placed in the list
represents the bottom right corner of the box” }
numberArray ::= { “an implementation-specific representation of an array of zero or more integer or
real numbers” }

```

2.5 *Format of Operator Specifications*

Attribute list specifications will be given in BNF in all operator descriptions in this document. Note that attribute identifiers will be included in the BNF specification without their corresponding values. For example:

The attribute list specification:

```
multiAttributeList(class.rev#) ::= { Measure & UnitsPerMeasure }
```

Implies the specification with embedded attribute values:

```
multiAttributeList(class.rev#) ::= { { aMeasureValue Measure } &
  { aUnitsPerMeasureValue UnitsPerMeasure } }
```

For ease-of-reading, attribute value types (i.e. **aMeasureValue**) are not included in the BNF for each attribute in the operator specification. Instead, corresponding attribute value data types are given in a table below the BNF specification for each operator.

For the purpose of clarifying specifications, a comment may precede the attribute value data type as a string of characters followed by a colon (i.e. “{**x1, y1, x2, y2:** boxValue}”). The comments preceding the colon are not part of the protocol syntax. The data ranges for attribute identifier names and enumerated attribute values are given in **Appendix F** and **Appendix G**.

2.6 PCL XL Protocol Class and Revision Numbers

The class and revision number listed for operator's attribute list implies the protocol class and revision number in which an operator is valid. For each new revision of an operator, a new attribute list is specified with the class and revision number noted within parentheses.

The class and revision number for each attribute identifier is located in **Appendix F**. The class and revision number for each attribute enumeration is located in **Appendix G**.

2.7 Error and Warning Reports

Most errors that can occur in a PCL XL device are reported on separate Error Pages following the completion of the job. Each error will be reported in as descriptive manner as possible and supply error codes where applicable. The names of these codes may lead the user to correction of the problem. Internal error codes are also reported when an internal system error occurs for which the user may not have direct control in terms of setting data or operator syntax.

See **Appendix K** and **Appendix J** of this document for more information regarding the error reporting scheme of PCL XL and the specific error codes.

2.8 *Binary Stream Format*

The purpose of this section is to describe the binary representation for PCL XL data and operator streams.

2.8.1 *Binary Stream Format Document Conventions*

Throughout this section and the associated sections in the appendix, several conventions will be used to illustrate binary protocol concepts. These conventions are listed below.

1. The use of `Courier` typeface indicates exact ASCII data.
2. Hexadecimal numbers are indicated with a “0x” prefix, and utilize lower case letters “a” through “f”.
3. Items in *Italics* indicate features or conditions that exist in a development environment, but which will not be exposed in a final product.

2.8.2 Binary Stream Headers

As with all PCL XL streams, a binary stream must begin with a stream header. The stream header precedes the stream body containing the operators and data. The section entitled “User-Defined Streams” in this document explains the header format.

The stream header has special significance for binary streams. The binding identifier field in the stream header specifies the ordering of most-significant to least-significant bytes in the stream for multi-byte binary fields. Below is an example of a PCL XL Stream header for a binary stream:

An example stream header is shown below:

```
)<SP>HP-PCL XL;1;1<CR><LF>
```

The first character in this stream header is a left parenthesis. The left parenthesis specifies that multi-byte fields representing binary data are ordered least-significant byte to most-significant byte from left to right. This ordering only applies to encoded values for operator identifiers, attribute identifiers, and associated attribute data. In least-significant-byte-first binary format, a 32-bit value for the integer 5 would appear as follows:

least significant	0101	0000	0000	0000	most significant
-------------------	------	------	------	------	------------------

Some operators may read binary data from the data source, not the attribute list. A few of these operators require data to always be ordered high byte first (i.e. font and character operators, image operators, and raster pattern operators). If an operator does not specify a byte ordering for binary fields, the ordering is determined by an OpenDataSource attribute (i.e. line and Bezier operators).

2.8.3 Binary Operator and Attribute List Construction

Attribute lists and operators are sent to the PCL XL device in a post-fix manner: The attribute list for a particular operator is specified first, followed by the operator itself. A First-In-Last-Out **stack** mechanism is implemented within the PCL XL device to acquire attribute list components. The PCL XL device accepts data by **pushing** attribute list data onto the stack, until an operator is received. The operator then executes, consuming the attributes by **popping** them off the stack. The stack is cleared after operator execution.

Tags are used to specify operators, attributes, and data types in the stream. An individual tag is a unique element in a set of 8 bit (single byte) values. Currently, 256 tag values are available, although many of those are reserved. A structure has been established allowing for multi-byte tags should there be such a need in the future.

2.8.3.1 Attribute ID Tags

Each attribute within an attribute list consists of an attribute ID and a value associated with that attribute. The attribute ID is a numeric value used to identify which attribute is being referenced.

The ordering of attributes within an attribute list is not important. There is no difference between an attribute list which consists of attributes #2, #3, and #4 and an attribute list with attributes #4, #2, and #3.

Attributes are constructed by first sending the attribute value, followed by the attribute ID.

Attribute IDs are unique. For example, several operators make use of the **BoundingBox** attribute. For all of the operators, the numeric value for the BoundingBox attribute ID remains the same. A complete list of attributes and their corresponding attribute IDs are listed in the appendix.

A single attribute may contain more than one data value. For example, the **UnitsPerMeasure** attribute for the **BeginSession** operator may consist of two numeric values representing the units of measure for the x and y directions of the page.

Attribute IDs are constructed by using an attribute ID tag, followed by the value for the attribute number. The attribute ID tag also indicates the data type of the attribute number. For the current release of this protocol, all attribute numbers are expressed as ubytes (unsigned bytes).

2.8.3.2 Data Type Tags

Each numeric data value within a operator sequence must be preceded by a data type tag. This tag indicates the format or data type of the upcoming data. Each data type supported by the PCL XL device is represented by a unique data type tag. Some operators support several different data types, allowing the user to specify data in the most optimum format. For example, if a print

Overview

PCL XL Operators and Attributes

driver is to provide several point values, all of which are close together, it may be more efficient to represent these point values as 8 bit numbers, rather than 32 bit.

Some data types will contain multiple data values. Two sets of data types have been developed to indicate two and four entry data values, to correspond with point (x/y), and box data types. The point data values are typically utilized to express x and y point coordinates, and the box data types are used to express the coordinates for the upper left and lower right corners of a rectangle or bounding box. A complete list of all of the supported data types is listed in the appendix.

As an example, a point in space can be represented with either 8 bit or 16 bit numbers. The tag representation for these two cases is illustrated below. Note that the ordering is sequential as the data would appear in a I/O stream, with the first item in the I/O stream listed first.

Data Tag uint16_xy	uint16 Data y value		uint16 Data x value
Data Tag ubyte_xy	ubyte Data y value	ubyte Data x value	

Arrays of data values may be specified, where multiple data values of a given data type can be listed. This is accomplished by first specifying a tag, which indicates an array of values of the desired data type, then listing the length of the array. The actual array data follows the array length. The array length is represented by a data type tag followed by the numeric value for the size of the array. The data types used for array lengths are limited to unsigned bytes, and unsigned 16 bit integers. An example of the binary array format is listed below:

Data Tag ubyte_array	Data Tag uint16	uint16 Data 16-bit length (N)	ubyte Data (0)	ubyte Data(1)	...	ubyte Data(N-2)	ubyte Data(N-1)
-------------------------	--------------------	----------------------------------	-------------------	------------------	-----	--------------------	--------------------

2.8.3.3 Operator Tags

A unique operator tag represents each operator. When the PCL XL device receives an operator tag, that operator will examine the stack, and determine what kind of attribute list is on the stack. An attribute list can contain zero, one, or multiple attributes.

If the attribute list is null (i.e. there are no attributes on the stack), then direct processing of the operator will occur. If the attribute list is non-empty, the operator will determine how many attributes are in the list, and pop them off of the stack, and utilize them as appropriate.

2.8.3.4 Embedded Data Tags

Some operators allow data to be read from the data source. PCL XL allows data to be embedded in the operator and data stream, which is the default data source. Whenever data is to be embedded in the stream, the data must be preceded by an “Embedded Data Tag” operator, and an 8-bit or 32-bit value indicating the number of bytes of data that follow. See the Appendix for

Overview**PCL XL Operators and Attributes**

the value of this tag. Operators which require the inclusion of the embedded data tag are identified in the operator specifications. The sequence of events involving the embedded data tag is illustrated below:

< attribute list > <operator> <embedded data tag> <number of data bytes> <embedded data>

An example of embedded data is provided in the Appendix.

2.8.3.5 White Space Tags

Several tags have been identified which are typically utilized as white space characters within a ASCII data stream. These tags are also white space within a PCL XL data stream, and are effectively ignored. This provides the ability to construct PCL XL print jobs using commonly available editors, and programming tools. The white space tags are identified in the Appendix.

2.8.4 Binary Stream Operator Syntax

This section describes the legal syntax for the imaging protocol operators and their associated attribute lists. The syntax is presented in modified Backus-Naur form (BNF) which is explained in the appendix of this document. See the operator sections that follow for detailed attribute-list specifications. Definitions for the tags referenced below are listed in the appendix.

```

operatorSequence ::= { attributeList operatorTag } { embeddedData }opt
operatorTag ::= { "One in a set of positive 8 bit integer values representing each available operator." }
attributeList ::= { singleAttributePair | multiAttributeList | nullAttributeList }
embeddedData ::= { { Tag(edata) dataLength } | { Tag(edatabyte) byteDataLength } } dataSequence }
singleAttributePair ::= { attributeValue attributeID }
multiAttributeList ::= { singleAttributePair }1+
nullAttributeList ::= { the attribute list is empty }
attributeValue ::= { dataType numericValue }
attributeID ::= { { Tag(attr_ubyte) | Tag(attr_uint16) } attributeIDValue }
dataType ::= { valueType | arrayType }
valueType ::= { singleValueType | xyValueType | boxValueType }
arrayType ::= { { singleValueArrayType | xyValueArrayType | boxValueArrayType } arraySize }
singleValueType ::= { Tag(ubyte) | Tag(uint16) | Tag(uint32) | Tag(sint16) | Tag(sint32) | Tag(real32) }
xyValueType ::= { Tag(ubyte_xy) | Tag(uint16_xy) | Tag(uint32_xy) | Tag(sint16) | Tag(sint32_xy) |
    Tag(real32_xy) }
boxValueType ::= { Tag(ubyte_box) | Tag(uint16_box) | Tag(uint32_box) | Tag(sint16_box) |
    Tag(sint32_box) | Tag(real32_box) }
singleValueArrayType ::= { Tag(ubyte_array) | Tag(uint16_array) | Tag(uint32_array) |
    Tag(sint16_array) | Tag(sint32_array) | Tag(real32_array) }
xyValueArrayType ::= { Tag(ubyte_xy_array) | Tag(uint16_xy_array) | Tag(uint32_xy_array) |
    Tag(sint16_xy_array) | Tag(sint32_xy_array) | Tag(real32_xy_array) }
boxValueArrayType ::= { Tag(ubyte_box_array) | Tag(uint16_box_array) | Tag(uint32_box_array) |
    Tag(sint16_box_array) | Tag(sint32_box_array) | Tag(real32_box_array) }
arraySize ::= { arraySizeType numericValue }
arraySizeType ::= { Tag(ubyte) | Tag(uint16) }
attributeIDValue ::= { "The numeric value of the attribute ID indicated by the data type tag previously
    read." }
numericValue ::= { "The numeric value of the data indicated by the data type tag previously read,
    interpretation dependent upon the data type indicated by the tag, and, if it exists, the
    array size." }
dataLength ::= { "A 32-bit numeric value indicating the quantity, in bytes, of embedded data following in
    the data stream }
byteDataLength ::= { "An 8-bit numeric value indicating the quantity, in bytes, of embedded data
    following in the data stream }
dataSequence ::= { embedded data of any form }

```

2.8.5 Binary Stream Operator Example

The following is an example of a fictitious operator. Suppose that this operator was specified in the following format:

OPERATOR: DEMOOPERATOR

Purpose

Demonstrate the binary stream format.

Attribute List Specification

multiAttributeList ::= { BoundingBox & StartPoint }

Attribute ID	Description and {value}
BoundingBox	The region in which the object should be imaged. { xy_value }
StartPoint	The point at which imaging should begin. { xy_value }

Note: This example assumes that the stream header specifies a binary ordering of least significant byte first.

Assume that it was desired to execute this operator with the following attribute values:

BoundingBox: upper left = (10, 20), lower right = (30, 40)
StartPoint: x, y = (105, 5280)

The attribute list could be constructed as:

{ <bounding box values> AttributeID#66 <start point values> AttributeID#79 }

Each of the attribute ID numbers need to be specified as a data type tag, data value pair. Also each of the attribute values need to be associated with their respective data types.

Assume that the bounding box may be constructed for unsigned byte values. There is a binary data type for four unsigned byte values named “ubyte_box.” The BoundingBox attribute may be constructed by specifying the data type tag, and the four values of the coordinates in the bounding box, the attribute ID tag, followed by the attribute ID number as follows:

Data Tag ubyte_box	0x0a	0x14	0x1e	0x28	Attr. Tag ubyte	0x42
	10 _{Dec}	20 _{Dec}	30 _{Dec}	40 _{Dec}		66 _{Dec}

Overview**PCL XL Operators and Attributes**

Assume that the start point may be constructed with two unsigned integer values. There is a binary data type for two unsigned integer values named “uint16_xy.” With 105_{Dec} equal to 0x0069 and 5280_{Dec} equal to 0x14a0, the binary representation for the StartPoint attribute would be:

Data Tag uint16_xy	0x69	0x00	0xa0	0x14	Attr. Tag ubyte	0x4f
-----------------------	------	------	------	------	--------------------	------

79_{Dec}

Combining all of the attributes together, along with a single byte for the DemoOperator gives us the final format of the operation in the stream:

Data Tag ubyte_box	0x0a	0x14	0x1e	0x28	Attr. Tag ubyte	0x42
Data Tag uint16_xy	0x69	0x00	0xa0	0x14	Attr. Tag ubyte	0x4f
Operator DemoOperator						

All data type values, attribute id values, and enumeration values are contained in the appendix of this document.

2.9 PCL XL Design Principles

Current and future PCL XL design is based on the following principles:

1. Backward compatibility is guaranteed within a protocol class where each new protocol class revision executes all old revisions of the same class (e.g. class 1.2 supports all class 1.1 operators)
2. Backward compatibility is encouraged for classes designed for the same category of devices (e.g. class 2.0 designed for printing supports all class 1.x operators since 1.x was designed for printing)
3. The imaging model is device-independent such that pages described in a PCL XL stream shall have the same appearance on any compatible device regardless of device resolution
4. The number of operators in any protocol class shall be kept to a minimum
5. Each operator shall perform a single logical function to minimize complexity of pre- and post-conditions
6. State dependencies among operators shall be kept to a minimum
7. Duplicate functionality among operators shall be kept to a minimum

session**Session Operators**

3.0 Session Operators

Session operators are the set of operators that provide the user control over the definition of a session and its default environment (see Section 5.0 for graphics state defaults). The operator set includes those operators that define session boundaries, define page boundaries, and define the data source for session read operations.

3.1 *Beginning and Ending a Session*

A *session* is the context in which a user interacts with a PCL XL device. The user begins a new session to start interactions with the device. The user may only use PCL XL operations to interact with a device during an active PCL XL session.

Sessions have both *required* and *optional* attributes. PCL XL devices require a units per measure attribute to be set for a new session. This attribute identifies the page coordinate system units in which the user prefers to describe pages during the session (i.e. 600 units-per-inch). The units per measure for a session may be different from the internal device resolution.

If an error occurs during a PCL XL operation, all protocol operators and data will be ignored until an EndSession operator is executed.

session

Beginning and Ending a Session**OPERATOR: *BEGINSESSION*****Purpose**

Begin a new session in which imaging operations shall be performed.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList ::= { Measure & UnitsPerMeasure & {ErrorReport}_{opt} }

Attribute ID	Description and {value}
Measure	The measure for each unit. { eInch eMillimeter eTenthsOfAMillimeter }
UnitsPerMeasure	The user resolution units in the x and y directions desired by the user for placing points during the session (i.e. xUnits=600, yUnits=600 for a 600 pixels-per-inch device). { xUnits (+number), yUnits (+number): xyValue }
ErrorReport	An enumeration indicating the method with which errors are reported to the user (default is eNoReporting). Prior to a BeginSession, default is eErrorPage. { see attribute enumeration table below }

Attribute Enumeration Table

Attribute	Enumeration	Meaning
ErrorReport	eNoReporting	Device will not perform any error reporting
ErrorReport	eBackChannel	Device will report errors through the back channel {the meaning of “back channel” is device-dependent}
ErrorReport	eErrorPage	Device will report errors by printing or displaying an error page
ErrorReport	eBackChAndErrPage	Device will report errors through the back channel and print or display an error page

session	Beginning and Ending a Session
----------------	---------------------------------------

ErrorReport	eNWBackChannel	Device will report errors only (no warnings) through the back channel {the meaning of “back channel” is device-dependent}
ErrorReport	eNWErrorPage	Device will report errors only (no warnings) only by printing or displaying an error page
ErrorReport	eNWBackChAndErr Page	Device will report errors only (no warnings) through the back channel and print or display an error page

Postcondition

The device has been put into a mode in which session operators may be accepted.

Examples

Begin session for a simple 600 dpi print job

```
uint16_xy 600 600 UnitsPerMeasure
ubyte eInch Measure
ubyte eBackChAndErrPage ErrorReport
BeginSession
```

Begin session for a simple 1200 dpi print job with with error and warning pages turned off.

```
uint16_xy 1200 1200 UnitsPerMeasure
ubyte eInch Measure
ubyte eNoReporting ErrorReport
BeginSession
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

session

Beginning and Ending a Session*OPERATOR: ENDSSESSION***Purpose**

Halt protocol processing and end the current session.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The session has been terminated.

If a data source was left open, it will be closed by EndSession with no error reported.

All session attribute settings have been lost and must be redefined for the next session.

Example

```
EndSession
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

session**Beginning and Ending a Page**

3.2 Beginning and Ending a Page

A *page* in a PCL XL device is a surface upon which painting operations may be performed. This surface represents the physical media (printed page or display) upon which objects are painted. The user must begin a new page before any painting operations may begin. Page imaging operations may paint text, vector, and raster objects on a physical page.

Pages have attributes that define and effect the real page received or viewed. For example, the size, type, source, and destination for the physical page media are attributes that may be associated with a page.

Painting operations may only occur within a Begin/EndPage context. The graphics state is initialized to default attributes each time BeginPage occurs (see Section 5.0 for graphics state defaults).

session

Beginning and Ending a Page**OPERATOR: *BEGINPAGE*****Purpose**

Begin the definition of a new page.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

```
multiAttributeList ::= { Orientation & {MediaSource}opt &
{ MediaSize | {CustomMediaSize & CustomMediaSizeUnits} } &
{MediaType}opt &
{MediaDestination}opt &
{ SimplexPageMode | {DuplexPageMode & {DuplexPageSide}opt}opt }
```

Attribute ID	Description and {value}
Orientation	The orientation of the logical page. {ePortraitOrientation eLandscapeOrientation eReversePortrait eReverseLandscape}
MediaSize	An enumeration for size of the target surface. { see attribute enumeration table below } or a case sensitive string matching an entry in the current printer's list of supported paper sizes.
CustomMediaSize	The dimensions of a custom media size. { xy_value }
CustomMediaSizeUnits	An enumeration for the unit of measure for the custom media size dimensions. { eInch eMillimeter eTenthsOfAMillimeter }
MediaType	A string "name". The attribute data type of ubyte_array is used to send the string name of the media type desired. {ubyte_array}
MediaSource	The source from which the media will be pulled for a physical-media device. If not included, the device will use the most recent MediaSource setting by a BeginPage operator. If MediaSource has not been set by a BeginPage operator, the device's default setting will be used. { see attribute enumeration table below }

session	Beginning and Ending a Page	
MediaDestination	<p>The destination for each page after it is printed. If not included, the device will use the most recent MediaDestination setting by a BeginPage operator. If MediaDestination has not been set by a BeginPage operator, the device's default setting will be used.</p> <p>{ see attribute enumeration table below }</p>	
SimplexPageMode	<p>An enumeration requesting that pages be rendered one page per physical media page and rendered on the front side of each physical media page. If not included, the device will use the most recent simplex/duplex setting by a BeginPage operator. If simplex/duplex has not been set by a BeginPage operator, the device will use the device default setting.</p> <p>{eSimplexFrontSide}</p>	
DuplexPageMode	<p>An enumeration requesting that pages be rendered on both sides of a physical media page and oriented for either horizontal or vertical binding of the physical pages. If not included, the device will use the most recent simplex/duplex setting by a BeginPage operator. If simplex/duplex has not been set by a BeginPage operator, the device will use the device default setting.</p> <p>{eDuplexHorizontalBinding eDuplexVerticalBinding}</p>	
DuplexPageSide	<p>An enumeration specifying the side of the media on which the current page should be rendered. If not included, the device will use the most recent simplex/duplex setting by a BeginPage operator. If simplex/duplex has not been set by a BeginPage operator, the device will use the device default setting.</p> <p>{eFrontMediaSide eBackMediaSide}</p>	

Attribute Enumeration Table

Attribute	Enumeration	Meaning
MediaSize	eA3Paper	The imagable area for the page will be set to A3 paper size.
MediaSize	eA4Paper	The imagable area for the page will be set to A4 paper size.
MediaSize	eA5Paper	The imagable area for the page will be set to A5 paper size.
MediaSize	eA6Paper	The imagable area for the page will be set to A6 paper size

session	Beginning and Ending a Page	
MediaSize	eLegalPaper	The imagable area for the page will be set to legal paper size.
MediaSize	eLedgerPaper	The imagable area for the page will be set to ledger paper size.
MediaSize	eLetterPaper	The imagable area for the page will be set to letter paper size.
MediaSize	eExecPaper	The imagable area for the page will be set to executive paper size.
MediaSize	eJB4Paper	The imagable area for the page will be set to JB4 paper size.
MediaSize	eJB5Paper	The imagable area for the page will be set to JB5 paper size.
MediaSize	eJB6Paper	The imagable area for the page will be reset to JB6 paper size
MediaSize	eJPostcard	The imagable area for the page will be set to J postcard size.
MediaSize	eJDoublePostcard	The imagable area for the page will be set to J double postcard size.
MediaSize	eB5Envelope	The imagable area for the page will be set to B5 envelope size
MediaSize	eC5Envelope	The imagable area for the page will be set to C5 envelope size.
MediaSize	eCOM10Envelope	The imagable area for the page will be set to COM10 envelope size.
MediaSize	eDLEnvelope	The imagable area for the page will be set to DL envelope size.
MediaSize	eMonarchEnvelope	The imagable area for the page will be set to Monarch envelope size.
MediaSize	"String"	See appendix __ for device dependant string labels for supported media sizes
MediaType	"String"	See appendix __ for device dependant string labels for supported media types
MediaSource	eAutoSelect	Automatically select the media source.
MediaSource	eDefaultSource	Choose the default media source.
MediaSource	eEnvelopeTray	Choose the media in the envelope tray.
MediaSource	eLowerCassette	Choose the media in the lower cassette.

session	Beginning and Ending a Page	
MediaSource	eManualFeed	Choose the media in the manual feed source.
MediaSource	eMultiPurposeTray	Choose the media in the multi-purpose tray.
MediaSource	eUpperCassette	Choose the media in the upper media cassette.
MediaSource	eThirdCassette	Selects the third cassette on printers so equipped.
MediaDestination	eDefaultDestination	Sets the paper output to the bin currently set by the operator.
MediaDestination	eFaceDownBin	Sets the paper output to the face-down bin
MediaDestination	eFaceUpBin	On printers so equipped, the paper output will be the face-up output bin.
MediaDestination	eJobOffset	On printers so equipped, the paper output will be job offset bin.

Postcondition

The device has been put into a mode in which painting operators may be accepted.

The graphics state attributes have been initialized to default values.

Example

Beginning a simple page.

```
ubyte ePortraitOrientation Orientation
ubyte eLetterPaper MediaSize
BeginPage
```

Beginning a page printing on Executive sized paper with duplexing enabled.

```
ubyte ePortraitOrientation Orientation
ubyte eExecPaper MediaSize
ubyte eDuplexVerticalBinding DuplexPageMode
BeginPage
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

MediaSize Notes:

MediaSize can be selected by enumeration or by sending a string name for the size desired. Every device stores a list of supported media sizes that includes the code numbers, the size information, and a name string. By scanning the device's list for a name matching the one

session	Beginning and Ending a Page
----------------	------------------------------------

sent to PCL XL by the user, the correct enumeration for that size can be obtained by XL and from that PCL XL gains the information on size and format. This way XL can support paper sizes it currently knows nothing about but that future devices support.

There are four cases in which PCL XL selects the paper size for the user. These cases are:

- The size specified is not a valid enumeration value.
- The size specified is a valid enumeration but not supported by the device.
- The string name sent does not match any name in the device's list of supported sizes.
- The device does not support a user defined custom paper size.

In the first three cases PCL XL will use the default paper size specified by front-panel settings.

In the last case, when the user sends down the dimension for a custom sized page, PCL XL will query the device to determine if that size is supported. If the device responds with an OK, then the page is processed normally. If the device responds that the dimensions are not supported, PCL XL will search the devices media size list for a standard size that most closely matches the desired size, select that media size, print, and issue a warning.

Example

Selecting paper size by string

```
ubyte_array (Letter) MediaSize
```

Selecting a paper size by enumeration

```
ubyte_array eLetterPaper MediaSize
```

MediaType Notes:

PCL XL normally passes the media type name to the device without checking validity of the type. There are no PCL XL warnings or errors associated with media types. There are two cases where PCL XL will respond to the media type the user defines.

- If the media type is "Transparency", PCL XL will disable duplexing automatically until a non-transparency media type is specified.
- If the media type is "PrePunch" and duplexing is enabled, and the printer is a landscape feed device, PCL XL will rotate the image on the page 180° before printing to ensure that the punched holes end up on the left edge of the page.

For HP LaserJet Printer devices, the MediaType attribute is persistent through the end of the session unless overridden by another MediaType setting on a successive page.

Example

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 51	BLD 4600
-----------	-----------------	---------	---------	----------

session**Beginning and Ending a Page**

Selecting a media type for letterhead paper.

```
ubyte_array (Letterhead) MediaType
```

Selecting media type as plain, use an empty (NULL) string

```
ubyte_array () MediaType
```

MediaSource Notes:

There are three cases in which PCL XL selects the paper source for the user. These cases are:

- The user does not specify a media source in BeginPage.
- The media source specified is not a valid enumeration value.
- The media source is a valid enumeration but not supported by the device.

In all cases, PCL XL will choose the last valid media source selected by a previous BeginPage, or the default media source if the current page is the first page in the session. The default media source is specified by front-panel settings or via PJL.

For HP LaserJet Printer devices, the MediaSource attribute is persistent through the end of the session unless overridden by another MediaSource setting on a successive page.

Example

Allowing the printer to select a tray based on media size, type, etc.

```
ubyte eAutoSelect MediaType
```

Selecting the manual feed tray for input.

```
ubyte eManualFeed MediaType
```

Selecting an external input device's first tray (all external input trays are numbered starting with 8).

```
ubyte 8 MediaType
```

MediaDestination Notes:

There are three cases in which PCL XL selects the paper destination for the user. These cases are:

- The user does not specify a media destination BeginPage,
- The media destination specified is not a valid enumeration value, and
- The media destination is a valid enumeration but not supported by the device.

In all cases, PCL XL will choose the last valid media destination selected by a previous BeginPage or the default media destination if the current page is the first page in the session. The default media destination is specified by front-panel settings or via PJL.

session**Beginning and Ending a Page**

For HP LaserJet Printer devices, the MediaDestination attribute is persistent through the end of the session unless overridden by another MediaDestination setting on a successive page.

Example

Selecting the face up bin on printers so equipped.

```
ubyte eFaceUpBin MediaDestination
```

Selecting an external device's first output bin (all external output bins are numbered starting with 5).

```
ubyte 5 MediaDestination
```

Duplexing Notes:

If the output device is in duplex mode and a change in MediaSource, MediaDestination, MediaSize, MediaType or DuplexBinding occurs in BeginPage for the front side of a page, the device will send one blank page to finish the duplexed page. The new front page has the new BeginPage attributes.

If the output device is in duplex mode and a change for MediaSource, MediaSize, MediaType, or DuplexBinding occurs in BeginPage for the backside of a page, the device will output two blank pages. These pages: 1) finish the back side for the current page and 2) provide a front side for the new back page having the new BeginPage attributes.

session

Beginning and Ending a Page**OPERATOR:** *ENDPAGE***Purpose**

End the definition of a page and cause it to be printed or displayed.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { PageCopies }_{opt}

Attribute ID	Description and {value}
PageCopies	The number of pages to be displayed or printed. {+integer}

Postcondition

The newly defined page(s) have been scheduled for printing or display.

Page-persistent and temporary-persistent raster patterns defined for the pen and/or brush have been removed.

Fonts and characters defined any time during the session have been retained.

User-defined streams defined any time during the session have been retained.

Patterns with session persistence have been retained.

Example

Simple end page operator

```
EndPage
```

End page operator with mutiple copies.

```
ubyte 3 PageCopies
EndPage
```

End of page operator where no printing is desired (see following comment).

```
ubyte 0 PageCopies
EndPage
```

Comments

Setting PageCopies to zero directs PCL XL to flush the current page. However, all persistent attributes set by the page's data will still be set. *Flushing the current page does not clear any settings made during the page's processing.*

session	Beginning and Ending a Page
---------	-----------------------------

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

session**Adding a Comment**

3.3 Adding a Comment

In some environments it may be desirable to add a free-form comment during the protocol sequence. The main use for this operator is in devices that use a byte stream protocol. The comment may represent font character codes or any other data desired by the user.

This is effectively a no-operation in PCL XL devices. In other words, the operator does not alter device state that would effect any other operator. The data associated with the comment is discarded upon operator execution.

session**Adding a Comment*****OPERATOR: COMMENT*****Purpose**

Add a free-form comment in the protocol sequence.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { CommentData }_{opt}

Attribute ID	Description and {value}
CommentData	An integer array of comment data. { integerArray }

Postcondition

The imaging state of the device is unaffected.

Example

```
ubyte_array (Text describing some event or operation) CommentData
Comment
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

session**Opening and Closing a Data Source**

3.4 Opening and Closing a Data Source

A data source is the source of byte-oriented data for all protocol read operators (i.e. Begin/EndFontHeader, Begin/End Char, ReadImage, ReadRastPattern, and all path operations that read points from the data source such as LinePath, BezierPath, etc.).

In many PCL XL operations, it is impractical to submit all the data as part of the operator's attribute list. These operations read potentially large blocks of byte data at a time for objects such as fonts, bitmap images, raster patterns, and complex path regions.

A data source must be open prior to any data read operation. The currently open data source is the source of byte data for the next data read operation. There may be one and only one open data source in the imaging protocol at a time.

A data source may be left opened following a read operator's execution such that the open data source may be used for multiple classes of operators. For example, a single data source could contain font header information, followed by character definition information. A data source may be opened for a **ReadFontHeader** operator and remain open for successive **ReadChar** operators. The data source may then be closed following the last **ReadChar** operator. A data source should be closed prior to initiating an **EndSession** operator.

In byte-stream protocols the data source is usually the default standard input channel. For example, if the default data source was opened in the **OpenDataSource** operator for a byte-stream, the data for a read operator may be embedded in the data stream just following the operator. If data source data are embedded in a PCL XL operator stream, there is a specific method of specifying the length of raw data blocks in the stream. See the section(s) in this document on stream formats.

If all operators in a session require data solely from the default data source, the data source may remain open for the entire session. In non-byte-stream protocols, such as programming language interfaces it is usually necessary to open a file or memory buffer as a data source.

session**Opening and Closing a Data Source*****OPERATOR: OPENDATASOURCE*****Purpose**

The purpose of this operator is to define the source of data for protocol read operations.

The data source is for read operations that cannot obtain all required data from the associated attribute list. There may be one open data source at a time in the device.

The open data source may be used for more than one operation. For example, a single data source may be used for Begin/EndFontHeader and successive Begin/End ReadChar operations.

The data source should be closed prior to initiating the EndSession operator.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

There is no currently-open data source (e.g. the last data source opened in the session was successfully closed with the CloseDataSource operator).

Attribute List Specification

multiAttributeList ::= { SourceType & DataOrg }

Attribute ID	Description and {value}
SourceType	An enumerated value specifying the data source to be opened. If “eDefaultDataSource” is specified, the data source is the default source for the device (i.e. the standard input stream or “stdin”). { eDefaultDataSource }
DataOrg	An enumerated value specifying the organization of binary data to be read. The enumeration eBinaryHighByteFirst means that the high (most significant) byte is first for multi-byte binary fields (i.e. 16- or 32-bit integers). The enumeration eBinaryLowByteFirst means that the low (least significant) byte is first for multi-byte binary fields within the data source. { eBinaryHighByteFirst eBinaryLowByteFirst }

session**Opening and Closing a Data Source****Postcondition**

The source for protocol read operations has been opened and the next byte read-pointer is positioned at the beginning of the data source.

Example

```
ubyte eDefaultDataSource SourceType  
ubyte eBinaryLowByteFirst DataOrgOpenDataSource
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

Some operators that read multi-byte binary fields from the data source require data to always be ordered high byte first (i.e. font and character operators, image operators, and raster pattern operators). If an operator does not specify a fixed byte ordering for binary fields, the ordering is determined by this OpenDataSource attribute (i.e. line and Bezier operators).

See also Begin/EndFontHeader, Begin/End Char, ReadImage, ReadRastPattern, ScanLineRel, and all vector operations that read points from the data source (i.e. LinePath, BezierPath, etc.).

session**Opening and Closing a Data Source*****OPERATOR: CLOSEDATASOURCE*****Purpose**

Close the data source for protocol read operations.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

 nullAttributeList

Postcondition

The source for protocol read operations has been closed.

If the data source was a file, the file has been closed and data in the file left unchanged.

Example

```
    CloseDataSource
```

Comments

See **Appendix K** for related error codes.

If the data source was not open prior to the execution of this operator, no operation is performed.

See also Begin/EndFontHeader, Begin/End Char, ReadImage, ReadRastPattern, ScanLineRel, and vector operations that use the data source to read points.

4.0 Font Control Operators

4.1 *Defining Fonts and Characters*

A PCL XL device allows the user to place characters of a font anywhere on the page. The actual font technology available in the device (TrueType, Bitmap, etc.) to describe characters is implementation-dependent.

Each character is treated as an independent graphical object. A single character that is painted is placed at the current cursor location. The current cursor location is defined or changed by path construction operations or text placement operations.

The imaging protocol allows one or more characters to be placed on the page in a single operation. The first character is placed at the current cursor. The remaining characters in a multi-character operation are placed at corresponding escapements (character spacings) provided by the user.

Character sizes are always specified in user units. The default user units for a page are equivalent to the session user units. As the user scales the page CTM (see `SetPageScale`), the character size per user unit also changes if the characters is an outline character (i.e. TrueType). Bitmap characters cannot be scaled nor rotated.

Painted characters are scaled and rotated according to page CTM manipulations by the user (i.e. `SetPageRotation` and `SetPageScale`). Characters may also be rotated, scaled, and sheared in an additive manner to the current page CTM through the setting of corresponding character rotation, scaling, and shearing attributes (i.e. `SetCharAngle`, `SetCharScale`, and `SetCharShear`). The page scale and rotate is applied to the character before the character scale and rotate settings are applied.

X and Y character spacing may be specified for correct placement of international characters. 8-bit and 16-bit character codes are supported in PCL XL.

See Appendix L for more information on font formats and on the specific use of font definition operators.

OPERATOR: BEGINFONTHEADER**Purpose**

Define a device font header.

The font name is part of the **BeginFontHeader** attribute list. This will be the font name used in the **SetFont** and **BeginChar** operators.

The set of font header formats supported is implementation-dependent. See the corresponding font format specifications in the appendix of this document.

After a font header is defined, the user may define characters for the font using the **BeginChar**, **ReadChar**, and **EndChar** operators. Depending on the implementation, some font headers may allow, or require, all characters to be defined with the font header (see appendix for details).

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The device must support the font imaging technology for the font to be defined.

Attribute List Specification

singleAttributePair ::= { FontName & FontFormat }

Attribute ID	Description and {value}
FontName	The name of the font for which the header is being defined. { +integerArray }
FontFormat	A header format number identifying the format of the font header being downloaded. See the device documentation for a list of font formats supported by the target device. { +integer }

Postcondition

The device has been put into a mode in which font header data may be accepted.

Example

```

ubyte_array (Courier SWA4000) FontName
ubyte 0 FontFormat
BeginFontHeader

```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

font control**Defining Fonts and Characters*****OPERATOR: READFONTHEADER*****Purpose**

Read a segment of a font header from the current protocol data source.

Font header data is read from the data source in a byte-oriented fashion. The first byte read always corresponds to the first byte in the font header format specification. The second byte read corresponds to the second byte in the font header format specification, and so on. No byte-swapping is performed on binary data within the font header data stream. The first byte for a binary field within font header data is always the most significant byte. The last byte for a binary field is always the least significant byte.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The data source must be open from which font header data may be read.

Attribute List Specification

multiAttributeList ::= { FontHeaderLength }

Attribute ID	Description and {value}
FontHeaderLength	The amount of data (in bytes) required to describe the font header. { +integer }

Postcondition

The device has read one segment of data describing a font header. The format of the data must match the format implied by the FontFormat attribute of BeginFontHeader.

Example

```
uint16 54 FontHeaderLength
ReadFontHeader
```

The ReadFontHeader operator is immediately followed by the font header data.

```
dataLength 54
hex_raw* [
00 00 02 75 01 00 00 ff 47 54 00 00 12 e8 00 01
00 00 00 08 00 80 00 03 00 00 63 76 74 20 00 9e
2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b
2b 2b 1d 00 00 00
]
```

Comments

font control**Defining Fonts and Characters**

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

OPERATOR: *ENDFONTHEADER*

Purpose

Complete the font header definition process and end the font header definition protocol.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

A font header has been read from the current open data source. The next data source read-position has been updated to point to the element just following the data read.

A font header has been defined for the device and the font header definition protocol has been terminated.

Example

```
EndFontHeader
```

Comments

See **Appendix K** for related error codes.

OPERATOR: BEGINCHAR**Purpose**

Set up the protocol for the definition of one or more characters for an existing font.

The character format is implementation-dependent. See the corresponding character format specifications for the desired device.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The font for which the characters are to be defined must already exist in the device.

The data source must be open from which character definitions are to be read.

Attribute List Specification

singleAttributePair ::= { FontName }

Attribute ID	Description and {value}
FontName	The name of the font to which the character will be added. { +integerArray }

Postcondition

The device has been put into a mode in which font character data may be accepted.

Example

```
ubyte_array (Courier SWA4000) FontName
BeginChar
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

font control**Defining Fonts and Characters*****OPERATOR: READCHAR*****Purpose**

Read character definition data from the current protocol data source. The data defines a single character for an existing font.

Character data is read from the data source in a byte-oriented fashion. The first byte read always corresponds to the first byte in the character format specification. The second byte read corresponds to the second byte in the character format specification, and so on. No byte-swapping is performed on binary data within the character data stream. The first byte for a binary field within character data is always the most significant byte. The last byte for a binary field is always the least significant byte.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The data source must be open from which character definitions are to be read.

The data source must have at least one character definition remaining matching the attribute list definition for BeginChar and the attribute list definition for this ReadChar.

Attribute List Specification

multiAttributeList ::= { CharDataSize & CharCode }

Attribute ID	Description and {value}
CharCode	The character code value. { integer }
CharDataSize	The amount of data (in bytes) required to describe the character. { +integer }

Postcondition

The device has read one block of data describing a character for the font specified by BeginChar.

Example

```
uint16 70 CharCode
uint32 356 CharDataSize
ReadChar
```

The ReadChar operator should be immediately followed by the character data.

```
dataLength 356
hex_raw* [
01 00 01 62 00 29 00 01 00 5c 00 00 04 5c 04 a2
```

font control	Defining Fonts and Characters
---------------------	--------------------------------------

```

00 38 00 c6 41 46 00 35 00 2f 00 36 00 32 00 28
00 2b 00 26 00 1c 00 18 00 11 00 2e 01 43 00 2b
00 ef 00 26 00 e8 00 32 00 ef 00 36 00 1d 01 43
00 23 01 43 00 20 00 ec 00 24 00 11 00 ba 00 36
00 ed 00 18 00 e5 00 0e 00 00 00 c5 00 07 00 07
00 00 00 27 00 25 00 19 00 23 00 1c 00 18 00 12
00 0e 00 08 00 04 00 10 00 0b 00 01 00 04 00 00
00 35 00 27 00 31 00 2e 00 23 00 31 00 1c 00 04
00 2e 00 37 00 25 00 0f 00 1c 00 15 00 0b 00 10
00 39 10 dc c4 32 c4 fc 3c c4 c4 10 ee 10 ee 32
11 12 39 11 12 17 39 11 12 39 11 12 39 39 31 00
2f ee 32 fe ee ee 32 fe e6 e6 10 e4 fe f4 e6 11
12 39 11 12 39 11 12 39 39 30 25 33 32 16 15 14
06 23 21 22 26 35 34 36 33 33 33 11 23 22 26 35
34 36 33 21 32 16 15 11 14 06 23 22 26 35 35 21
11 21 35 34 36 33 32 16 15 15 14 06 23 22 26 35
35 21 11 02 5e 27 2a 26 24 22 fe 13 22 24 27 2f
21 3f 3f 4c 2b 24 22 03 75 22 23 23 22 2a 1e fd
e1 01 08 1c 24 20 1d 1b 22 24 1c fe f8 8d 21 24
23 25 25 23 25 20 03 85 1f 29 23 25 29 27 fe fe
23 23 2c 4d 8f fe af 1a 49 32 26 2a fe 32 26 32
49 1d fe 45]
    
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

OPERATOR: ENDCHAR

Purpose

Complete the character definition process and end the character definition protocol.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

Characters have been specified for the corresponding font and the character definition protocol has been terminated.

Example

EndChar

Comments

See **Appendix K** for related error codes.

4.2 *Removing Fonts*

Temporary fonts defined through the Begin/EndFontHeader and Begin/Read/EndChar operators may be removed from the device. This operation frees internal memory that may be used for additional fonts or other device resources.

font control**Removing Fonts*****OPERATOR: REMOVEFONT*****Purpose**

Remove a device font previously defined with Begin/EndFontHeader. Removes all character and header definitions associated with the font.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { FontName }

Attribute ID	Description and {value}
FontName	The name of the font to be removed. { +integerArray }

Postcondition

The device font specified by the FontName attribute has been removed from the device along with all associated character definitions.

Example

```
ubyte_array (Courier SWA4000) FontName
RemoveFont
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Graphics State Operators**

5.0 Graphics State Operators

Graphics state attributes are set by the user to obtain a specific result during painting. For example, the graphics state contains the current paint source (color) associated with the brush. When an object is painted, the device retrieves the brush's current paint source from the graphics state. If the brush is associated with a valid paint source PCL XL fills the object with the corresponding color or pattern specified.

All graphics state attributes are set by commands preceded by “Set...” with the exception of the current path which is set by path operators (i.e. Begin/EndPath, Arc, Line, etc.). Graphics state attributes and defaults at each **BeginPage** are listed in the table below:

GS Attribute	Description	Default
BrushSource	Paint source currently associated with the brush	RGB black
CharAngle	The angle at which to draw characters (additive to page CTM)	0
CharScale	The scaling factor for characters (additive to page CTM)	x=1, y=1
CharShear	The shearing factor for characters (additive to page CTM)	x=0, y=0
ClipMode	The mode determining even-odd or non-zero winding construction of the clip path	eNonZeroWinding
ColorSpace	Current color space	RGB
CurrentClipPath	The region defining the current clip path	imagable area of the page
CurrentFont	The font that will be used for painting characters	no font defined
CurrentPath	The region defining the current path	no path defined
DefaultCTM	The default page coordinate transformation matrix	Device dependent
DitherAnchor	The x,y point in which the dither is anchored	0,0
DitherMatrixID	A user definable identifier used to select a specific dither matrix	none
HalftoneMethod	The method used for halftone operations	DitherMatrix eDeviceBest
FillMode	The mode in which closed paths should be filled	eNonZeroWinding
LineCap	The shape to draw on the end of lines (open subpaths)	eButtCap
LineDash	The dash style to use when stroking lines with a pen	solid line
LineJoin	The shape to draw where two lines meet at an angle	eMiterJoin
MiterLimit	The length limit on miter join shapes	10.0
PageCTM	Current page coordinate transformation matrix	session defaults at the current page orientation
PaintTxMode	The current transparency mode for patterns	eOpaque
PaletteID	A user definable identifier used to select a specific palettes	none
PatternAnchor	The x,y point in which the brush and pen patterning is started	0,0
PenSource	Paint source currently associated with the pen	RGB black
PenWidth	The current width (in user units) for pen stroking operations	1 user unit
ROP	The current raster operation in effect	ROP3=252
SourceTxMode	The current transparency mode for source objects	eOpaque

graphics state

Graphics State Operators

5.1 *Saving and Restoring the Graphics State*

To reduce workload and complexity for the user, there is a graphics state stack upon which the attribute values of the current graphics state may be saved. Saving the current graphics state is accomplished by a “push” operation. The push graphics state operation does the following:

1. Creates a new graphics state object and places it on top of the graphics state stack
2. Copies the attribute values from the current graphics state to the new graphics state object on the top of the graphics state stack

Each successive push operation is non-destructive to existing copies of the graphics state on the stack. Each newly pushed copy of the graphics state becomes a new top-of-stack, leaving the previously pushed graphic state entries unaffected. The maximum number of push operations allowed during a session is device-dependent. The LaserJet 6 class of printers set the maximum graphic state save level to 20, LaserJet 4000 class printers set it to 32.

The graphics state stack is a last-in, first-out mechanism. The most recently pushed graphics state object may be retrieved through a “pop” operation. The *pop graphics state* operation does the following:

- 1) Deletes any obsolete graphic state components and causes the previously pushed graphics state to become the current top of stack.

The push/pop combination allows the user to push (save) the graphics state, then modify the page CTM, pens, brushes, etc. for specific painting effects, and finally pop (restore) all graphics state attributes to their previous values.

The path and clippath are also saved during a push operation. The efficiency of *gsave/grestore* is directly related to the size (number of path/clippath elements) of the path and clip path.

graphics state**Saving and Restoring the Graphics State**

Figure 5-1 provides an example of how graphics state push/pop operations effect painting. In the example, the rectangle painted in step 7 is brushed with the same paint source as the ellipse in step 2. This is due to the pop operation in step 6 that restored all graphics state attributes to previous values.




Protocol Operation	Page Output
1. Set Graphics State "Brush Source" Attribute to a Gray Level	...
2. Paint an Ellipse	
3. Push Graphics State	...
4. Set Graphics State "Brush Source" Attribute to Black	...
5. Paint an Ellipse	
6. Pop Graphics State	...
7. Paint a Rectangle	

Figure 5-1. Effects of Push/Pop Graphics State on Painting.

graphics state**Saving and Restoring the Graphics State*****OPERATOR: POPGS*****Purpose**

Deletes any obsolete graphic state components and causes the previously pushed graphics state to become the current top of stack.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The graphics state stack size is not zero.

Attribute List Specification

nullAttributeList

Postcondition

All of the attributes of the graphics state object on top of the graphics state stack have been copied to the attributes of the current graphics state object.

Example

PopGS

Comments

If the graphic state level prior to the PopGS operator equals 0, the PopGS operation is ignored and a warning is issued.

See **Appendix K** for related error codes.

graphics state**Saving and Restoring the Graphics State*****OPERATOR: PUSHGS*****Purpose**

Push a copy of the current graphics state on top of the graphics state stack.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The graphics state stack size is not at its limit.

Attribute List Specification

nullAttributeList

Postcondition

A new graphics state object has been placed on top of the graphics state stack.

All of the attributes of the current graphics state have been copied to the attributes of the graphics state object on top of the graphics state stack.

Example

PushGS

Comments

If the graphic state level exceeds the maximum defined for that device, no operation is performed and an error is raised.

See **Appendix K** for related error codes.

5.2 Setting and Changing the Cursor Location

The current cursor location defines the initial point for relative placement operations involving paths, text and images. The cursor location is always specified in user units. A new cursor location may be defined with absolute points or with points relative to the current cursor location.

The cursor location is updated by all path operations not defining a closed object. For example, operations for lines, arcs, and Bezier's all update the cursor location to be at the end of the path drawn. Operations involving closed objects defined by a bounding box leave the current cursor at the top left corner of the bounding box.

Text operations are always relative to the current cursor location. After painting or describing a character, the text operation leaves the current cursor at the character's escapement described in the "x" and/or "y" value offsets from the current cursor (see Section 6.13 on painting text).

Image painting operations are also relative to the current cursor position (see Section 6.10 on painting raster images).

graphics state**Setting and Changing the Cursor Location****OPERATOR: SETCURSOR****Purpose**

Set the cursor to a new x, y location.

This operation defines the starting point for a new subpath and the starting location for painted text.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { Point }

Attribute ID	Description and {value}
Point	The user space x, y point at which the cursor will be located. The point will be transformed by the CTM in effect. { x, y: xyValue }

Postcondition

The cursor has been set to the new absolute x, y location specified in the SetCursor attribute list.

Example

```
sint16_xy 0 86 Point
SetCursor
```

Comments

Any valid coordinate is acceptable. If the coordinate is off the imangible area it will be clipped when painted.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting and Changing the Cursor Location*****OPERATOR: SETCURSORREL*****Purpose**

Set the cursor to a new x, y location using relative user units.

This operation defines the starting point for a new subpath and the starting location for painted text.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The cursor is set to a valid x, y point (the *NewPath* operator causes the current cursor to be invalid).

Attribute List Specification

singleAttributePair ::= { Point }

Attribute ID	Description and {value}
Point	The x, y distance (in user units) from the current cursor at which the new cursor will be relocated. { x, y: xyValue }

Postcondition

The cursor has been set to the new x, y location relative to the current cursor (e.g. currentX+X, currentY+Y).

Example

```
sint16_xy 100 -86 Point
SetCursor
```

Comments

An error is raised if the current cursor is undefined.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

5.3 *Setting Color Space and Paint Sources*

A paint source defines the color or graphical pattern used to paint a vector or text object. A paint source associated with a pen is the source of a color object or raster pattern for a stroking operation. A paint source associated with a brush is the source of a color object or raster pattern for a filling operation.

Color objects are conceptually a single color in a specific color space (i.e. RGB, Gray, etc.). In a PCL XL device, color objects are single values or an ordered set of values that map to a specific color in the current color space. These values may be associated with a pen or brush to define the color with which a graphical object is painted.

The components of a color object must be compatible with the active color space. For example, a color object intended for use in an RGB color space must contain three ordered components, each representing a red, green, and blue intensity value. In an RGB color space, a color object with the ordered values 0, 0.9, 0 would produce green when used as a paint source. Color objects may represent gray-scaled and mono-toned colors.

A user-defined mapping into a color space may also be constructed using palettes.

Graphic State

Setting Color Space and Paint Sources

OPERATOR: SETCOLORSPACE**Purpose**

Set the current color space for the pen source, brush source, raster patterns, and images.

If a palette is specified it may be used as an index into the color space.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

```
multiAttributeList ::= { ColorSpace | deviceIndependentColorSpace } | { PaletteData
    & PaletteDepth & { ColorSpace | deviceIndependentColorSpace }
```

```
deviceIndependentColorSpace ::= { crgbSpace }
```

```
crgbSpace ::= { ColorimetricColorSpace & XYChromaticities &
    WhitePointReference & CRGBMinMax & GammaGain }
```

Attribute ID	Description and {value}
ColorSpace	The desired color space. { eRGB eGray eSRGB }
PaletteDepth	An enumerated value specifying the bits-per-component color depth of the palette data. { e8Bit }
PaletteData	The palette data used to index the color space. The length of the palette array is the maximum range of values implied by the ColorDepth attribute of BeginImage or BeginRastPattern multiplied by the number of components in the color space. For example, a raster image with a “e1Bit” ColorDepth per pixel in “eRGB” space must have a palette of 2 (depth) times 3 (number of components) for a total of 6 palette entries. { +integerArray }

Graphic State**Setting Color Space and Paint Sources**

ColorimetricColorSpace	Colorimetric RGB color spaces are based on the 1931 standard 2-degree observer and specified by CIE xy chromaticity coordinates. They use the standard D6500 viewing illuminate and a 45-degree illumination model with a 0-degree collector geometry for reflective data. {eCRGB}
XYChromaticities	A real array specifying the xy chromaticity for Luminance-Chrominance Spaces and RGB color spaces (a 2x3 Matrix where each x+y must be less than or equal to 1) {realArray}
WhitePointReference	Two reals specifying the white chromaticity for Luminance-Chrominance Spaces and RGB color spaces {real,real}
CRGBMinMax	A float array specifying the min. max. pair for RGB color space (a 2x3 Matrix where max must be greater than or equal min). {realArray}
GammaGain	A float array specifying the gamma gain for Luminance-Chrominance Spaces and RGB color spaces. 2x3 Matrix {realArray}

Postcondition

The current color space has been set.

The pen paint source and the brush paint source have been set to default values that will paint black for the corresponding color space.

Example

Setting a simple gray scale color space

```
ubyte eGray ColorSpace
SetColorSpace
```

Setting a color space that will use sRGB colors.

```
ubyte eSRGB ColorSpace
SetColorSpace
```

Setting a bi-level color space using a two entry palette

```
ubyte eGray ColorSpace
```

Graphic State**Setting Color Space and Paint Sources**

```

ubyte e8Bit PaletteDepth
ubyte_array [ 0 255 ] PaletteData
SetColorSpace

```

Setting a Colormetric color space

```

ubyte eCRGB ColorimetricColorSpace
real32_array [ 0.25 0.75 0.50 0.50 0.0 1.0 ] XYChromaticities
real32_xy 1.75 2.15 WhitePointReference
Real32_array [0.0 1.0 0.0 1.0 0.0 1.0 ] CRGBMinMax
Real32_array [1.40 1.0 1.38 1.0 1.22 1.0 ] GammaGain
SetColorSpace

```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

Data Order for the following attribute:

XYChromaticities: Values for

Red : x,y

Green : x,y

Blue : x,y

CRGBMinMax

Red : min, max

Green : min, max

Blue : min, max

GammaGain

Red : gamma, gain

Green : gamma, gain

Blue : gamma, gain

WhitePointReference

White: x,y

Graphic State**Setting Color Space and Paint Sources****OPERATOR: SETBRUSHSOURCE****Purpose**

Set the paint source for the current brush. Brushes are used to fill vector, text and raster objects.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The paint source identified in the attribute list is compatible with the current color space.

If an array of color components is specified for the brush color, the number of components in the array must exactly match the number of components expected for the color space.

If a pattern is being selected for the brush, the current color space must match the raster pattern data. If the pattern pixel colors are indexed to a palette, the palette must exist in the color space and the palette length must be compatible with the depth of the pattern source pixels.

Attribute List Specification

singleAttributePair ::=

{ RGBColor | GrayLevel | {PrimaryArray & PrimaryDepth} | NullBrush |
{ PatternSelectID & {PatternOrigin}_{opt} & {NewDestinationSize}_{opt} } }

Attribute ID	Description and {value}
RGBColor	RGB color values used to set a paint source are each expressed as an intensity level, zero being the lowest intensity. If the RGB color values are real numbers they must be in the range of 0.0 through 1.0. If the RGB colors are integer values the range allowed is the range of the integer data type. { red, green, blue: number array }
GrayLevel	The level of gray for the paint source is expressed as a intensity level, zero being the lowest intensity. A value of zero is black. The highest value in the range of the number is white. If the gray value is a real number it must be in the range of zero through 1.0. If the gray level is an integer value the range allowed between 0 and 255. { number }

Graphic State	Setting Color Space and Paint Sources
----------------------	--

PatternSelectID	A value used to identify which pattern is to be associated with the brush (see section on raster patterns for more information). { integer }
PatternOrigin	The origin for the pattern tiling process in user units. If this attribute is not supplied, the origin for the pattern is x=0, y=0 in current user units. { xyValue }
NewDestinationSize	The size of the destination box on the page for the raster pattern specified in user units. When this attribute is supplied, it replaces the original destination size specified as an attribute in the BeginRastPattern operator. The destination size is only altered for this use of the pattern. If this optional attribute is not supplied as part of SetBrushSource, the destination size for pattern will be the original destination size as specified in BeginRastPattern. Neither the x nor y value of the destination size may be zero. { xyValue }
PrimaryDepth	An enumerated value specifying the bits-per-component color depth of each element in the primary array. { e8Bit }
PrimaryArray	An array to containing the primary components used to set the paint source. The number of primary components must match the number expected for the current color space (e.g. if the color space is RGB then three and only three elements must be in the array). {Number} Range dependents upon the PrimaryDepth
NullBrush	A value indicating the brush is not to be used in painting. {0}

Postcondition

The paint source for the current brush has been set.

Example

Setting a simple gray brush

```
sint16 196 GrayLevel
SetBrushSource
```

Setting a NULL brush to indicate that the brush not be used for painting.

Graphic State**Setting Color Space and Paint Sources**

```
ubyte 0 NullBrush  
SetBrushSource
```

Setting the brush color to a pale blue.

```
ubyte e8Bit PrimaryDepth  
ubyte_array [ 60 121 204 ] PrimaryArray  
SetBrushSource
```

Setting the brush to a previously loaded pattern (See BeginRasterPattern).

```
sint16 8 PatternSelectID  
SetBrushSource
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

Brushes may be set with direct color data when a palette is set in the color space as long as the basic color space (RGB, Gray, etc.) matches the direct color data.

Graphic State**Setting Color Space and Paint Sources*****OPERATOR: SETPENSOURCE*****Purpose**

Set the paint source for the current pen.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The paint source identified in the attribute list is compatible with the current color space.

If an array of color components is specified for the pen color, the number of components in the array must exactly match the number of components expected for the color space.

If a pattern is being selected for the pen, the current color space must match the raster pattern data. If the pattern pixel colors are indexed to a palette, the palette must exist in the color space and the palette length must be compatible with the depth of the pattern source pixels.

Attribute List Specification

singleAttributePair ::=

{ RGBColor | GrayLevel | {PrimaryArray & PrimaryDepth} | NullPen |
{ PatternSelectID & {PatternOrigin}_{opt} & {NewDestinationSize}_{opt} } }

Attribute ID	Description and {value}
RGBColor	RGB color values used to set a paint source are each expressed as an intensity level, zero being the lowest intensity. If the RGB color values are real numbers they must be in the range of zero through 1.0. If the RGB colors are integer values the range allowed is the range of the integer data type. { red, green, blue: number array }
GrayLevel	The level of gray for the paint source is expressed as a intensity level, zero being the lowest intensity. A value of zero is black. The highest value in the range of the number is white. If the gray value is a real number it must be in the range of zero through 1.0 If the gray level is an integer value the range allowed between 0 and 255. { number }
PatternSelectID	A value used to identify which pattern is to be associated with the pen (see section on raster patterns for more information). { +integer }

Graphic State**Setting Color Space and Paint Sources**

PatternOrigin	The origin for the pattern tiling process in user units. If this attribute is not supplied, the origin for the pattern is x=0, y=0 in current user units. { xyValue }
NewDestinationSize	The size of the destination box on the page for the raster pattern <u>specified in user units</u> . When this attribute is supplied, it replaces the original destination size specified as an attribute in the BeginRastPattern operator. The destination size is only altered for this use of the pattern. If this optional attribute is not supplied as part of SetPenSource, the destination size for pattern will be the original destination size as specified in BeginRastPattern. Neither the x nor y value of the destination size may be zero. { xyValue }
PrimaryDepth	An enumerated value specifying the bits-per-component color depth of each element in the primary array. { e8Bit }
PrimaryArray	An array to containing the primary components used to set the paint source. The number of primary components must match the number expected for the current color space (e.g. if the color space is RGB then three and only three elements must be in the array). { Number } Range dependents upon the PrimaryDepth
NullPen	A value indicating the pen is not to be used in painting. { 0 }

Postcondition

The paint source for the current pen has been set.

Example

Setting a simple gray pen

```
sint16 196 GrayLevel
SetPenSource
```

Setting a NULL pen to indicate that the pen not be used for drawing.

```
ubyte 0 NullPen
SetPenSource
```

Setting the pen color to a pale blue.

```
ubyte e8Bit PrimaryDepth
ubyte_array [ 60 121 204 ] PrimaryArray
```

Graphic State**Setting Color Space and Paint Sources**`SetPenSource`

Setting the brush to a previously loaded pattern (See `BeginRasterPattern`).

```
sint16 8 PatternSelectID
SetPenSource
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

Pens may be set with direct color data when a palette is set in the color space as long as the basic color space (RGB, Gray, etc.) matches the direct color data.

See also: `BeginRastPattern`

graphics state**Setting Font and Character Attributes**

5.4 *Setting Font and Character Attributes*

This section describes operators that set font and character attributes.

graphics state**Setting Font and Character Attributes*****OPERATOR: SETCHARANGLE*****Purpose**

Set the angle in degrees at which individual characters should be rotated relative to the current page CTM.

The angle setting is not cumulative, as is the case with page CTM manipulation operators (i.e. SetPageRotation, SetPageOrigin, etc.). Two successive settings of SetCharAngle with 30 degrees rotate all painted characters 30 degrees, not 60 degrees relative to the current page CTM.

This operator has an effect only on supported outline fonts (i.e. non-bitmap fonts). That is, bitmap fonts will not be rotated.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { CharAngle }

Attribute ID	Description and {value}
CharAngle	The angle in degrees at which a character should be rotated when the character is painted. This value is added to the current page CTM at painting time. { number } Range 0 through 360, 0 through -360

Postcondition

The angle at which individual characters should be rotated has been set in the graphics state.

The current page CTM is unaffected.

Example

```
sint16 90 CharAngle
SetCharAngle
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Font and Character Attributes****OPERATOR: SETCHARSCALE****Purpose**

Set the factor at which characters should be scaled relative to the current page CTM.

The scale factor setting is not cumulative as is the case with page CTM manipulation operators (i.e. SetPageScale, SetPageOrigin, etc.) . Two successive settings of SetCharScale with x=2 scales in the x direction by 2 units, not 4 units relative to the current page CTM.

This operator has an effect only on supported outline fonts (i.e. non-bitmap fonts). That is, bitmap fonts will not be scaled.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { CharScale }

Attribute ID	Description and {value}
CharScale	The scaling factor for scaling each character relative to the current page CTM. { xScale, yScale: xyValue }

Postcondition

The factor at which characters should be scaled has been set in the graphics state.

The current page CTM is unaffected.

Example

```
real32_xy 1.333 1.0 CharScale
SetCharScale
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Font and Character Attributes****OPERATOR:** *SETCHARSHEAR***Purpose**

Set the factor at which characters should be sheared relative to the current page CTM.

The shear factor setting is not cumulative as is the case with page CTM manipulation operators (i.e. SetPageScale, SetPageOrigin, etc.) . Two successive settings of SetCharShear with x=2 shears horizontally 2 units, not 4 units relative to the current page CTM.

This operator has an effect only on supported outline fonts (i.e. TrueType). That is, bitmap fonts will not be sheared.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { CharShear }

Attribute ID	Description and {value}
CharShear	The factor for shearing each character relative to the current page CTM. This is expressed in two linear coordinates: x for horizontal shear and y is for vertical shear. These values are in user units. { xShear, yShear: xyValue }

Postcondition

The factor at which characters should be sheared has been set in the graphics state.

The current page CTM is unaffected.

Example

```
sint16_xy 10 5 CharShear
SetCharShear
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

OPERATOR: SETCHARBOLDVALUE**Purpose**

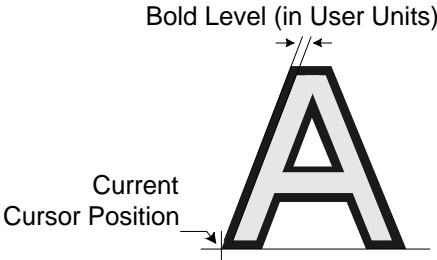
Set the percent of boldness to be *added* to characters during the text painting process. This operator is provided to achieve a pseudo-bold effect for characters of a font when a bold version of that font is unavailable. The bold effect provided by this operator is not typographically correct and may not preserve the artistic integrity of the character.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { CharBoldValue }

Attribute ID	Description and {value}
CharBoldValue	<p>The bold value is a real number between 0.0 and 1.0. This value is multiplied by the character height to obtain the number of user units by which each character should be bolded (e.g. a 0.2 bold value multiplied by a 20 user-unit character height results in a 4 user unit bold level). The bold level specifies how much the character should be bolded on the inside and the outside of the character outline. In the figure below, the gray shaded area represents the original character and the black area represents the result of the bolding process.</p> <div style="text-align: center;">  </div> <p>{+number zero through 1 }</p>

Postcondition

The percent of character pseudo-bolding has changed in the graphics state.

Example

```
real32 0.25 CharBoldValue
SetCharBold
```


graphics state**Setting Font and Character Attributes****Comments**

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Font and Character Attributes****OPERATOR: SETCHARSUBMODE****Purpose**

Set the type of character substitutions to be performed during the text painting process.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The modes specified in the substitution mode array must not conflict and may not be duplicated.

Attribute List Specification

singleAttributePair ::= { CharSubModeArray }

Attribute ID	Description and {value}
CharSubModeArray	An array of enumerations specifying character substitution modes. If vertical substitution is specified for a font without vertical substitution characters defined, the attribute is ignored. {eNoSubstitution eVerticalSubstitution} ₁₊

Postcondition

The character substitution mode for painting the characters of a font has been set in the graphics state.

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Font and Character Attributes****OPERATOR: SETCHARATTRIBUTES****Purpose**

Set the current character attributes. Set the current writing mode of the printer.

This operator has an effect only on supported outline fonts (i.e. non-bitmap). That is, bitmap fonts will not be effected.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { WritingMode }

Attribute ID	Description and {value}
WritingMode	The current writing mode. { eVertical eHorizontal }

Postcondition

The writing mode has been specified in the graphics state.

The current page CTM is unaffected.

Example

```
ubyte eHorizontal WritingMode
SetCharAttributes
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state

Setting Font and Character Attributes

OPERATOR: SETFONT**Purpose**

Set the current font from which characters will be selected by character codes in text operators.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The font is already defined in the device.

Note: If the requested font does not exist, PCL XL will substitute a font from those existing in the device at the time of the request. A warning will be generated due to the font substitution that may optionally be reported to the user. The exact font substitution algorithm is device- and language-region dependent. See the PCL XL addendum for the target device for more details concerning the font substitution algorithms.

Attribute List Specification

multiAttributeList ::= { SymbolSet & CharSize & FontName }

Attribute ID	Description and {value}
FontName	The name of the font to be selected (i.e. "Times New Roman Bold"). { +integer array }
CharSize	The size of the characters to be rendered for the font in user units. This value is ignored for characters downloaded as bitmaps. The size of bitmap characters are the size of their original download definition. { +number greater than zero }
SymbolSet	The identifier of the symbol set to use for the current font (See Appendix O for values.) { +integer }

Postcondition

The current font has been set in the graphics state.

Example

```
ubyte_array (TimesNewRmn) FontName
real32 100 CharSize
uint16 629 SymbolSet
SetFont
```

Comments

graphics state**Setting Font and Character Attributes**

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Attributes of the Current Path**

5.5 Setting Attributes of the Current Path

This section describes the operators used to set the attributes for painting the current path.

Painting the path consists of filling the geometric region defined by the interior of the path outline with the current brush. Filling only occurs when the current brush is defined. The current clip path will clip the filled area.

If the current Pen is defined, the path will also be stroked. Stroking the path will fill an area along the edge of the path that is $\frac{1}{2}$ the PenWidth outside the edge and $\frac{1}{2}$ the PenWidth inside the path. This area will have line joins applied to connect joined line segments and have line ends applied to all ending line segments. The current clip path will clip the resulting edge segments.

OPERATOR: SETFILLMODE**Purpose**

Set the current mode for filling path objects when a brush is applied to the object at painting time.

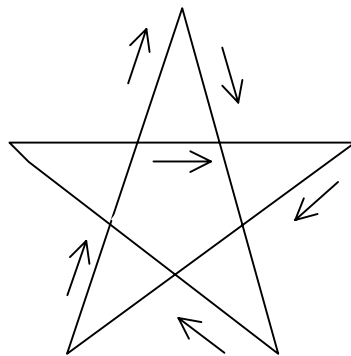
All automatically closed path objects (i.e. Ellipse, Circle, Pie, Chord, and Rectangle) are drawn counterclockwise by PCL XL. All open path objects are drawn in the direction that points are placed into the path.

The FillMode determines the “insideness” of vector objects.

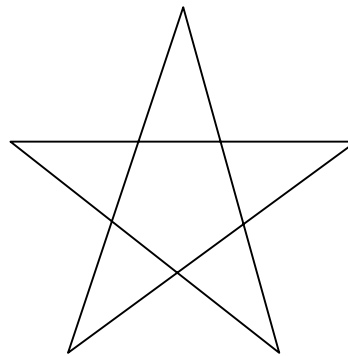
Even-Odd Fill Rule (or Alternate Fill Rule): a point is considered inside the path if a line that is drawn from the point in question to infinity in any direction results in an odd number of intersections with the path. Filling occurs between odd-numbered and even-numbered polygon edges (e.g. edge1-edge2, edge3-edge4, etc.).

Non-zero Winding Rule: a point is considered inside the path if a line that is drawn from the point in question to infinity in any direction results in an intercession count that is non-zero. The count is determined by starting at 0 and incrementing it when an edge is crossed that is ascending (relative to the coordinate system) and decrementing it when an edge is crossed that is descending. See illustration below (note: arrows indicate direction in which the polygon was drawn):

Non-Zero Winding Fill



Even Odd Fill

**Precondition**

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { FillMode }

Attribute ID	Description and {value}
--------------	-------------------------

graphics state**Setting Attributes of the Current Path**

FillMode

The mode determining the nature of filling (brush) operations for closed path objects.
{ eNonZeroWinding | eEvenOdd }

Postcondition

The current fill mode has been set in the graphics state.

Example

```
ubyte eEvenOdd FillMode  
SetFillMode
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

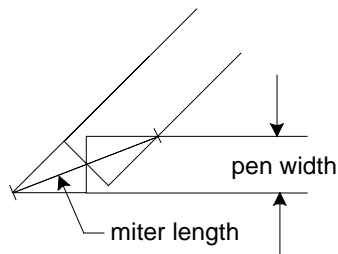
See **Appendix K** for related error codes.

graphics state

Setting Attributes of the Current Path

OPERATOR: *SETMITERLIMIT***Purpose**

Set a limit on the length of the miter shape to be drawn where any two consecutive line segments in a subpath meet at an angle. The miter limit determines the maximum length of a mitered join. The miter limit is a ratio of the length of an imaginary line drawn through the join to the pen width (see illustration below.)



$$\text{Miter Limit} = \frac{\text{miter length}}{\text{pen width}}$$

If the miter shape exceeds this limit for a particular join the device will apply a bevel shape to that join.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { MiterLength }

Attribute ID	Description and {value}
MiterLength	The length limit of the miter shape expressed as a ratio of the miter length to the pen width. { +number }

Postcondition

The current miter limit has been set in the graphics state.

Example

```
sint16 20 MiterLength
SetMiterLimit
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

graphics state

Setting Attributes of the Current Path

See **Appendix K** for related error codes.

graphics state**Setting Attributes of the Current Path****OPERATOR:** *SETLINECAP***Purpose**

Set the shape that should be drawn at the end of lines when stroking paths. Paths that are explicitly closed do not have ends and thus do not get capped. The size and shape of the line cap is determined at stroke time and is influenced by the line width and transformations in effect.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { LineCapStyle }

Attribute ID	Description and {value}
LineCapStyle	The line end style. { eRoundCap eTriangleCap eSquareCap eButtCap }

Postcondition

The current line-end shape has been set in the graphics state.

Example

```
ubyte eTriangleCap LineCapStyle
SetLineCap
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Attributes of the Current Path****OPERATOR: SETLINEJOIN****Purpose**

Set the shape to be applied when joining any two connected line segments. . The size and shape of the line join is determined at stroke time and is influenced by the line width and transformations in effect.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { LineJoinStyle }

Attribute ID	Description and {value}
LineJoinStyle	An enumerated value representing the line join style. { eMiterJoin eRoundJoin eBevelJoin eNoJoin }

Postcondition

The current line join shape has been set in the graphics state.

Example

```
ubyte eBevelJoin LineJoinStyle
SetLineJoin
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

See also: SetMiterLimit

graphics state**Setting Attributes of the Current Path****OPERATOR: SETLINEDASH****Purpose**

Set the repeating dash style with which a subpath should be stroked. The user configures a line dash by either specifying a solid line or by specifying a repeating sequence of user-unit segments designating on and off segment lengths.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList ::= { {DashOffset}_{opt} & LineDashStyle } | { SolidLine }

Attribute ID	Description and {value}
LineDashStyle	<p>An array describing the dash style. The first element of the array specifies the user unit dash length to be stroked. The next is the user unit dash length to be skipped. The next is the number of units to be stroked, alternating in this fashion. If the subpath to be stroked is longer in user-units than the dash array, the dash style will repeat. For the repeat effect, consider the elements in the original dash array to be appended to the original array in a successive manner to satisfy the length of the subpath. If the original dash array had an odd length of elements, the first element of the first repeat will be the number of units <i>not stroked</i>, continuing with the <i>stroked, not stroked</i> pattern.</p> <p>Each Subpath when stroked will begin at the start of the dash array.</p> <p>LineDashStyle is limited to 20 segments.</p> <p>{ unitsStroked, unitsNotStroked, unitsStroked... :numberArray }</p>
DashOffset	<p>A value indicating an offset into the dash style array from which the dash style will begin.</p> <p>{ +number }</p>
SolidLine	<p>Specifying this attribute sets the line style to solid.</p> <p>{ 0 }</p>

Postcondition

The current pen dash style has been set in the graphics state with dash style and dash offset fixed to the current user units.

graphics state**Setting Attributes of the Current Path****Example**

Setting the line drawing style to a solid line

```
ubyte 0 SolidLine  
SetLineDash
```

Setting the line drawing style to a dashed line where 100 pixels will be drawn, 10 skipped, and the pattern repeated.

```
sint16_array [100 10 ] LineDashStyle  
SetLineDash
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

Setting the LineDashStyle array to all zeros, or setting any member of the LineDashStyle array to a negative number will generate an error page. See **Appendix K** for related error codes.

Since the dash style and offset are fixed according to the current user units at **SetLineDash** time, the dash style of a line is not scaled when a line is scaled.

graphics state**Setting Attributes of the Current Path*****OPERATOR: SETPENWIDTH*****Purpose**

Set the geometric width in user units of the Pen. The Pen is used to stroke the current path. The Line Cap and Line Join are also scaled proportionally with the Pen thickness.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { PenWidth }

Attribute ID	Description and {value}
PenWidth	A value specifying the pen width in user units. { number }

Postcondition

The current pen width has been set in the graphics state.

Example

```
ubyte 10 PenWidth
SetPenWidth
```

Comments

Unlike most other PCL XL operations, the PenWidth operator is not transformed to device space immediately. The transformation of the PenWidth occurs at stroke time and thus becomes transformed by the CTM in effect at the time of stroking.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

5.6 *Setting the Current Clip Path*

Clip paths allow the application or driver to constrain areas in which marks may be placed on a page for painting path or bitmap objects. All painting operations may be confined to the interior or exterior region(s) defined by the current clip path depending on the clip mode (see SetClipMode).

The default clip path for a page is the imagable area of the page. The default clip path is always constructed in a clockwise direction.

The following are key concepts behind clip paths:

- ◆ There is always an active clip path in the device
- ◆ The initial clip path is set to the imagable area of the page
- ◆ Any region defined by a path may be used to define the current clip path
- ◆ The current clip path may be saved and restored at any time during a painting session
- ◆ The current clip mode has an effect on construction of the clip path
- ◆ Clip paths may be inclusive or exclusive (show the interior or exterior)
- ◆ The current path at the graphic state top is destroyed when setting a new clip path
- ◆ The clip path must prevent imaging off the logical page

graphics state**Setting the Current Clip Path*****OPERATOR: SETCLIPREPLACE*****Purpose**

Replace the clip path with the intersection of the default clip path and the current path. The inside definition is determined by the ClipRegion enumeration. The inside definition is determined by the ClipRegion enumeration.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

If the ClipRegion is “eExterior” the current graphics state ClipMode must be “eEvenOdd.”

Attribute List Specification

singleAttributePair ::= { ClipRegion }

Attribute ID	Description and {value}
ClipRegion	An enumeration specifying the region that should be exposed by the new clip path during painting. This will either be the region inside the clip path outside the clip path. { eInterior eExterior }

Postcondition

The clip path has been set to the intersection of the current path and default clip path.

If there is no current path before the operator is executed, the clip path becomes empty. Thus all objects will be clipped until the next SetClipToPage operation.

The current path has been destroyed.

Example

```
ubyte eExterior ClipRegion
SetClipReplace
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state

Setting the Current Clip Path

OPERATOR: *SETCLIPINTERSECT***Purpose**

Replace the current clip path with the intersection of the current path and the current clip path. The inside definition is determined by the ClipRegion enumeration.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

If the ClipRegion is “eExterior” the current graphics state ClipMode must be “eEvenOdd.”

Attribute List Specification

singleAttributePair ::= { ClipRegion }

Attribute ID	Description and {value}
ClipRegion	An enumeration specifying the region that should be exposed by the new clip path during painting. This will either be the region inside the clip path intersection or outside the clip path intersection. { eInterior eExterior }

Postcondition

The clip path has been set to the intersection of the current path and the clip path that existed prior to operator execution.

If there is no current path before the operator is executed, the clip path becomes an empty path. Thus all future painting will be blocked across the entire page until a new clip path is set.

The current path has been destroyed.

Example

```
ubyte eExterior ClipRegion
SetClipIntersect
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state

Setting the Current Clip Path

OPERATOR: *SETCLIPRECTANGLE***Purpose**

Replace the current clip path with the intersection of the rectangular region supplied and the current clip path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

If the ClipRegion is “eExterior” the current graphics state ClipMode must be “eEvenOdd.”

Attribute List Specification

singleAttributePair := { ClipRegion & BoundingBox }

Attribute ID	Description and {value}
ClipRegion	An enumeration specifying the region that should be exposed during painting. This will either be the region inside bounding box or the entire region outside the bounding box. { eInterior eExterior }
BoundingBox	The points describing the bounding box for the rectangle to be used as the clipping region boundary. { x1, y1, x2, y2: boxValue }

Postcondition

The current path has been destroyed.

The current clip path has been replaced with intersection of the current clip path and the rectangular region.

Example

```
ubyte eInterior ClipRegion
sint16_box 0 0 1200 1200 BoundingBox
SetClipRectangle
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state

Setting the Current Clip Path*OPERATOR: SETCLIPTOPAGE***Purpose**

Reset the clip path to be the limits of the imageable area on the target page.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The clip path has been set to the limits of the imageable area on the page.

The current path has been destroyed.

Example

```
SetClipToPage
```

Comments

See **Appendix K** for related error codes.

graphics state

Setting the Current Clip Path

OPERATOR: *SETPATHTOCLIP***Purpose**

Replace the current path with a path representing the area defined by the current clip path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

There exists a valid clip path.

Attribute List Specification

nullAttributeList

Postcondition

The current path has been replaced with the current clip path.

The current clip path has been preserved.

Example

```
SetClipToPath
```

Comments

Note: The area of a clipping region defined by PCL XL operators is required to be consistent from device-to-device. The *number* and *variety* of lines and Beziers used internal to the device to construct a clipping region are not required to be consistent. Thus, filling a path created by SetPathToClip with a brush is *device-independent*. However, stroking such a path with a pen is *device-dependent* and should be avoided.

See **Appendix K** for related error codes.

graphics state**Setting the Current Clip Path****OPERATOR:** *SETCLIPMODE***Purpose**

Set the mode in which the clip path should be constructed during SetClipIntersect and SetClipRectangle.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { ClipMode }

Attribute ID	Description and {value}
ClipMode	<p>An enumeration specifying the insideness of the clip path. The insideness is determined geometrically by the same rules as specified in SetFillMode.</p> <p>Note: Only “eEvenOdd” is allowed for performing exterior clipping with a clip path.</p> <p>{ eEvenOdd eNonZeroWinding }</p>

Postcondition

The clip mode has been set in the graphic state.

Example

```
ubyte eEvenOdd ClipMode
SetClipMode
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

5.7 PCL XL Logical Operations

5.7.1 Logical Operators Introduction

PCL XL logical operations determine how source images (the objects being painted) interact with destination images (images already on the page) in conjunction with the paint being applied and current transparency mode settings.

Binary raster operations (e.g., AND, OR, XOR, NOT) may be applied to the source image, the destination (page) image, and the paint with which the image is colored. In addition to binary raster operations, transparency modes may also be specified. This section describes the purpose and use of PCL XL logical operations.

5.7.2 PCL XL Raster Operations (ROP's)

Binary raster operations (ROP'S) in PCL XL involve up to three operands. These operands are the source image, the destination image, and the paint to be applied to the object.

Definitions

Source Image Operand: The pixels of the region defined by the object being painted.

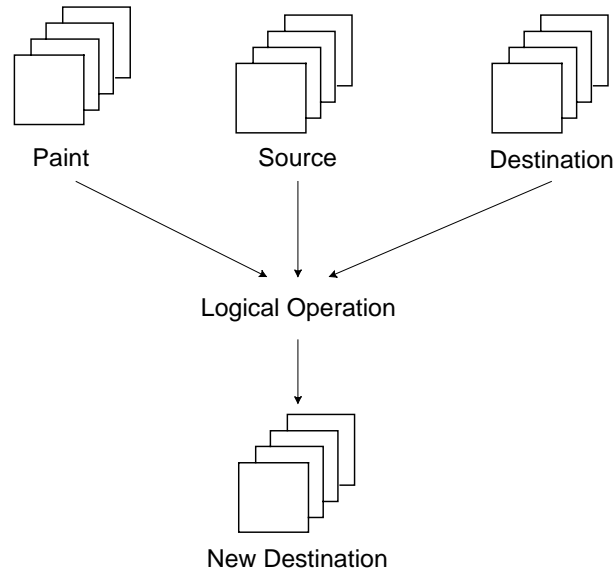
Destination Image Operand: The destination image contains whatever is currently defined on the page. It includes any paths, characters, and/or raster images painted to the page through previous operations.

Paint Operand: The paint used to color the object may be a color or a pattern. The SetPenSource and SetBrushSource operators define the current paint source for an object. The area that the paint may effect is inside the boundaries of the object being painted and inside the current clip path.

The PCL XL SetROP operator specifies the raster operation (ROP3) to be performed on destination, source and paint to produce new destination data. Assuming four-bits-per-pixel, the following diagram shows the ROP process with each overlapping box representing a bit.

graphics state

Setting Painting Process Attributes



NOTE: Raster operations are defined in this and most other documents in RGB space (white = 1, black = 0) rather than in CMY space (white = 0, black = 1). For example, ORing a white object with a black object in RGB space yields a white object. This is the same as ANDing the two objects in CMY space. All the figure examples for raster operations in the document assume RGB space (white = 1). If the internals of the device are based on a color space other than RGB, the RGB ROP3s defined by the user are translated appropriately by the device for correct internal operation.

Raster operations are specified in this document using RPN (reverse polish, or postfix notation). A character string where operands are designated as single upper-case letters and operators are designated as single lower-case letters describes ROP3 functions. For example, the ROP of value 225 corresponds to PDSoxn (Paint/Destination/Source/or/xor/not), the logical function of:

NOT (Paint XOR (Source OR Destination))

ROP Example

A ROP of 252 (PSo), corresponds to a binary operation of:

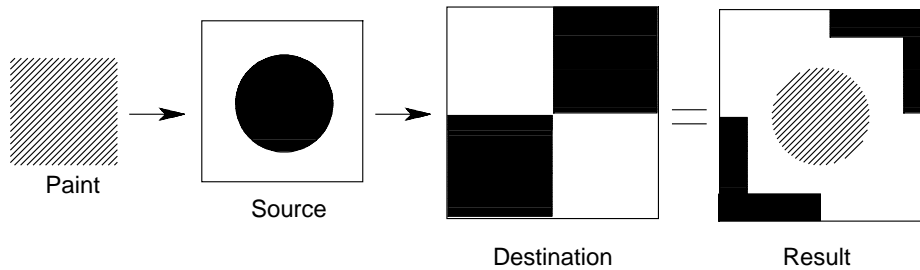
Paint OR Source

Sample operands are computed below for ROP PSo with source and paint opaque. Each column of destination, source, and paint values are the inputs to the raster operation function.

graphics state	Setting Painting Process Attributes
-----------------------	--

	Bits							
	7	6	5	4	3	2	1	0
Paint	1	1	1	1	0	0	0	0
Source	1	1	0	0	1	1	0	0
Destination	1	0	1	0	1	0	1	0
PSo ROP3 Result	1	1	1	1	1	1	0	0

A graphical example of the PSo ROP is illustrated below. Source and paint transparencies are default (white pixels are opaque). This example is in RGB space such that a white pixel is a 1 bit.



A Graphical Example of the PSo ROP

A complete table of ROP3 operations with their ROP codes and functions are listed in the appendix of this document.

graphics state**Setting Painting Process Attributes**

5.7.3 The ROP3 Operands

The following tables define the source, destination, and paint operands in terms of raster images, characters, and paths used in ROP operations.

Source Image Operand for ROP3 Operations

Painted Object	Source Operand Description
Raster Image	<p>The raster image pixels (downloaded by the user with Begin/ Read/ EndImage) make up the source image operand of the ROP. The size of the source image is the size of the destination bounding box of the downloaded raster. The raster image may have been dithered.</p> <p>If the current brush is null and the current ROP includes a paint operand, the raster image has no effect in the ROP process.</p>
Character	<p>The source operand is either the black pixels of a bitmap character or the inside regions of an outline character (a character defined with lines and curves). If the character is a bitmap character, the white pixels are transparent and not considered in the ROP process. If the character is an outline character, consider the inside regions to be filled with black pixels.</p> <p>If the current brush is null and the current ROP includes a paint operand, the character has no effect in the ROP process.</p>
Path Strokes	<p>PCL XL allows a limited subset of ROP3 codes to be used during a stroke operation. These include ROP 0, 160, 170, 240, 250 and 255. PCL XL will always substitute ROP 240 for any other ROP3 code during the stroke operation and restore the original ROP3 when complete.</p> <p>If the current pen is null and the current ROP includes a paint operand, the path will not be stroked.</p>
Path Region	<p>The “inside” regions defined by the current path and the current fill mode setting comprise the source image operand. If the current subpath was left open, it will be temporarily closed to determine the inside regions. For the ROP process, consider the inside regions of the path to be filled with black pixels.</p> <p>If the current brush is null and the current ROP includes a paint operand, the inside regions of a path have no effect during the ROP process.</p>

graphics state**Setting Painting Process Attributes****Paint Operand for ROP3 Operations**

Painted Object	Paint Operand Description
Raster Image	<p>The destination box of the raster image (downloaded by the user with Begin/Read/EndImage) defines the region in which paint may have an effect during the ROP process.</p> <p>If the current brush is null and the current ROP includes the paint operand, the raster image has no effect in the ROP process.</p>
Character	<p>The regions in which paint may effect the ROP are either the black pixels of a bitmap character or the inside regions of an outline character.</p> <p>If the current brush is null and the current ROP includes a paint operand, the character has no effect in the ROP process.</p>
Path Strokes	<p>PCL XL will always substitute a ROP3 of 240_{decimal} (Paint operand only) with a paint transparency mode of eOpaque during the path stroking process. Therefore, the paint operand is always used when a path is stroked by the current pen. Once the path stroking process completes, the original ROP3 will be restored.</p> <p>If the current pen is null, the path cannot be stroked. Thus path strokes are not part of the ROP process when the current pen is null.</p>
Path Region	<p>The “inside” regions defined by the current path and the current fill mode setting define the regions in which paint may have an effect in the ROP process. If the current subpath was left open, it will be temporarily closed to determine the “inside” regions.</p> <p>If the current brush is null and the current ROP includes a paint operand, the inside regions of a path have no effect during the ROP process.</p>

graphics state**Setting Painting Process Attributes****Destination Operand for ROP3 Operations**

Painted Object	Destination Operand Description
Raster Image	<p>The destination box of the raster image (downloaded by the user with Begin/Read/EndImage) defines the region of the destination image that may be effected by the ROP process.</p> <p>If the current brush is null and the current ROP includes the paint operand, the raster image has no effect in the ROP process.</p>
Character	<p>The regions of the destination image that may be effected by the ROP are either the black pixels of a bitmap character or the inside regions of an outline character.</p> <p>If the current brush is null and the current ROP includes a paint operand, the character has no effect in the ROP process.</p>
Path Strokes	<p>PCL XL will always substitute a ROP3 of 240_{decimal} (Paint operand only) with a paint transparency mode of eOpaque during the path stroking process. Therefore, the destination operand is never used when a path is stroked by the current pen. Once the path stroking process completes, the original ROP3 will be restored.</p> <p>If the current pen is null, the path cannot be stroked. Thus path strokes are not part of the ROP process when the current pen is null.</p>
Path Region	<p>The “inside” region defined by the current fill mode is the area in which the destination may be effected. If the current subpath was left open, it will be temporarily closed to determine the inside regions.</p> <p>If the current brush is null and the current ROP includes a paint operand, the inside regions of a path have no effect during the ROP process.</p>

5.7.4 Transparency Modes

In PCL XL the white pixels of a source image or paint may be made transparent or opaque during the painting process.

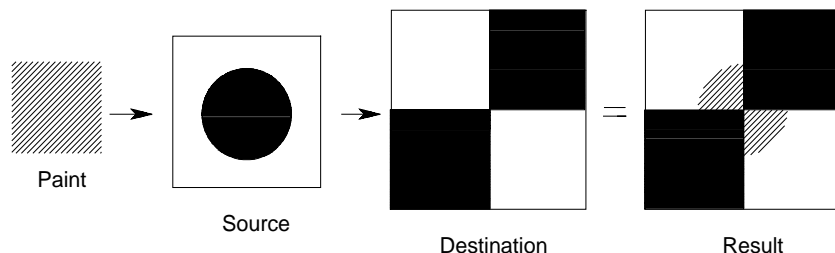
SetSourceTxMode (Set Source Transparency Mode)

A value of eTransparent in the SetSourceTxMode operator makes the source image's white pixels transparent, allowing the corresponding parts of the destination image to show through. A value of eOpaque makes the source image's white pixels opaque, blocking out the corresponding parts of the destination.

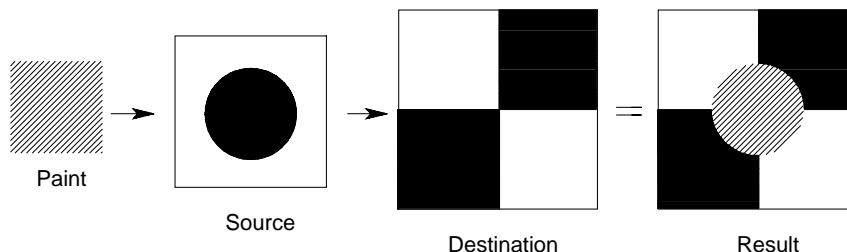
SetPaintTxMode (Set Paint Transparency Mode)

The paint's non-white pixels are poured through the source image's non-white pixels onto the destination. A value of eTransparent in the SetPaintTxMode operator makes the paint's white pixels transparent, allowing the corresponding parts of the destination image to show through. A value of eOpaque makes the paint's white pixels opaque, painting the corresponding parts of the destination white.

The following are examples of various transparency mode settings. For the examples that follow, assume the ROP setting to be a value of 252_{decimal} (Paint ORed with the Source).

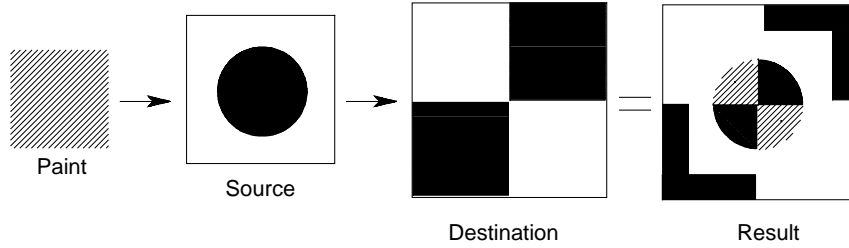


Source Transparency = eTransparent
Paint Transparency = eTransparent

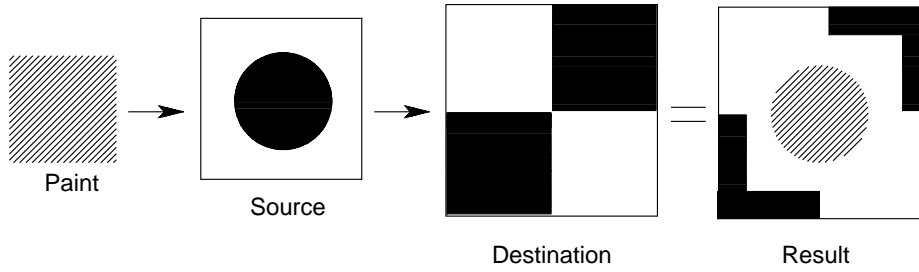


Source Transparency = eTransparent
Paint Transparency = eOpaque

graphics state	Setting Painting Process Attributes
-----------------------	--



Source Transparency = eOpaque
Paint Transparency = eTransparent



Source Transparency = eOpaque
Paint Transparency = eOpaque

5.7.5 Raster Operations and Transparency Interactions

Transparency modes operate in combination with raster operations. The raster operations in the ROP3 table listed in the appendix are true only if source image and paint transparency (for white pixels) are explicitly set to opaque. If source and/or paint transparency modes are transparent, additional operations must be performed to achieve the final result.

The four basic interactions among ROP3 and transparency mode settings are expressed below in bit operation sequences (where ‘&’ represents a bitwise AND operation and ‘|’ represents a bitwise OR operation):

- ◆ Case 1: Source and Paint are opaque.

NEW DESTINATION = ROP3 (Dest, Src, Paint)

- ◆ Case 2: Source is opaque, Paint is transparent.

Temporary_ROP3 = ROP3 (Dest, Src, Paint).
 Image_A = Temporary_ROP3 & Not Src.
 Image_B = Temporary_ROP3 & Paint.
 Image_C = Not Paint & Src & Dest.
 NEW DESTINATION = Image_A | Image_B | Image_C

- ◆ Case 3: Source is transparent, Paint is opaque.

Temporary_ROP3 = ROP3 (Dest, Src, Paint).
 Image_A = Temporary_ROP3 & Src.
 Image_B = Dest & Not Src.
 NEW DESTINATION = Image_A | Image_B

- ◆ Case 4: Source and Paint are transparent

Temporary_ROP3 = ROP3 (Dest, Src, Paint).
 Image_A = Temporary_ROP3 & Src & Paint.
 Image_B = Dest & Not Src.
 Image_C = Dest & Not Paint.
 NEW DESTINATION = Image_A | Image_B | Image_C.

graphics state**Setting Painting Process Attributes**

The result computed in the example below is similar to a previous example except that two cases are shown: case 1 (source and paint opaque) and case 4 (source and paint transparent). Note that the expected PSo ROP3 results only with case 1, when both source and paint transparency modes are set to opaque.

	Bits							
	7	6	5	4	3	2	1	0
Paint	1	1	1	1	0	0	0	0
Source	1	1	0	0	1	1	0	0
Destination	1	0	1	0	1	0	1	0
PSo ROP3 (source & paint are opaque)	1	1	1	1	1	1	0	0
PSo ROP3+Transparencies (source & paint are transparent)	1	1	1	0	1	0	1	0

graphics state**Setting Painting Process Attributes**

5.7.6 ROP2 Equivalents in the ROP3 Set

The ROP3 codes found in the appendix of this document have ROP2 equivalents. The table below specifies these equivalents. The names used for the ROP2 values in the table below are the common ROP2 names used in Microsoft Windows 3.1 and Win32 Programmers Reference Manuals. The ROP2 and ROP3 raster functions shown below apply to RGB space (white=1, black=0).

ROP2 Name	ROP2 Function	ROP2 Value	ROP3 Value
R2_BLACK	0	1	0
R2_NOTMERGEPEN	DPon	2	5
R2_MASKNOTOPEN	DPna	3	10
R2_NOTCOPYPEN	Pn	4	15
R2_MASKPENNOT	PDna	5	80
R2_NOT	Dn	6	85
R2_XORPEN	DPx	7	90
R2_NOTMASKPEN	DPan	8	95
R2_MASKPEN	DPa	9	160
R2_NOTXORPEN	DPxn	10	165
R2_NOP	D	11	170
R2_MERGENOTPEN	DPno	12	175
R2_COPYPEN	P	13	240
R2_MERGEPENNOT	DPno	14	245
R2_MERGEPEN	DPo	15	250
R2_WHITE	1	16	255

graphics state**Setting Painting Process Attributes****OPERATOR: SETPAINTTXMODE****Purpose**

Set the effect that the paint's white pixels will have on the destination image. Either the white paint pixels are transparent on the destination image page or they appear white on the destination image.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { TxMode }

Attribute ID	Description and {value}
TxMode	The transparency mode value for the paint. { eOpaque eTransparent }

Postcondition

The paint transparency mode has been set in the graphics state.

Example

```
ubyte eOpaque TxMode
SetPaintTxMode
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Painting Process Attributes****OPERATOR: SETSOURCEtxMODE****Purpose**

Set the effect that a source object's (painted object's) white pixels will have on the destination image. Either the white source pixels are transparent on the destination image or they appear white on the destination image.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { TxMode }

Attribute ID	Description and {value}
TxMode	The transparency mode value for the source object. { eOpaque eTransparent }

Postcondition

The source transparency mode has been set in the graphics state.

Example

```
ubyte eTransparent TxMode
SetSourceTxMode
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state**Setting Painting Process Attributes*****OPERATOR: SETROP*****Purpose**

Set the current raster operation to be applied to the paint source, graphical object, and destination image.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { ROP3 }

Attribute ID	Description and {value}
ROP3	The value representing the desired ROP3 operation. { +integer }

Postcondition

The current ROP has been set in the graphics state.

Example

```
ubyte 252 ROP3
SetROP
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

graphics state	Setting Painting Process Attributes
-----------------------	--

5.8 SETTING AND USING HALFTONE METHODS

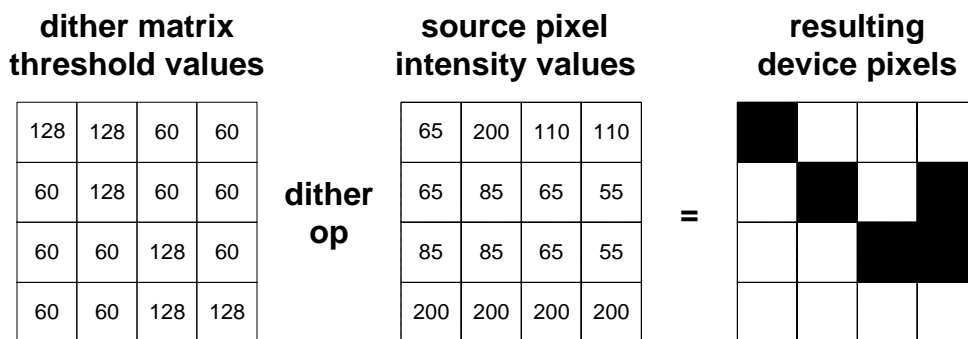
PCL XL allows the user to define methods for the halftoning process. The current version of PCL XL allows the user to define a dither matrix for the halftone method.

The dither matrix is a device-dependent method of instructing the PCL XL device how to color device pixels based on the intensity value of each colored source pixel of a path, text, or raster object. The dither matrix itself is an N by M matrix of pixel intensity threshold values. Each cell in a dither matrix contains a threshold value from zero through 255 inclusive. The dither matrix is tiled across the page such that all device pixels correspond to a dither cell.

Regardless of the color space, the user’s view of the dithering process is that zero is the lowest intensity and 255 is the highest intensity for a color. The device handles color space translation internally.

Note. On some PCL XL devices, dithering may not be desirable. On these devices it is acceptable to read and ignore the dither matrix data without causing an error condition. In these case, the user-defined dither matrix will have no effect.

If the intensity value of a colored path, text, or raster source pixel is less than the corresponding dither cell intensity value, the device pixel representing that source pixel is set to the lowest intensity value possible (black on a monochrome device). If the intensity value of a source pixel is equal to or greater than the intensity value in the corresponding dither matrix cell, the corresponding device pixel is set to the highest intensity value possible (white on a monochrome device). A “zero” entry in the dither matrix behaves like a “one” entry such that a zero intensity source pixel will cause the corresponding device pixel to be set to zero intensity by the dither operation. The following shows an example dither operation on 16 sample source pixels (assume a monochrome device).



If the device is a multi-plane device, a single dither cell value is applied to pixels on each plane corresponding to that dither cell’s x, y locations on the physical page. The origin of the dither matrix tiling process may be set by the user in current user units.

graphics state**Setting Painting Process Attributes**

The bottom row of the dither matrix is parallel to the bottom row of the physical page based on BeginPage orientation. Therefore the bottom row of a dither matrix on a portrait page is parallel to the bottom short edge of the page. The bottom row of a dither matrix on a landscape page is parallel to the bottom long edge of the page.

If the source pixel intensity is bi-level, the dithering process is not performed.

graphics state**Setting Painting Process Attributes****OPERATOR: SETHALFTONEMETHOD****Purpose**

Set the halftone method by specifying the current dither matrix to be used in the halftoning process for scanned (bitmap) images, raster patterns, and colors by reading the matrix data from the currently open data source. This is a device resolution-dependent operator.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The length of each dither matrix row must be a multiple of four bytes. If row data is not a multiple of four bytes, row data must be padded with the appropriate number of bytes.

The depth of the dither matrix data must be eight bits.

Attribute List Specification

```
multiAttributeList ::= {DitherOrigin}opt & { { DeviceMatrix } |
  { DitherMatrixDataType & DitherMatrixSize & DitherMatrixDepth } }
```

Attribute ID	Description and {value}
DitherOrigin	The origin for the dithering process in user units. If this attribute is missing the origin is x=0, y=0 in user units. { xyValue }
DeviceMatrix	An enumeration to select internal device dither matrices. { eDeviceBest }
DitherMatrixDataType	The data type of data to be read from the data source for the new dither matrix setting. Each data element read from the data source represents one dither matrix cell value. The first byte of the data is the left-most cell of a row (no byte swapping is performed on dither matrix data). { eUByte }
DitherMatrixSize	The width (x value) and height (y value) of the dither matrix in device pixels, the values of which may each be one or greater. If the actual width is not a multiple of four bytes, the data for each row must be padded to four bytes. { xyValue }
DitherMatrixDepth	The depth of each dither matrix entry in bits. { e8Bit }

Postcondition

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 135	BLD 4600
-----------	-----------------	---------	----------	----------

graphics state**Setting Painting Process Attributes**

The current halftone method has been set in the graphics state.

Example

Setting the printer's default dither matrix

```
ubyte eDeviceBest DeviceMatrix
SetHalftoneMethod
```

Setting a downloaded dither pattern

```
uint16_xy 16 2 DitherMatrixSize
ubyte 0 DitherMatrixDataType
ubyte 2 DitherMatrixDepth
SetHalftoneMethod
```

This operator must be immediately followed by dither matrix data.

```
dataLength 32
hex_raw* [
80 00 80 00 80 00 80 00 80 00 80 00 80 00 80 00
00 80 00 80 00 80 00 80 00 80 00 80 00 80 00 80
]
```

Comments

Note. Some PCL XL implementations may choose to read and ignore the dither matrix data without causing an error condition. In this case, the user-defined dither matrix will have no effect.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path**Path Definition and Painting Operators**

5.9 *Setting Page Coordinate System Attributes*

This section contains a description of the operators used to setup and modify the page coordinate system.

path	Path Definition and Painting Operators
-------------	---

OPERATOR: *SETPAGEDFAULTCTM*

Purpose

Set the current page coordinate transformation matrix to BeginPage defaults (this includes the default orientation for the target page).

This operator does not effect the current character CTM settings.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The current page coordinate transformation matrix has been set such that the origin is at the top left corner of the physical page and the x-axis of the coordinate system is parallel to the top edge of the target page. The user units are equivalent to those set in the attribute list of the BeginSession command.

Example

```
SetPageDefaultCTM
```

Comments

See **Appendix K** for related error codes.

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 138	BLD 4600
-----------	-----------------	---------	----------	----------

path**Path Definition and Painting Operators****OPERATOR:** *SETPAGEORIGIN***Purpose**

Translate the origin of the page coordinate system to a new x, y location.

The initial page origin is the top left corner of the device's physical page.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { PageOrigin }

Attribute ID	Description and {value}
PageOrigin	The relative x, y point at which the page coordinate system origin will be relocated. The new origin is relative to the old origin (e.g. originX+x, originY+y) { xOrigin, yOrigin: xyValue }

Postcondition

The page CTM in the graphics state has been updated such that the new origin is at the x, y location relative to the previous origin.

Example

```
sint16_xy 2200 4850 PageOrigin
SetPageOrigin
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path	Path Definition and Painting Operators
-------------	---

OPERATOR: *SETPAGEROTATION***Purpose**

Rotate the user page coordinate system about its origin by +/- N degrees. A positive number rotates the coordinate system counterclockwise about its origin.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { PageAngle }

Attribute ID	Description and {value}
PageAngle	The relative angle at which the page coordinate system should be rotated about its current origin. Any rotation angle is accepted. A non-orthogonal (non-divisible by 90 degrees) will cause the Image, Brush and Pen subsystems to generate an error and flush the job. The new page angle is relative to the old page angle (e.g. OldPageAngle+NewPageAngle). { number }

Postcondition

The page CTM in the graphics state has been updated such the page coordinate system has been rotated about its own origin by N degrees as specified in the attribute list.

Example

```
sint16 270 PageAngle
SetPageRotation
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 140	BLD 4600
-----------	-----------------	---------	----------	----------

path**Path Definition and Painting Operators****OPERATOR: SETPAGE SCALE****Purpose**

Scale the page coordinate system in the x and y direction in user units.

Two options are allowed for SetPageScale. Option one allows the user to perform a simple scale of the page user units in the x and y direction. Option two scales the page coordinate system using the ratio of the session user units to the new user units specified in the SetPageScale attribute list. The actual operation performed for option two is shown below:

$$\text{SetPageScale} \frac{\text{Session User Units X}}{\text{New User Units X}}, \frac{\text{Session User Units Y}}{\text{New User Units Y}}$$

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair ::= { PageScale | { Measure & UnitsPerMeasure } }

Attribute ID	Description and {value}
PageScale	The factor for scaling the page coordinate system. { xUnits, yUnits: xyValue }
Measure	The new measure for each user unit. { eInch eMillimeter eTenthsOfAMillimeter }
UnitsPerMeasure	The new user resolution units in the x and y directions desired by the user for placing points (i.e. xUnits=600, yUnits=600 for 600 pixels-per-inch). { xUnits (+number), yUnits (+number): xyValue }

Postcondition

The page CTM in the graphics state has been updated such that the page coordinate system has been scaled according to the x, y values in the attribute list or according to the ratio of the session user units to the new user units specified in the attribute list.

Example

```
real32 1.25 PageScale
SetPageScale
```

Setting the page scale using the Measure and UnitsPerMeasure option

```
ubyte eInch Measure
sint16_xy 750 750 UnitsPerMeasure
SetPageScale
```

path**Path Definition and Painting Operators****Comments**

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path	Path Definition and Painting Operators
-------------	---

6.0 Painting Operators

This section contains a description of PCL XL operators to paint paths, scan lines and raster images.

6.1 *Path Manipulation Operators*

This section contains a description of the operators used to manipulate paths.

path	Defining and Painting a Path
-------------	-------------------------------------

OPERATOR: *CLOSESUBPATH*

Purpose

Join a line from the current cursor to the beginning of the current subpath to form a closed path (closed region).

CloseSubPath closes the most recent subpath so that proper join shapes will be drawn when the path is painted. Note: closed sub-paths do not get line caps applied during stroke operations.

CloseSubPath is automatically performed on all open subpaths when a non-null brush is set prior to a path paint operation. If a subpath is automatically closed, the “inside” of the path is filled with the non-null brush based on the current fill mode. After the fill takes place, the subpath is restored to its original state before any stroking operations. If the **PaintPath** operation also strokes the path due to a non-null pen, only the original path elements will be stroked.

After the painting operation, all automatically closed subpaths are restored to their previous state, thus preserving the original path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

A line has been joined from the current cursor to the beginning of the current subpath, closing the subpath.

The current cursor has been updated to be the close point of the path.

The current cursor is the beginning of a new subpath.

Example

```
CloseSubPath
```

Comments

See **Appendix K** for related error codes.

See also SetCursor and SetCursorRel.

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 144	BLD 4600
-----------	-----------------	---------	----------	----------

path**Defining and Painting a Path*****OPERATOR: NEWPATH*****Purpose**

Delete the current path (including subpaths) at the current graphic state level.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The current path has been deleted.

There is no current cursor.

Example

NewPath

Comments

See **Appendix K** for related error codes.

See also SetCursor and SetCursorRel.

path**Defining and Painting a Path*****OPERATOR: PAINTPATH*****Purpose**

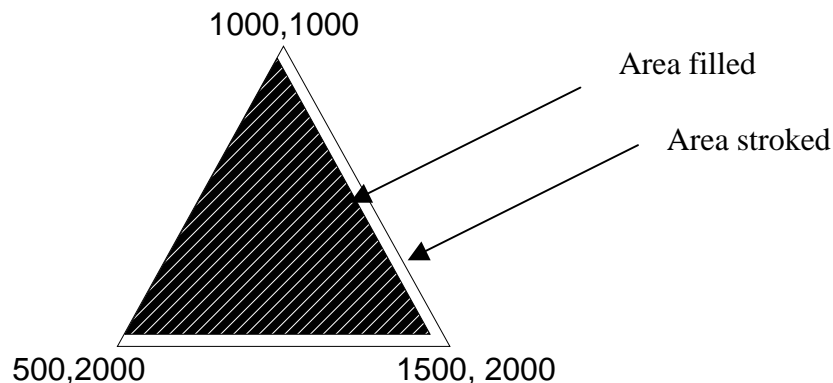
Paint the current path according to attribute settings in the graphics state.

The PaintPath operation first fills a geometric region defined by the CurrentPath with the current color or pattern define by the CurrentBrush. If the CurrentPath is empty or the CurrentBrush is NULL, no fill operation transpires. Paths are always implicitly closed prior to a filling operation by connecting the last point in the path to the first of the subpath. Subsequently, the path is restored to its previous state before stroking.

Secondly, the PaintPath operation strokes the geometric region defined by the CurrentPath with the current color or pattern define by the CurrentPen. If the CurrentPath is empty or the CurrentPen is NULL, no stroke operation transpires. LineEnds are applied to all ending line segments and line joins are applied to all connected segments. The actual geometric figure used for joins and ends is determined by the graphics state.

The area filled by a PaintPath operation is considered non-inclusive of the right and bottom of the path object. That is, the right and bottom device pixel specified by the path object is not included during fill operations. Referring to the triangle below, the area filled is approximately [999.5,1001.5], [1499,1999], [499.5,1999]. The actual bits filled are device dependent.

The region stroked by a PaintPath operator follows the geometric border of the path object and thus is considered inclusive of the right and bottom edge. The PenWidth determines the actual area stroked by a PaintPath. Again referring to the triangle below, the region stroked is approximately [1000,1000], [1500,2000], [500,2000]. The actual bits stroked are device dependent.



path**Defining and Painting a Path****Precondition**

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The current path has been painted to the page according to the values given in the graphics state.

The current path has been preserved.

The current cursor has been preserved.

If no current path existed prior to PaintPath, there is no painting performed.

Example

```
PaintPath
```

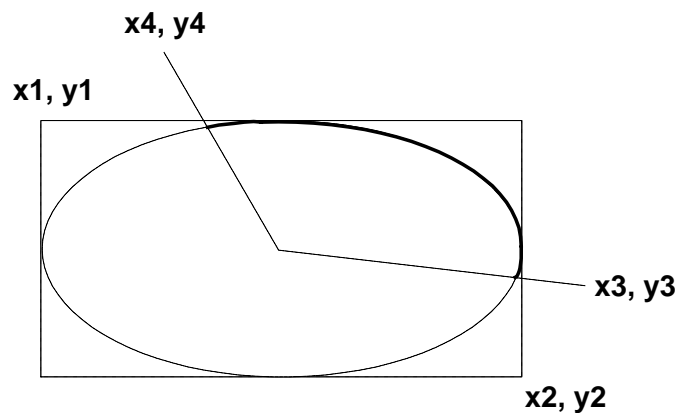
Comments

See **Appendix K** for related error codes.

path**Defining and Painting Arcs**

6.2 Defining and Painting Arcs

Arc operators describe an elliptical arc (see figure below). An imaginary ellipse defines the shape of the arc, which is the size of a bounding box. The length of the arc is defined by the intersection of the ellipse and two imaginary rays drawn from the center of the bounding box through points x_3, y_3 and x_4, y_4 . The starting point for the arc is the point at which the ray through point x_3, y_3 intersects the ellipse. The ending point for the arc is the point at which the ray through point x_4, y_4 intersects the ellipse. The arc is described or drawn in the direction specified by the ArcDirection setting in attribute list.



Arc Defined with Above X,Y Points and ArcDirection Set to CounterClockWise

CloseSubPath is performed automatically by closed-object path operators such as Circle, Ellipse, Rectangle, Pie, and Chord. A zero degree arc cannot be created.

An error occurs if the point x_3, y_3 or the point x_4, y_4 is equal to the center point of the bounding box.

path

Defining and Painting Arcs**OPERATOR: ARCPATH****Purpose**

Add an arc to the current path using absolute points.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList ::=

{ BoundingBox & StartPoint & EndPoint & {ArcDirection}_{opt} }

Attribute ID	Description and {value}
BoundingBox	The bounding box for the arc. { x1, y1, x2, y2: boxValue }
StartPoint	The starting point for the arc. { x3, y3: xyValue }
EndPoint	The ending point for the arc. { x3, y3: xyValue }
ArcDirection	The direction in which the arc should be drawn. If this attribute is missing the default is eCounterClockWise. { eClockWise eCounterClockWise }

Postcondition

An arc description has been added to the current path according to the points given in the attribute list. A 360-degree arc is created when the start point is equal to the end point. The cursor is at the end of the arc description.

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
sint16_xy 2000 1200 StartPoint
sint16_xy 4800 2000 EndPoint
ArcPath
```

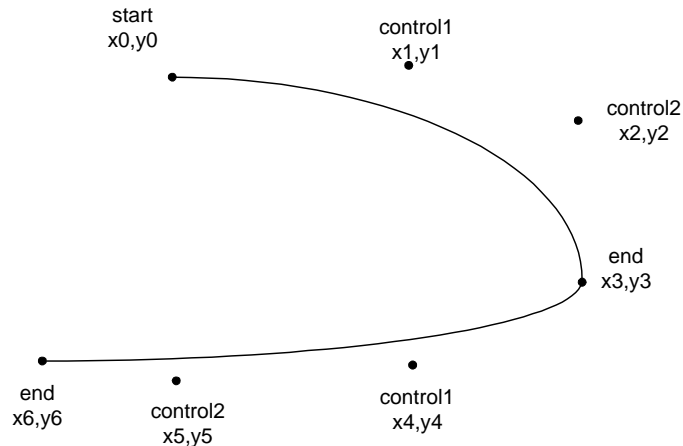
Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

6.3 Defining and Painting Beziers

Bezier operations describe a series of connected quadratic Bezier curves, with a start point, two control points, and an end point (see figure below).



The start point for a Bezier curve is the location of the current cursor. In addition to the start point, two control points and an end point must be defined for all Bezier curves. An operator describing a single Bezier curve acquires the three points following the start point from the attribute list.

An operator describing multiple Bezier curves reads multiple sets of three points following the start point from the data source. The first two points in the data source are control points one and two respectively for the first Bezier. The third point in the data source is the end point. The current cursor is set to the end point for the last curve defined in the data source. Each subsequent Bezier curve needs only three points from the data source. This is because the current cursor set at the previous Bezier's end point becomes the start point for the next Bezier. The total number of points for a multiple Bezier definition is defined in the Bezier operator's attribute list.

CloseSubPath is performed automatically by closed-object path operators such as Circle, Ellipse, Rectangle, Pie, and Chord.

path

Defining and Painting Beziers***OPERATOR: BEZIERPATH*****Purpose**

Describe a Bezier curve to be appended to the current path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current cursor is defined.

Attribute List Specification

```
multiAttributeList ::= { NumberOfPoints & PointType } |
                    { ControlPoint1 & ControlPoint2 & EndPoint }
```

Attribute ID	Description and {value}
ControlPoint1	The first control point for the Bezier. { x1, y1: xyValue }
ControlPoint2	The second control point for the Bezier. { x2, y2: xyValue }
EndPoint	The end point for the Bezier. { x3, y3: xyValue }
NumberOfPoints	The number of points to be read from the data source. This number must be a multiple of 3. { +integer }
PointType	The data type of the points to be read from the data source. This value specifies how to read point values from the data source. If the PCL XL stream is in a binary form, eUByte/eSByte will cause each point to be read as a single unsigned/signed byte, eUInt16 will cause each point to be read as two bytes containing an unsigned integer, and eSInt16 will read cause each point to be read as two bytes containing a signed integer. The position of the most significant byte in multi-byte points is determined by the OpenDataSource operator. If the PCL XL stream is in an ASCII form, the PointType value specifies data type and sign attribute of ASCII numbers read from the source. (eUByte eSByte eUInt16 eSInt16)

Postcondition

path**Defining and Painting Beziers**

Bezier description(s) have been added to the current path according to the points given in the attribute list/or the data source. The cursor is at the end of the last Bezier description.

Example

Adding a Bezier curve to the current path

```
uint16_xy 1800 1000 ControlPoint1
uint16_xy 2400 1800 ControlPoint2
uint16_xy 2400 2400 EndPoint
BezierPath
```

Adding two Bezier curve paths using points in a data array.

```
uint16 6 NumberOfPoints
ubyte eUInt16 PointType
BezierPath

dataLengthByte 24
sint16_raw* [ 1800 1000 2400 1800 2400 2400
              1800 3000 1200 3200 0 2600 ]
```

Comments

An error is raised if the current cursor is undefined.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path**Defining and Painting Beziers*****OPERATOR: BEZIERRELPATH*****Purpose**

Describe a Bezier curve to be appended to the current path using relative points.

When points are read from the data source instead of the attribute list, the points for the first Bezier are relative to the current cursor. The points for each successive Bezier are relative to the new current cursor, which is the end point of the previous Bezier.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current cursor is defined.

Attribute List Specification

```
multiAttributeList ::= { NumberOfPoints & PointType } |
                    { ControlPoint1 & ControlPoint2 & EndPoint }
```

Attribute ID	Description and {value}
ControlPoint1	The first control point for the Bezier, relative to the current cursor. { x1, y1: xyValue }
ControlPoint2	The second control point for the Bezier, relative to the current cursor. { x2, y2: xyValue }
EndPoint	The end point for the Bezier, relative to the current cursor. { x3, y3: xyValue }
NumberOfPoints	The number of points to be read from the data source. This number must be a multiple of 3. { +integer }

path	Defining and Painting Beziers
PointType	<p>The data type of the points to be read from the data source. This value specifies how to read point values from the data source.</p> <p>If the PCL XL stream is in a binary form, eUByte/eSByte will cause each point to be read as a single unsigned/signed byte, eUInt16 will cause each point to be read as two bytes containing an unsigned integer, and eSInt16 will read cause each point to be read as two bytes containing a signed integer. The position of the most significant byte in multi-byte points is determined by the OpenDataSource operator.</p> <p>If the PCL XL stream is in an ASCII form, the PointType value specifies data type and sign attribute of ASCII numbers read from the source.</p> <p>(eUByte eSByte eUInt16 eSInt16 }</p>

Postcondition

Bezier description(s) have been added to the current path according to the points given in the attribute list/or the data source. The cursor is at the end of the last Bezier description.

Adding a Bezier curve to the current path

```
sint16_xy 400 -200 ControlPoint1
sint16_xy 1200 600 ControlPoint2
sint16_xy 1200 1200 EndPoint
BezierRelPath
```

Adding two Bezier curves to the current path using points in a data array.

```
uint16 6 NumberOfPoints
ubyte eSInt16 PointType
BezierRelPath

dataLengthByte 24
sint16_raw* [ 400 -200 1200 600 1200 1200
              600 1800 0 2000 -1200 1400 ]
```

Comments

An error is raised if the current cursor is undefined.

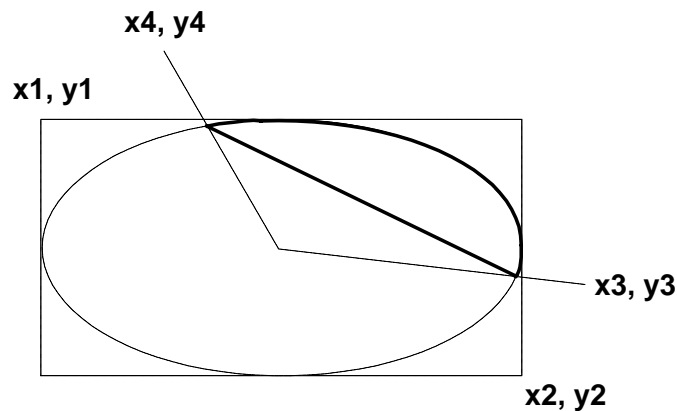
See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

6.4 Defining and Painting a Chord

Chord operators describe an arc and a line placed between the end points of the arc (see figure below). A chord is a closed subpath. An imaginary ellipse defines the shape of the arc portion of the chord, which is the size of a bounding box. The length of the arc is defined by the intersection of the ellipse and two imaginary rays drawn from the center of the bounding box through points x_3, y_3 and x_4, y_4 . The starting point for the arc is the point at which the ray through point x_3, y_3 intersects the ellipse. The ending point for the arc is the point at which the ray through point x_4, y_4 intersects the ellipse. The arc is described or drawn in a counterclockwise direction. The chord is placed between the end points of the arc.

A 360 degree arc is created when the start point is equal to the end point. The 360 degree arc is closed. An error occurs if the point x_3, y_3 or the point x_4, y_4 is equal to the center point of the bounding box.



A Chord Defined With Above X,Y Points

path

Defining and Painting a Chord**OPERATOR: CHORD****Purpose**

Paint a chord to the page using the current graphic state attributes.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList ::= { BoundingBox & StartPoint & EndPoint }

Attribute ID	Description and {value}
BoundingBox	The bounding box for the chord. { x1, y1, x2, y2: boxValue }
StartPoint	The starting point for the arc. { x3, y3: xyValue }
EndPoint	The ending point for the arc. { x3, y3: xyValue }

Postcondition

A chord has been painted to the page according to the points given in the attribute list. A 360-degree arc is created when the start point is equal to the end point.

The path existing prior to the operation has been destroyed.

The current path is left empty.

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
sint16_xy 2000 1200 StartPoint
sint16_xy 4800 2000 EndPoint
Chord
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

This operation is functionally equivalent to an expanded protocol sequence. The equivalent expanded protocol is shown below in BNF syntax:

Chord ::= { NewPath ChordPath PaintPath }

path

Defining and Painting a Chord*OPERATOR: CHORDPATH***Purpose**

Add a chord to the current path using the current graphic state attributes.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList ::= { BoundingBox & StartPoint & EndPoint }

Attribute ID	Description and {value}
BoundingBox	The bounding box for the chord. { x1, y1, x2, y2: boxValue }
StartPoint	The starting point for the arc. { x3, y3: xyValue }
EndPoint	The ending point for the arc. { x3, y3: xyValue }

Postcondition

An chord description has been added to the current path according to the points given in the attribute list.

The current cursor location is at point x1, y1 of the bounding box (the upper left point).

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
sint16_xy 2000 1200 StartPoint
sint16_xy 4800 2000 EndPoint
ChordPath
```

Comments

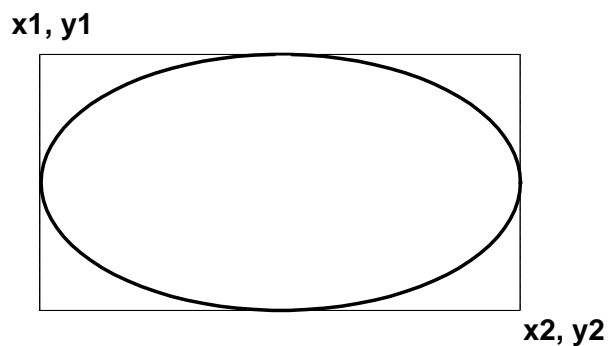
See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path**Defining and Painting an Ellipse**

6.5 *Defining and Painting an Ellipse*

Ellipse operators describe an ellipse that is contained within the bounding box defined by $x1,y1$ and $x2,y2$ (see figure below). An ellipse is a closed subpath. The ellipse contacts the bounding box at each box edge. The ellipse is described or drawn in a counterclockwise direction.



path**Defining and Painting an Ellipse*****OPERATOR: ELLIPSE*****Purpose**

Paint an ellipse to the page using the current graphic state attributes.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair := { BoundingBox }

Attribute ID	Description and {value}
BoundingBox	The points describing the bounding box for the ellipse. { x1, y1, x2, y2: boxValue }

Postcondition

An ellipse has been painted to the page according to the points given in the attribute list.

The path existing prior to the operation has been destroyed.

The current path is left empty.

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
Ellipse
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

This operation is functionally equivalent to an expanded protocol sequence. The equivalent expanded protocol is shown below in BNF syntax:

```
Ellipse ::= { NewPath EllipsePath PaintPath }
```

path**Defining and Painting an Ellipse****OPERATOR:** *ELLIPSEPATH***Purpose**

Add an ellipse description to a path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair := { BoundingBox }

Attribute ID	Description and {value}
BoundingBox	The points describing the bounding box for the ellipse. { x1, y1, x2, y2: boxValue }

Postcondition

An ellipse has been added to the current path according to the points given in the attribute list.

The current cursor location is at point x1, y1 of the bounding box (the upper left point).

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
EllipsePath
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path**Defining and Painting Lines**

6.6 Defining and Painting Lines

A line definition requires a start point and an end point. The start point for all lines is the location of the current cursor. For single lines, the end point of the line is specified by the end point attribute in the attribute list.

Placing end points in the data source may specify multiple lines. The first element in the data source establishes the end point of the first line. The cursor is set to that end point. The current cursor now acts as the start-point for the next line. The next element in the data point is the end point for the second line.

Lines so defined are placed into the current path. They can then be painted using the current graphic state or intersected with the current clip path using one of the clip path operators.

path

Defining and Painting Lines**OPERATOR: LINEPATH****Purpose**

Add one or more line descriptions to the current path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current cursor is defined.

Attribute List Specification

multiAttributeList ::= { EndPoint | { NumberOfPoints & PointType } }

Attribute ID	Description and {value}
EndPoint	The end point for the line. { x3, y3: xyValue }
NumberOfPoints	The number of points to be read from the data source. { +integer }
PointType	The data type of the points to be read from the data source. This value specifies how to read point values from the data source. If the PCL XL stream is in a binary form, eUByte/eSByte will cause each point to be read as a single unsigned/signed byte, eUInt16 will cause each point to be read as two bytes containing an unsigned integer, and eSInt16 will read cause each point to be read as two bytes containing a signed integer. The OpenDataSource operator determines the position of the most significant byte in multi-byte points. If the PCL XL stream is in an ASCII form, the PointType value specifies data type and sign attribute of ASCII numbers read from the source. (eUByte eSByte eUInt16 eSInt16)

Postcondition

One or more lines have been added to the current path according to the end point given in the attribute list or the end points in the data source.

The current cursor is at the last end point.

Example

Setting a new line end point for the current path

```
uint16_xy 2400 4250 EndPoint
```

path**Defining and Painting Lines**

LinePath

Setting a line path using points in a data array.

```
uint16 4 NumberOfPoints  
ubyte eSInt16 PointType  
LinePath
```

```
dataLengthByte 16  
sint16_raw* [ 0 0 4200 0 4200 5760 0 4200 ]
```

Comments

An error is raised if the current cursor is undefined.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path**Defining and Painting Lines****OPERATOR: *LINERELPATH*****Purpose**

Add line segments to the current path using relative points. The end point defined for the line is relative to the current cursor.

The end point for the first line is relative to the current cursor. The end point for each successive line is relative to the new current cursor, which is the end point of the previous line.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current cursor is defined.

Attribute List Specification

multiAttributeList ::= { EndPoint } | { NumberOfPoints & PointType }

Attribute ID	Description and {value}
EndPoint	The end point for the line, relative to the current cursor. { x3, y3: xyValue }
NumberOfPoints	The number of points to be read from the data source. { +integer }
PointType	The data type of the points to be read from the data source. This value specifies how to read point values from the data source. If the PCL XL stream is in a binary form, eUByte/eSByte will cause each point to be read as a single unsigned/signed byte, eUInt16 will cause each point to be read as two bytes containing an unsigned integer, and eSInt16 will read cause each point to be read as two bytes containing a signed integer. The position of the most significant byte in multi-byte points is determined by the OpenDataSource operator. If the PCL XL stream is in an ASCII form, the PointType value specifies data type and sign attribute of ASCII numbers read from the source. (eUByte eSByte eUInt16 eSInt16)

Postcondition

One or more lines have been added to the current path according to the end point given in the attribute list or the end points given in the data source.

path**Defining and Painting Lines**

The current cursor is at the last end point.

Example

Setting a new line end point for the current path

```
uint16_xy -350 425 EndPoint  
LineRelPath
```

Setting a line path using points in a data array.

```
uint16 4 NumberOfPoints  
ubyte eSInt16 PointType  
LineRelPath  
  
dataLengthByte 16  
sint16_raw* [ -5254 0 0 -57 5254 0 0 57 ]
```

Comments

An error is raised if the current cursor is undefined.

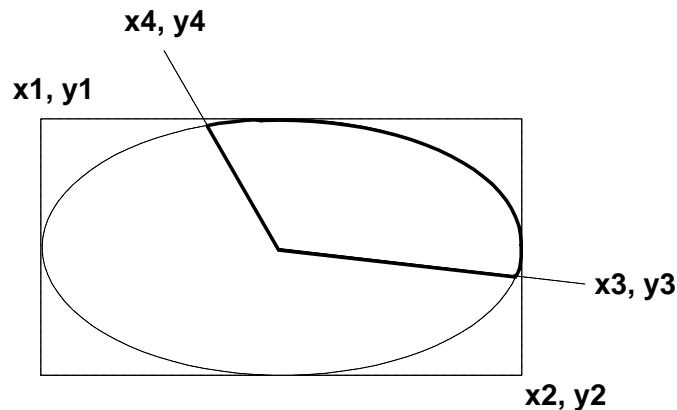
See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

6.7 Defining and Painting Pies

Pie operators describe an elliptical arc and two line segments placed in the path to close the pie shape (see figure below). A pie is a closed subpath. An imaginary ellipse defines the shape of the arc portion of the pie, which is the size of a bounding box. The length of the arc is defined by the intersection of the ellipse and two imaginary rays drawn from the center of the bounding box through points x_3, y_3 and x_4, y_4 . The starting point for the arc is the point at which the ray through point x_3, y_3 intersects the ellipse. The ending point for the arc is the point at which the ray through point x_4, y_4 intersects the ellipse. The arc is described or drawn a counterclockwise direction. Lines are placed along the rays for x_3, y_3 and x_4, y_4 to the edge of the ellipse defined by the bounding box to complete the pie shape.

The pie appears as a 360 degree arc when the start point is equal to the end point, having a line drawn from the center of the pie to the overlaid start and end points. Even though there appears to be only one line from the center, both lines (edges) are actually drawn and overlaid. An error occurs if the point x_3, y_3 or the point x_4, y_4 is equal to the center point of the bounding box.



A Pie Defined With Above X,Y Points

path**Defining and Painting Pies****OPERATOR: PIE****Purpose**

Paint a pie wedge to the current page using the current graphic state.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList ::= { BoundingBox & StartPoint & EndPoint }

Attribute ID	Description and {value}
BoundingBox	The bounding box for the pie shape. { x1, y1, x2, y2: boxValue }
StartPoint	The starting point for the pie arc. { x3, y3: xyValue }
EndPoint	The ending point for the pie arc. { x3, y3: xyValue }

Postcondition

A pie has been painted to the page according to the points given in the attribute list. The pie appears as a 360-degree arc when the start point is equal to the end point, having a line drawn from the center of the pie to the overlaid start and end points. Even though there appears to be only one line from the center, both lines (edges) are actually drawn and overlaid.

The path existing prior to the operation has been destroyed.

The current path is left empty.

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
sint16_xy 2000 1200 StartPoint
sint16_xy 4800 2000 EndPoint
Pie
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

This operation is functionally equivalent to an expanded protocol sequence. The equivalent expanded protocol is shown below in BNF syntax:

Pie ::= { NewPath PiePath PaintPath NewPath }

path

Defining and Painting Pies

OPERATOR: *PIEPATH***Purpose**

Add a pie description to the current path using the current graphic state.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList ::= { BoundingBox & StartPoint & EndPoint }

Attribute ID	Description and {value}
BoundingBox	The bounding box for the pie shape. { x1, y1, x2, y2: boxValue }
StartPoint	The starting point for the pie arc. { x3, y3: xyValue }
EndPoint	The ending point for the pie arc. { x3, y3: xyValue }

Postcondition

A pie description has been added to the current path according to the points given in the attribute list.

The current cursor location is at point x1, y1 of the bounding box (the upper left point).

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
sint16_xy 2000 1200 StartPoint
sint16_xy 4800 2000 EndPoint
PiePath
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

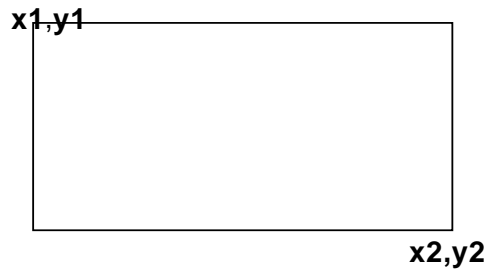
See **Appendix K** for related error codes.

path

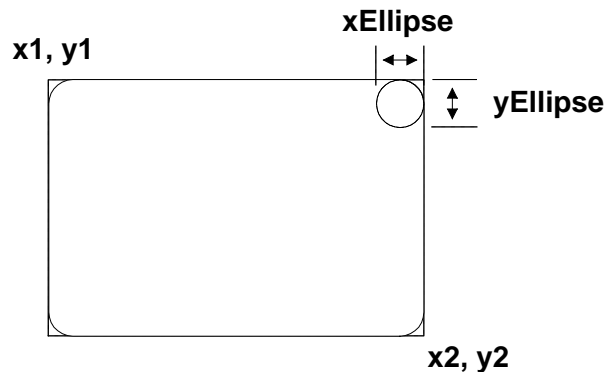
Defining and Painting Rectangles

6.8 Defining and Painting Rectangles and Round Rectangles

Rectangle operators describe a rectangle from an upper left point to a lower right point specified by a *bounding box* (see figure below). A rectangle is a closed subpath. The rectangle is described or drawn in a counterclockwise direction.



Rounded rectangle operators describe a rounded rectangle from the upper left point to the lower right point specified by the *bounding box* and the x and y dimensions of an ellipse determining the shape of the rounded corners (see figure below). The rounded rectangle is described or drawn in a counterclockwise direction. A rounded rectangle is a closed subpath.



path

Defining and Painting Rectangles

OPERATOR: RECTANGLE

Purpose

Paint a rectangle to a page using absolute points.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair := { BoundingBox }

Attribute ID	Description and {value}
BoundingBox	The points describing the bounding box for the rectangle. { x1, y1, x2, y2: boxValue }

Postcondition

A rectangle has been painted to the page according to the points given in the attribute list.

The path existing prior to the operation has been destroyed.

The current path is left empty.

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
Rectangle
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

This operation is functionally equivalent to an expanded protocol sequence. The equivalent expanded protocol is shown below in BNF syntax:

```
Rectangle ::= { NewPath RectanglePath PaintPath }
```

path**Defining and Painting Rectangles****OPERATOR: RECTANGLEPATH****Purpose**

Add a rectangle description to the current path using absolute points.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

singleAttributePair := { BoundingBox }

Attribute ID	Description and {value}
BoundingBox	The points describing the bounding box for the rectangle. { x1, y1, x2, y2: boxValue }

Postcondition

A rectangle description, drawn in counterclockwise direction, has been added to the current path according to the points given in the attribute list.

The current cursor location is at point x1, y1 of the bounding box (the upper left point).

Example

```
sint16_box 1200 1200 4800 2400 BoundingBox
Rectangle
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

path

Defining and Painting Rounded Rectangles***OPERATOR: ROUNDRECTANGLE*****Purpose**

Paint a rounded rectangle to a page using the current graphic state attributes.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList := { EllipseDimension & BoundingBox }

Attribute ID	Description and {value}
BoundingBox	The points describing the bounding box for the ellipse. { x1, y1, x2, y2: boxValue }
EllipseDimension	The x and y dimension of an ellipse that determines the shape of the rounded corners. { xEllipse, yEllipse: xyValue }

Postcondition

A rounded rectangle has been painted to the page according to the points given in the attribute list.

The path existing prior to the operation has been destroyed.

The current path is left empty.

Example

```
sint16_xy 24 48 EllipseDimension
sint16_box 1200 1200 4800 2400 BoundingBox
RoundRectangle
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

This operation is functionally equivalent to an expanded protocol sequence. The equivalent expanded protocol is shown below in BNF syntax:

```
RoundRectangle ::= { NewPath RoundRectanglePath PaintPath NewPath }
```

path**Defining and Painting Rounded Rectangles*****OPERATOR: ROUNDRECTANGLEPATH*****Purpose**

Add a rounded rectangle description to the current path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

multiAttributeList := { EllipseDimension & BoundingBox }

Attribute ID	Description and {value}
BoundingBox	The points describing the bounding box for the ellipse. { x1, y1, x2, y2: boxValue }
EllipseDimension	The x and y dimension of an ellipse that determines the shape of the rounded corners. { xEllipse, yEllipse: xyValue }

Postcondition

A rounded rectangle description has been added to the current path according to the points given in the attribute list.

The current cursor location is at point x1, y1 of the bounding box (the upper left point).

Example

```
sint16_xy 24 48 EllipseDimension
sint16_box 1200 1200 4800 2400 BoundingBox
RoundRectanglePath
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

image**Defining and Painting Bitmap Images**

6.10 Defining and Painting Bitmap Images

Bitmap images must be read into the device within Begin/Read/EndImage operators.

BeginImage defines the start of a new logical image and specifies parameters that are static during the process of reading image data. ReadImage specifies parameters pertaining to individual blocks of image data.

If the color space and palette needed for the image are different from the current color space and palette, the user may wish to perform a PushGS before setting the color space and palette for the Begin/Read/EndImage protocol sequence. A PopGS will restore the previous color space and palette settings.

image**Defining and Painting Bitmap Images****OPERATOR: BEGINIMAGE****Purpose**

Setup the device for painting a new bitmap image at the current cursor location.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The data source must be open from which image blocks are to be read.

The current color space must match the image data.

If the image pixel colors are indexed to a palette, the palette must exist in the color space and the palette length must be compatible with the depth of the image source pixels.

The current cursor must be set to a valid x, y point.

Neither the source width nor source height is zero.

Neither the x nor y destination size value is zero.

Attribute List Specification

```
multiAttributeList ::= { ColorDepth & ColorMapping &
                        DestinationSize & SourceWidth & SourceHeight }
```

Attribute ID	Description and {value}
ColorMapping	An enumeration specifying whether the component color mapping is direct or indexed through a palette. { eDirectPixel eIndexedPixel }
ColorDepth	The number of bits per image component. { e1Bit e4Bit e8Bit }
SourceWidth	The width of the image source in source pixels. { +integer }
SourceHeight	The height of the image source in source pixels. { +integer }
DestinationSize	The size of the destination box on the page for the image in current user units. { xyValue }

Postcondition

The device has been set up to accept ReadImage operators.

Example

```
ubyte eDirectPixel ColorMapping
```

image**Defining and Painting Bitmap Images**

```
ubyte e8Bit ColorDepth
uint16 640 SourceWidth
uint16 480 SourceHeight
uint16_xy 3200 2400 DestinationSize
BeginImage
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

image**Defining and Painting Bitmap Images*****OPERATOR: READIMAGE*****Purpose**

Read a block of a bitmap image data from the current data source.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The StartLine attribute is in sequence relative to the end line of the previous ReadImage.

The data source from which image blocks are read must be open.

The data source must have at least one block remaining that matches the BeginImage attribute list definition and the attribute list definition for this ReadImage.

The first byte of the image block is the left-most byte of the block (No byte swapping is performed on the binary data describing the image block).

The length of each line of image data within the image block must be a multiple of four bytes unless the PadBytesMultiple parameter is specified. If the PadBytesMultiple attribute is supplied the number of bytes in each line must be a multiple of the value of that attribute.

Attribute List Specification

multiAttributeList ::= { CompressMode & StartLine & BlockHeight
& { PadBytesMultiple }_{opt} & { BlockByteLength }_{opt} }

Attribute ID	Description and {value}
StartLine	The line within the source image at which this block is located. The line position is in source pixels. The line must be in sequence relative to the end line of the previous ReadImage. The first start line must be zero. { +integer }
BlockHeight	The height of the image data block in source pixels. { +integer }
CompressMode	The compression mode for the block being read. { eNoCompression eRLECompression eJPEGCompression }
PadBytesMultiple	The number of pad bytes for each line in the image block (default = 4). This number must be in the range 1 to 4. { +ubyte }

image	Defining and Painting Bitmap Images
--------------	--

BlockByteLength	<p>This attribute specifies the number of bytes used to describe the image block. If the block is compressed, the number is the number of bytes in compressed form. If this attribute is not included the source image data is embedded within the XL stream and the embedded data tag describes the number of bytes in the block.</p> <p>{+integer}</p>
-----------------	--

Postcondition

A block of bitmap image data has been read by the device from the current data source.

Example

```
uint16 0 StartLine
uint16 480 BlockHeight
ubyte eNoCompression CompressMode
// Operator Position: 10
ReadImage

dataLength 921600
hex_raw* [
34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34
55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55
.
.
.
6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d
34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34
[
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

image**Defining and Painting Bitmap Images*****OPERATOR: ENDIMAGE*****Purpose**

End the definition of a bitmap image and cause it to be painted.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The newly defined image has been scheduled for painting.

Example

```
EndImage
```

Comments

See **Appendix K** for related error codes.

6.11 Defining and Painting Raster Patterns

Raster patterns must be read into the device within Begin/Read/EndRastPattern operators. Raster patterns are tiled bitmap images used to stroke or fill path objects and brush text on the page. A raster pattern must be associated with a numeric identifier when it is read into the device.

BeginRastPattern defines the start of a new logical bitmap pattern and specifies parameters that are static during the process of reading pattern data. ReadRastPattern defines parameters pertaining to individual blocks of raster pattern data.

Raster patterns remain in the device until a PopGS, the end of a page (EndPage) or the end of a session (EndSession) depending on the persistence requested for the pattern. These types of patterns are called temporary, page, and session persistent respectfully.

Patterns can be assigned to the Pen and Brush prior to painting any graphical object. When painted, these objects will be “filled” with the raster pattern. The raster pattern will be duplicated multiple times in the X and Y direction to entirely “fill” the object. The anchor for the pattern will be the current pattern anchor defined in the current graphic state.

The destination box specifying the size of the raster pattern on the page is defined in the user units at the time the pattern is downloaded. The actual size of the pattern on the page will depend on the page CTM settings in effect when the pattern is associated with a pen or a brush. The page CTM converts the user units specified for the pattern size to device units when SetPenSource or SetBrushSource is performed. The user has the option of overriding the original destination box size when the pen or brush is set.

The current CTM rotation determines the rotation of the pattern at the time SetPenSource or SetBrushSource is performed. If the CTM is in default rotation, the bottom of the pattern as downloaded will be parallel to the bottom of the physical page (i.e. parallel to the short edge for portrait orientation and to the long edge for landscape orientation).

The color space and palette settings for the pattern must be set prior to the SetPenSource or SetBrushSource operators such that they are compatible with the pattern data to be read. If the color space and palette needed for the pattern are different from the current color space and palette, the user may wish to perform a PushGS before setting the color space and palette for the Begin/Read/EndRastPattern protocol sequence. A PopGS will restore the previous color space and palette settings.

The identifier associated with the raster pattern is used as an attribute value for the SetPenSource and SetBrushSource operators. Once this pattern is selected for a pen or brush, the pattern will stay in effect for the pen or brush in the current graphics state. This is true even if the user downloads a new pattern with a matching PatternDefineID.

pattern**Defining and Painting Raster Patterns**

Note: A problem has been identified with the PCL XL 1.x pattern subsystem that requires a rule be followed when designing for backward compatibility with PCL XL 2.0. For PCL XL 1.x, the color space in effect when a pattern is set to a pen or a brush must be the same color space and palette that was in effect at the time the pattern was originally downloaded.

pattern**Defining and Painting Raster Patterns****OPERATOR: *BEGINRASTPATTERN*****Purpose**

Start the definition of a new raster pattern (bitmap pattern).

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The data source from which raster pattern blocks are to be read must be open.

Neither the source width nor source height is zero.

Neither the x nor y destination size value is zero.

Attribute List Specification

```
multiAttributeList ::= { ColorDepth & ColorMapping &
                        DestinationSize & SourceWidth & SourceHeight &
                        { PatternDefineID & PatternPersistence } }
```

Attribute ID	Description and {value}
ColorMapping	An enumeration specifying whether the component color mapping is direct or indexed through a palette. { eDirectPixel eIndexedPixel }
ColorDepth	The number of bits per image component. { e1Bit e4Bit e8Bit }
SourceWidth	The width of the raster pattern source in source pixels. { +integer }
SourceHeight	The height of the raster pattern source in source pixels. { +integer }
DestinationSize	The size of the destination box on the page for the raster pattern specified in user units. { xyValue }

pattern**Defining and Painting Raster Patterns**

Attribute ID	Description and {value}
PatternDefineID	<p>The numeric identifier by which the pattern will be known and selected in a SetBrushSource and/or SetPenSource operation. The same id number may be reused for temporary-, page-, and session-persistent patterns without destroying the previously defined patterns. However, the precedence on pattern selection in SetPen/BrushSource is temporary first (if exists), page second (if exists), and session third (if exists).</p> <p>{integer}</p>
PatternPersistence	<p>An enumerated value specifying the persistence of the pattern.</p> <p>If the enumeration is eTempPattern, the pattern will be created with temporary persistence. The pattern is accessible by a SetPenSource or SetBrushSource using this identifier in the current or higher (PushGS) graphics state levels until replaced by a temporary pattern using the same identifier or until the current graphics state level is popped (PopGS).</p> <p>If the enumeration is ePagePattern the pattern will be created with page persistence. The pattern is accessible by a SetPenSource or SetBrushSource using the identifier for the remainder of the page or until replaced by a page pattern using the same identifier.</p> <p>If the enumeration is eSessionPattern, the pattern will be created with session persistence. The pattern is accessible by a SetPenSource or SetBrushSource using the identifier for the remainder of the session or until replaced by a session pattern using the same identifier.</p> <p>{ eTempPattern ePagePattern eSessionPattern }</p>

Postcondition

A raster pattern descriptor has been defined and associated to a pattern identifier. No pattern data yet exists.

Any pattern defined with the same pattern ID at the same persistence level will have been deleted.

Example

```

ubyte eDirectPixel ColorMapping
ubyte e8Bit ColorDepth
uint16 8 SourceWidth

```

pattern**Defining and Painting Raster Patterns**

```
uint16 8 SourceHeight
uint16_xy 8 8 DestinationSize
ubyte ePagePattern PatternPersistence
BeginRastPattern
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

pattern**Defining and Painting Raster Patterns****OPERATOR: READRASTPATTERN****Purpose**

Read a block of a raster pattern data from the current data source.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The data source from which raster pattern blocks are read must be open.

The StartLine attribute is in sequence relative to the end line of the previous ReadRastPattern.

The data source must have at least one block remaining that matches the attribute list definition for BeginRastPattern and the attribute list definition for this ReadRastPattern.

The first byte of the pattern block is the left-most byte of the block (no byte swapping is performed on the binary data describing the raster pattern block).

The length of each line of image data within the pattern block must be a multiple of four bytes. If a line is not a multiple of four bytes, it must be padded with the appropriate number of bytes with a value of zero.

The length of each line of pattern data within the pattern block must be a multiple of four bytes unless the PadBytesMultiple parameter is specified. If the PadBytesMultiple attribute is supplied the number of bytes in each line must be a multiple of the value of that attribute.

Attribute List Specification

multiAttributeList ::= { CompressMode & StartLine & BlockHeight
& { PadBytesMultiple }_{opt} & { BlockByteLength }_{opt} }

Attribute ID	Description and {value}
StartLine	The line within the source raster block to start reading data in source pixels. The line must be in sequence relative to the end line of the previous ReadImage. The first start line must be zero. { +integer }
BlockHeight	The height of the raster data block in source pixels. { +integer }
CompressMode	The compression mode for the block being read. { eNoCompression eRLECompression eJPEGCompression }

pattern	Defining and Painting Raster Patterns
---------	---------------------------------------

PadBytesMultiple	The number of pad bytes for each line in the pattern image block (default = 4). This number must be greater than zero. {+ubyte}
BlockByteLength	This attribute specifies the number of bytes used to described the pattern image block. If the block is compressed, the number is the number of bytes in compressed form. If this attribute is not included the source image data is embedded within the XL stream and the embedded data tag describes the number of bytes in the block. {+integer}

Postcondition

A block of raster pattern data has been read by the device from the current data source.

Example

```
uint16 0 StartLine
uint16 8 BlockHeight
ubyte eNoCompression CompressMode
// Operator Position: 10
ReadRastPattern

dataLength 64
hex_raw* [
34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34
55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55
6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d
34 55 6d 34 55 6d 34 55 6d 34 55 6d 34 55 6d 34
]
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

pattern**Defining and Painting Raster Patterns*****OPERATOR: ENDRASTPATTERN*****Purpose**

End the definition of a raster pattern. The pattern defined is ready for use during painting operations by assignment to the Pen or Brush.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

A new raster pattern has been defined for a pen and/or a brush.

Example

```
EndRastPattern
```

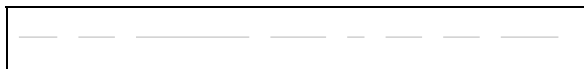
Comments

See **Appendix K** for related error codes.

6.12 Defining and Painting Scan Lines

Applications and graphical user interfaces sometimes find it necessary to resort to drawing complex graphical regions with one user-unit high scan lines. Scan lines must be read into a PCL XL device using the BeginScan, ScanLineRel, EndScan operators.

Each scan line segment of a scan line is one user unit high and has butt-end caps. Scan lines are colored with the current brush setting in the graphics state. The following is a sample scan line:



A Sample Scan Line

BeginScan defines the start of a new logical block of scan lines. The actual data for the scan lines is read from the data source. The ScanLineRel operator reads one or more scan lines from the open data source. The ScanLineRel operator contains an attribute describing the number of scan lines to be found in the data source. The EndScan operator finishes a logical block of scan lines. The current path is null following the EndScan operator.

Each scan line has an XStart, YStart value in the data source describing the scan line starting location. The next scan line data are a pair of x values describing the starting and ending points for the first scan line segment. These x values are relative to XStart and at YStart. Each remaining pair of x values in the scan line are relative to the end point of the most recent scan line. Each scan line segment is drawn up to, but not including its end point.

The format of scan line data is as follows:

```
{XStart, YStart type: ubyte}
{XStart1}{YStart1}{# of x-pairs1: uint16}{x-pair type1: ubyte}{xpair1,1}{xpair1,2}...{xpair1,n}
{XStart2}{YStart2}{# of x-pairs2: uint16}{x-pair type2: ubyte}{xpair2,1}{xpair2,2}...{xpair2,n}
...
{XStartN}{YStartN}{# of x-pairsN: uint16}{x-pair typeN: ubyte}{xpairN,1}{xpairN,2}...{xpairN,n}
```

Each scan line is described in a compact data form. The first byte of the scan line data defines the data type of the XStart and YStart values found in each scan line description. This data type value is an enumeration identical to the PointType enumeration for operators such as LineRelPath. Note: the only XStart, YStart data type allowed for all current protocol classes of PCL XL is eSInt16. Following the data type field are the XStart and YStart values for the first scan line of the ScanLineRel block.

The next field is the number of x-pairs for the scan line. This is always an unsigned 16-bit integer value greater than zero. Following the field for number of x-pairs is the x-pair type field, which is also a PointType enumeration. This enumeration is restricted to eUByte and eUInt16. The scan line segment x-pairs follow the x-pair data type field.

scan

Defining and Painting Scan Lines

OPERATOR: *BEGINSCAN*

Purpose

Start the definition of a new block of scan lines to be filled with the current brush source.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The device is set up to begin reading scan line definitions.

Example

```
BeginScan
```

Immediately followed by 18 bytes of data describing one scan line with three X pairs.

```
dataLength 36
hex_raw* [
03          // Enum for eSInt16 (anything else is an error)
00 00 00 00 // Scan line starts at 0,0
03 00       // Number of scan line pairs low byte first
00          // Data type of the XPairs, eUInt16 in this case
00 00 20 00 // First line section, 0,0 to 32,0
30 00 40 00 // Second line section 48,0 to 64,0
60 00 80 00 // Third line section 96,0 to 128,0
           // Second scan line
03          // Enum for eSInt16 (anything else is an error)
00 00 01 00 // Scan line starts at 0,1
03 00       // Number of scan line pairs low byte first
00          // Data type of the XPairs, eUInt16 in this case
00 00 28 00 // First line section, 0,1 to 40,1
38 00 48 00 // Second line section 56,1 to 72,1
68 00 88 00 // Third line section 104,1 to 136,1
           [
```

Comments

See **Appendix G** for valid embedded data field ranges for XStart, YStart point type and x-pair type (PointType Enumeration).

See **Appendix K** for related error codes.

The number of x-pairs field for each scan line in the data source must always be greater than zero.

scan**Defining and Painting Scan Lines*****OPERATOR: SCANLINEREL*****Purpose**

Start the definition of a new block of scan lines using coordinates *relative* to the X, Y starting point. The relative coordinates are positive x values.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The data in the current open data source will satisfy the request by the ScanLineRel operator.

Attribute List Specification

singleAttributePair ::= {NumberOfScanLines}_{opt}

Attribute ID	Description and {value}
NumberOfScanLines	<p>The number of scan lines defined in the data source read by the ScanLineRel operator. If this attribute is missing, it is assumed that the number of scan lines is one.</p> <p>The range of this attribute is integer values of 1 or more.</p> <p>{+integer greater than zero and less than 65536}</p>

Postcondition

A block of scan line data has been read and painted to the page.

Example

```
uint16 2 NumberOfScanLines
ScanLineRel
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

scan

Defining and Painting Scan Lines*OPERATOR: ENDSCAN***Purpose**

End the definition of a new block of scan lines.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

Attribute List Specification

nullAttributeList

Postcondition

The scanline subsystem has been closed and its resources returned to the device.

The current path has been destroyed.

Example

EndScan

Comments

See **Appendix K** for related error codes.

text	Painting Text
-------------	----------------------

6.13 *Painting Text*

This section describes the set of operators used to paint text to a page.

Text in a PCL XL device can be described by two distinct categories: outlined text and bitmap text. Outline text describes the text in geometric form that is inherently transformable - scale, rotate, transform, and skew. Outline text can also be placed into the current path and current clip path, consequently text can be operated upon as path objects.

Bitmap text is a device specific raster definition that cannot be transformed in any form. Bitmap text format is primarily provided for backward compatibility and its use should be restricted whenever possible.

For device independent reasons, it is advisable to utilize outline text whenever possible.

text	Painting Text
-------------	----------------------

OPERATOR: TEXT**Purpose**

Paint characters starting from the current cursor using user-specified character escapements. No device escapements are used. If the character data represents outline-defined characters, the area inside the outlines defines the source pixels to be painted. If the character data represents bitmap characters, only the black pixels are treated as the source pixels to be painted. The white pixels have no effect in the painting process.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

There is a valid current font set in the graphics state.

All array attributes provided in the attribute list contain the same number of elements.

The current cursor is valid.

Attribute List Specification

multiAttributeList ::= { {XSpacingData}_{opt} & {YSpacingData}_{opt} & TextData }

Attribute ID	Description and {value}
TextData	An unsigned integer array representing individual character codes in the current font. { integerArray }
XSpacingData	An array representing “x” value escapements for the corresponding character codes in the TextData series. If this attribute is not present, the “x” escapement will be zero for each character in the TextData array. { numberArray }
YSpacingData	An array representing “y” value escapements for the corresponding character codes in the TextData series. If this attribute is not present, the “y” escapement will be zero for each character in the TextData array. { numberArray }

Postcondition

Characters have been painted to the page and spaced according the SpacingData in the attribute list.

The current cursor is at the location calculated by the sum of relative moves implied by the X and/or Y spacing data arrays and relative to the cursor location prior to the operation.

Example

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 193	BLD 4600
-----------	-----------------	---------	----------	----------

text	Painting Text
-------------	----------------------

```

ubyte_array [
  80 67 76 32 88 76 32 70 // PCL XL F
101 97 116 117 114 101 32 82 // eature R
101 102 101 114 101 110 99 101 // eference
] TextData
ubyte_array [
  56 67 61 25 72 61 25 56 // 8C=.H=.8
  44 44 28 50 33 44 25 67 // ,,.2!,,.C
  44 33 44 33 44 50 44 44 // ,!,!,2,,
] XSpacingData
Text

```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

text	Painting Text
-------------	----------------------

OPERATOR: TEXTPATH**Purpose**

Add character outlines to the current path in the form of lines and Beziers. The starting point for the path is the current cursor. The character escapements provided define the character spacing.

Note: the boldness setting is not applied to characters that are placed into the path.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

There is a valid current font set in the graphics state.

All array attributes provided in the attribute list are the same length.

The current cursor is valid.

Attribute List Specification

multiAttributeList ::= { {XSpacingData}_{opt} & {YSpacingData}_{opt} & TextData }

Attribute ID	Description and {value}
TextData	An unsigned integer array representing individual character codes in the current font. { integerArray }
XSpacingData	An array representing “x” value escapements for the corresponding character codes in the TextData series. If this attribute is not present, the “x” escapement will be zero for each character in the TextData array. { numberArray }
YSpacingData	An array representing “y” value escapements for the corresponding character codes in the TextData series. If this attribute is not present, the “y” escapement will be zero for each character in the TextData array. { numberArray }

Postcondition

Characters have been added to the current path and spaced according to the SpacingData in the attribute list.

The current cursor is at the location calculated by the sum of relative moves implied by the X and/or Y spacing data arrays and relative to the cursor location prior to the operation.

Example

REV: p1.8	DATE: 06Jan1998	DWG NO:	PAGE 195	BLD 4600
-----------	-----------------	---------	----------	----------

text	Painting Text
-------------	----------------------

```

ubyte_array [
  80 67 76 32 88 76 32 70 // PCL XL F
101 97 116 117 114 101 32 82 // eature R
101 102 101 114 101 110 99 101 // eference
] TextData
ubyte_array [
  56 67 61 25 72 61 25 56 // 8C=.H=.8
  44 44 28 50 33 44 25 67 // ,,.2!,,.C
  44 33 44 33 44 50 44 44 // ,!,!,2,,
] XSpacingData
TextPath

```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

7.0 User-Defined Streams

7.1 Defining and Using User-Defined Streams

A stream in a PCL XL device is a byte-oriented sequence of operators and data. All PCL XL devices with a serial byte-stream interface execute a PCL XL stream to process session operators.

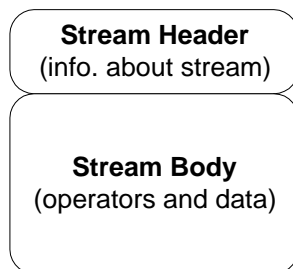
A PCL XL device allows the user to name a sequence of operators and data. This sequence is called a “user-defined stream” and is defined in a byte-oriented manner compatible with the target device.

A PCL XL device via the **Begin/Read/EndStream** operators loads a user-defined stream. The stream may be referenced by name in the **ExecStream** operator for later execution. Embedded data source data may be interleaved with operators in user-defined streams. The syntax for embedding data source data in a stream is format-specific (see the stream format section of this document).

A PCL XL user-defined stream may be executed anywhere within a **BeginSession / EndSession** operator pair using the **ExecStream** operator. All PCL XL operators and data are allowed in a user-defined stream with the exception of the **BeginStream, ReadStream, EndStream, and ExecStream** operators. Nested execution where a user-defined stream executes another user-defined stream is not allowed. Streams must not contain nested sessions where a **BeginSession** occurs inside another **BeginSession**.

Once defined, user-defined streams remain accessible by name throughout the current session or until a **RemoveStream** operation. Devices with non-volatile mass storage such as disk or flash may have streams that last across session boundaries.

All streams have two parts: (1) the stream header and (2) the stream body as shown in the figure below:



A PCL XL Protocol Stream

The stream header identifies the ASCII or binary format of the stream and other information about the stream. The stream body contains operators and data for the stream. All PCL XL streams conform to this format, including the parent stream containing one or more sessions for a byte-stream device.

streams**Defining and Using User-Defined Streams**

7.2 Stream Header Format.

The stream header format shown below is the same for user-defined streams and the parent session stream for byte-stream devices.

Starting Byte	Ending Byte	Header Element Description
0	0	Binding Format Identifier
1	1	Reserved, use ASCII space (hex 0x20)
2	N-1	Stream Descriptor String
N	N	End of Header, ASCII line feed (hex 0x0a)
N+1	N+1	Beginning of PCL XL Stream body containing operators and data

Binding Format Identifier

The **Binding Format Identifier** is a single byte denoting the binding (ASCII or binary) for operators and data in the stream body. The following table lists valid binding format identifier values:

Char	Hex Value	Binding Format Identifier Meaning
'	0x27	A single quote character. A PCL XL ASCII binding is used for operators and data.
)	0x29	A right parenthesis character. A binary binding follows in the stream body where operator identifiers, attribute identifiers, and attribute values are expressed in a form where the least significant byte is the first byte in the binary field (from left to right) and the most significant byte is last (to the right).
(0x28	A left parenthesis character. A binary binding follows in the stream body where operator identifiers, attribute identifiers, and attribute values are expressed in a form where the most significant byte is first byte in the binary field (from left to right) and the least significant byte is last (to the right).

Binding Format Identifier Values

Stream Descriptor String

The stream Descriptor String is a formatted, line-feed terminated string containing stream information. This string has three required fields: (1) the stream class name, (2) the protocol class number of the operators in the stream body, and (3) the protocol class revision of the operators in the stream body. In this string, fields are separated by semicolons and data values are separated by commas. A sample string is shown below:

```
HP-PCL XL;2;0<lf>
```

Example

```
hex_raw* [ 29 20 48 50 2d 50 43 4c 20 58 4c 3b 32 3b 30 0a ]
```

In the sample string, the stream class name is identified as “HP-PCL XL.” Following the class name is the PCL XL protocol class number (‘2’) and protocol class revision (‘0’). The string is terminated by an ASCII line feed character (<lf> = 0x0a_{hex}), although a carriage return character is allowed just before the line feed character (<cr> = 0x0d_{hex}). Devices may use the stream class fields to determine compatibility with the stream.

The stream class name field, protocol class number, and revision field must be the first three consecutive fields in a stream header. Additional supported fields may be added using the field and data separator conventions as described above. For example, a field defined for stream header comments may be added to the stream class string as follows:

```
HP-PCL XL;2;0;Comment Copyright Hewlett-Packard Company 1989-1997<lf>
```

Example

```
hex_raw* [
48 50 2d 50 43 4c 20 58 4c 3b 32 3b 30 3b 43 6f
6d 6d 65 6e 74 20 43 6f 70 79 72 69 67 68 74 20
48 65 77 6c 65 74 74 2d 50 61 63 6b 61 72 64 20
43 6f 6d 70 61 6e 79 20 31 39 38 39 2d 31 39 39
37 0a
]
```

In the above example, every character following the “Comment” string is skipped until a semicolon or line feed is reached. In all cases, the stream descriptor string must be terminated with a line feed character prior to the stream body containing stream operators and data.

OPERATOR: *BEGINSTREAM***Purpose**

Begin the capture of a user-defined stream where the stream is a set of PCL XL operators and data in a binding format compatible with the device. The stream is not interpreted nor checked for correct syntax at capture-time.

Once a stream is defined, it may be accessed anytime during the session with the ExecStream operator.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current stream in which this operator is interpreted is not a user-defined stream.

Attribute List Specification

multiAttributeList ::= { StreamName }

Attribute ID	Description and {value}
StreamName	The name of the stream to be captured. { integerArray }

Postcondition

The device has been put into a mode in which stream data may be accepted.

If a stream of the same name existed prior to BeginStream being executed, the original stream has been replaced by the new description.

Example

```
ubyte_array (SampleStream) StreamName
BeginStream
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

streams**Defining and Using User-Defined Streams*****OPERATOR: READSTREAM*****Purpose**

Read a segment of a user-defined stream.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current stream is which this operator is interpreted not a user-defined stream.

The data source is open from which stream data will be read.

Attribute List Specification

multiAttributeList ::= { StreamDataLength }

Attribute ID	Description and {value}
StreamDataLength	The length in bytes of stream data for this segment. { +integer }

Postcondition

A segment of PCL XL operators and data has been captured and appended to the previous data read for the current Begin/Read/EndStream operator sequence.

Example

```
uint32 131 StreamDataLength
ReadStream

dataLength 131
hex_raw* [
29 20 48 50 2d 50 43 4c 20 58 4c 3b 31 3b 31 3b
43 6f 6d 6d 65 6e 74 3a 20 43 6f 70 79 72 69 67
68 74 20 28 63 29 20 31 39 39 35 20 48 65 77 6c
65 74 74 2d 50 61 63 6b 61 72 64 20 43 6f 6d 70
61 6e 79 2e 20 20 41 6c 6c 20 72 69 67 68 74 73
20 72 65 73 65 72 76 65 64 2e 0d 0a c1 08 00 f8
4b 7a c0 01 f8 03 6a c5 00 00 00 00 f8 09 79 c5
00 00 80 3f f8 09 63 e1 b0 04 b0 04 68 10 68 10
f8 42 98
]
```

Comments

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

streams**Defining and Using User-Defined Streams***OPERATOR: ENDSTREAM***Purpose**

End the definition of a user-defined stream.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current stream is which this operator is interpreted not a user-defined stream.

Attribute List Specification

nullAttributeList

Postcondition

Data for a user-defined stream has been defined and associated with the stream name given in the corresponding BeginStream attribute list.

Example

```
EndStream
```

Comments

See **Appendix K** for related error codes.

streams**Defining and Using User-Defined Streams*****OPERATOR: EXECSTREAM*****Purpose**

Execute a previously defined stream that contains a sequence of PCL XL operators and data.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The stream contains legal PCL XL attribute lists and operators.

The current stream in which this operator is interpreted is not a user-defined stream.

Attribute List Specification

multiAttributeList ::= { StreamName }

Attribute ID	Description and {value}
StreamName	The name of the user-defined stream to be executed. { integerArray }

Postcondition

The stream associated with the stream name has been executed.

Example

```
ubyte_array (SampleStream) StreamName
ExecStream
```

Comments

If no stream exists by StreamName PCL XL will raise an error.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

See **Appendix K** for related error codes.

streams**Defining and Using User-Defined Streams*****OPERATOR: REMOVESTREAM*****Purpose**

Remove the user-defined stream by name.

Precondition

Immediate execution of this operator occurs in a legal PCL XL operator sequence.

The current stream in which this operator is interpreted is not the stream being deleted.

Attribute List Specification

multiAttributeList ::= { StreamName }

Attribute ID	Description and {value}
StreamName	The name of the stream to be removed. { integerArray }

Postcondition

If a stream of the same name existed and was defined during the current session, the stream has been deleted.

Example

```
ubyte_array (SampleStream) StreamName
RemoveStream
```

Comments

If no stream exists by StreamName PCL XL will raise an error.

See **Appendix F** and **Appendix G** for valid attribute data types and ranges.

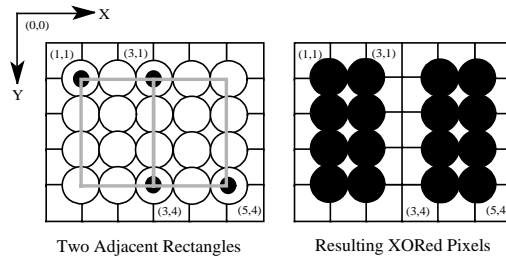
See **Appendix K** for related error codes.

Appendix

A. PCL XL and Pixel Placement

A PCL XL device places pixels at the intersection of the squares of a theoretical, device-dependent grid covering the printable area of a page. The grid dimensions correspond to the native device pixel resolution. For example, a 600 dpi device would have a 600 by 600 units-per-inch theoretical native grid for placing pixels.

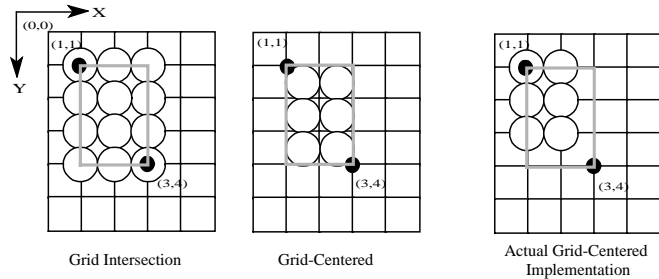
In past PCL implementations a problem could occur when the boundaries of two vector objects touch along a common border. Vector objects are path objects like ellipses, rectangles, polygons, etc. Consider the two rectangle objects in the following figure.



The left figure shows two rectangles that are adjacent to each other. The boundary of one rectangle is from coordinate (1,1) to coordinate (3,4). The boundary of the other rectangle is from coordinate (3,1) to coordinate (5,4). The circles in the figure above demonstrate pixels that would be filled to represent the area of the rectangles. As depicted in the figure, twelve pixels represent the area of each rectangle. There are four common pixels on the column (3,1) through (3,4) where the rectangles meet. The common pixels will be considered twice in the painting process, once for each rectangle.

To correct the problem of adjacent vector objects, the PCL5e language implementation provided a choice of pixel placement models: *grid intersection* and *grid-centered*.

The figure below illustrates the concepts of the two models and the actual implementation in



PCL XL.

Assume a filled rectangle in the figure above extends from coordinate position (1,1) to position (3,4) as shown above. As seen before, the grid intersection model of the rectangle comprises

Appendix**A. PCL XL and Pixel Placement**

twelve pixels. In the grid-centered model, the number of rows and columns are each reduced by one, and pixels are placed in the center of the squares, rather than at the intersections. For the same coordinates, the grid-centered model produces a rectangle that is one dot row thinner and one dot row shorter than the grid intersection model. In this model, the problem of overlapping of adjacent vector objects is averted.

Since PCL XL devices image pixels only at the intersections of the grid, the actual PCL XL implementation of the grid-centered model is shown on the right. PCL XL always uses this model for vector objects to avoid the overlapping problem. In the PCL XL grid centered implementation, the resulting bottom and right boundaries of vector objects are reduced by one device pixel to avoid the overlapping problem.

Appendix

B. Binary Stream Tag Values

B. Binary Stream Tag Values

The following table outlines the specific values assigned for attribute list tags, operator tags, and data type tags.

Tag Value	Tag Name	Tag Type	Description
0x00		White Space	Null
0x01 - 0x08			Not Used
0x09 - 0x0d		White Space	HT, LF, VT, FF, CR
0x0e - 0x1f			Not Used
0x20		White Space	Space
0x21 - 0x26			Not Used
0x27			Reserved for beginning of ASCII binding.
0x28			Reserved for beginning of binary binding - high byte first.
0x29			Reserved for beginning of binary binding - low byte first.
0x2a - 0x40			Not Used
0x41	BeginSession	Operator	
0x42	EndSession	Operator	
0x43	BeginPage	Operator	
0x44	EndPage	Operator	
0x45		Operator	Reserved for future use.
0x46		Operator	Reserved for future use.
0x47	Comment	Operator	
0x48	OpenDataSource	Operator	
0x49	CloseDataSource	Operator	
0x4a		Operator	Reserved for future use.
0x4b		Operator	Reserved for future use.
0x4c		Operator	Reserved for future use.
0x4d		Operator	Reserved for future use.
0x4e		Operator	Reserved for future use.
0x4f	BeginFontHeader	Operator	
0x50	ReadFontHeader	Operator	
0x51	EndFontHeader	Operator	
0x52	BeginChar	Operator	
0x53	ReadChar	Operator	
0x54	EndChar	Operator	
0x55	RemoveFont	Operator	
0x56	SetCharAttributes	Operator	Reserved for future use.

Appendix**B. Binary Stream Tag Values**

0x57		Operator	Reserved for future use.
0x58		Operator	Reserved for future use.
0x59		Operator	Reserved for future use.
0x5a		Operator	Reserved for future use.
0x5b	BeginStream	Operator	
0x5c	ReadStream	Operator	
0x5d	EndStream	Operator	
0x5e	ExecStream	Operator	
0x5f	RemoveStream	Operator	
0x60	PopGS	Operator	
0x61	PushGS	Operator	
0x62	SetClipReplace	Operator	
0x63	SetBrushSource	Operator	
0x64	SetCharAngle	Operator	
0x65	SetCharScale	Operator	
0x66	SetCharShear	Operator	
0x67	SetClipIntersect	Operator	
0x68	SetClipRectangle	Operator	
0x69	SetClipToPage	Operator	
0x6a	SetColorSpace	Operator	
0x6b	SetCursor	Operator	
0x6c	SetCursorRel	Operator	
0x6d	SetHalftoneMethod	Operator	
0x6e	SetFillMode	Operator	
0x6f	SetFont	Operator	
0x70	SetLineDash	Operator	
0x71	SetLineCap	Operator	
0x72	SetLineJoin	Operator	
0x73	SetMiterLimit	Operator	
0x74	SetPageDefaultCTM	Operator	
0x75	SetPageOrigin	Operator	
0x76	SetPageRotation	Operator	
0x77	SetPageScale	Operator	
0x78	SetPatternTxMode	Operator	
0x79	SetPenSource	Operator	
0x7a	SetPenWidth	Operator	
0x7b	SetROP	Operator	
0x7c	SetSourceTxMode	Operator	
0x7d	SetCharBoldValue	Operator	
0x7f	SetClipMode	Operator	
0x80	SetPathToClip	Operator	
0x81	SetCharSubMode	Operator	
0x82		Operator	
0x83		Operator	Reserved for future use.
0x84	CloseSubPath	Operator	
0x85	NewPath	Operator	

Appendix**B. Binary Stream Tag Values**

0x86	PaintPath	Operator	
0x87		Operator	Reserved for future use.
0x88		Operator	Reserved for future use.
0x89		Operator	Reserved for future use.
0x8a		Operator	Reserved for future use.
0x8b		Operator	Reserved for future use.
0x8c		Operator	Reserved for future use.
0x8d		Operator	Reserved for future use.
0x8e		Operator	Reserved for future use.
0x8f		Operator	Reserved for future use.
0x90		Operator	Reserved for future use.
0x91	ArcPath	Operator	
0x92		Operator	Reserved for future use.
0x93	BezierPath	Operator	
0x94		Operator	Reserved for future use.
0x95	BezierRelPath	Operator	
0x96	Chord	Operator	
0x97	ChordPath	Operator	
0x98	Ellipse	Operator	
0x99	EllipsePath	Operator	
0x9a		Operator	Reserved for future use.
0x9b	LinePath	Operator	
0x9c		Operator	Reserved for future use.
0x9d	LineRelPath	Operator	
0x9e	Pie	Operator	
0x9f	PiePath	Operator	
0xa0	Rectangle	Operator	
0xa1	RectanglePath	Operator	
0xa2	RoundRectangle	Operator	
0xa3	RoundRectanglePath	Operator	
0xa4		Operator	Reserved for future use.
0xa5		Operator	Reserved for future use.
0xa6		Operator	Reserved for future use.
0xa7		Operator	Reserved for future use.
0xa8	Text	Operator	
0xa9	TextPath	Operator	
0xaa		Operator	Reserved for future use.
0xab		Operator	Reserved for future use.
0xac		Operator	Reserved for future use.
0xad		Operator	Reserved for future use.
0xae		Operator	Reserved for future use.
0xaf		Operator	Reserved for future use.
0xb0	BeginImage	Operator	
0xb1	ReadImage	Operator	
0xb2	EndImage	Operator	
0xb3	BeginRastPattern	Operator	

Appendix**B. Binary Stream Tag Values**

0xb4	ReadRastPattern	Operator	
0xb5	EndRastPattern	Operator	
0xb6	BeginScan	Operator	
0xb7		Operator	Reserved for future use.
0xb8	EndScan	Operator	
0xb9	ScanLineRel	Operator	
0xba - 0xbf			Reserved for future use.
0xc0	ubyte	Data Type	Unsigned 8-bit value
0xc1	uint16	Data Type	Unsigned 16-bit value
0xc2	uint32	Data Type	Unsigned 32-bit value
0xc3	sint16	Data Type	Signed 16-bit value
0xc4	sint32	Data Type	Signed 32-bit value
0xc5	real32	Data Type	Real number value
0xc6		Data Type	Reserved for future use.
0xc7		Data Type	Reserved for future use.
0xc8	ubyte_array	Data Type	Array of Unsigned 8-bit values
0xc9	uint16_array	Data Type	Array of Unsigned 16-bit values
0xca	uint32_array	Data Type	Array of Unsigned 32-bit values
0xcb	sint16_array	Data Type	Array of Signed 16-bit values
0xcc	sint32_array	Data Type	Array of Signed 32-bit values
0xcd	real32_array	Data Type	Array of Real number values
0xce		Data Type	Reserved for future use.
0xcf		Data Type	Reserved for future use.
0xd0	ubyte_xy	Data Type	Two Unsigned 8-bit values
0xd1	uint16_xy	Data Type	Two Unsigned 16-bit values
0xd2	uint32_xy	Data Type	Two Unsigned 32-bit values
0xd3	sint16_xy	Data Type	Two Signed 16-bit values
0xd4	sint32_xy	Data Type	Two Signed 32-bit values
0xd5	real32_xy	Data Type	Two Real number values
0xd6		Data Type	Reserved for future use.
0xd7		Data Type	Reserved for future use.
0xd8		Data Type	Reserved for future use.
0xd9		Data Type	Reserved for future use.
0xda		Data Type	Reserved for future use.
0xdb		Data Type	Reserved for future use.
0xdc		Data Type	Reserved for future use.
0xdd		Data Type	Reserved for future use.
0xde		Data Type	Reserved for future use.
0xdf		Data Type	Reserved for future use.
0xe0	ubyte_box	Data Type	Four Unsigned 8-bit values
0xe1	uint16_box	Data Type	Four Unsigned 16-bit values
0xe2	uint32_box	Data Type	Four Unsigned 32-bit values
0xe3	sint16_box	Data Type	Four Signed 16-bit values
0xe4	sint32_box	Data Type	Four Signed 32-bit values

Appendix**B. Binary Stream Tag Values**

0xe5	real32_box	Data Type	Four Real number values
0xe6		Data Type	Reserved for future use.
0xe7		Data Type	Reserved for future use.
0xe8		Data Type	Reserved for future use.
0xe9		Data Type	Reserved for future use.
0xea		Data Type	Reserved for future use.
0xeb		Data Type	Reserved for future use.
0xec		Data Type	Reserved for future use.
0xed		Data Type	Reserved for future use.
0xee		Data Type	Reserved for future use.
0xef		Data Type	Reserved for future use.
0xf0 - 0xf7			Reserved for future use.
0xf8	attr_ubyte	Attribute	Unsigned, 8-bit Attribute
0xf9	attr_uint16	Attribute	Unsigned, 16-bit Attribute
0xfa	dataLength	Embed Data	Embedded Data Follows
0xfb	dataLengthByte	Embed Data	Embedded Data Follows (0-255 bytes)
0xfc - 0xff			Reserved for future use.

Appendix

C. Operator Name to Tag Value Table

C. Operator Name to Tag Value Table

The following table outlines the binary stream tag values for PCL XL operators.

Tag Name	Tag Value
ArcPath	0x91
BeginChar	0x52
BeginFontHeader	0x4f
BeginImage	0xb0
BeginPage	0x43
BeginRastPattern	0xb3
BeginScan	0xb6
BeginSession	0x41
BeginStream	0x5b
BezierPath	0x93
BezierRelPath	0x95
Chord	0x96
ChordPath	0x97
CloseDataSource	0x49
CloseSubPath	0x84
Comment	0x47
Ellipse	0x98
EllipsePath	0x99
EndChar	0x54
EndFontHeader	0x51
EndImage	0xb2
EndPage	0x44
EndRastPattern	0xb5
EndScan	0xb8
EndSession	0x42
EndStream	0x5d
ExecStream	0x5e
LinePath	0x9b
LineRelPath	0x9d
NewPath	0x85
OpenDataSource	0x48
PaintPath	0x86
Pie	0x9e
PiePath	0x9f
PopGS	0x60
PushGS	0x61
ReadChar	0x53
ReadFontHeader	0x50

ReadImage	0xb1
ReadRastPattern	0xb4
ReadStream	0x5c
Rectangle	0xa0
RectanglePath	0xa1
RemoveFont	0x55
SetCharAttributes	0x56
RemoveStream	0x5f
RoundRectangle	0xa2
RoundRectanglePath	0xa3
ScanLineRel	0xb9
SetClipReplace	0x62
SetBrushSource	0x63
SetCharAngle	0x64
SetCharBoldValue	0x7d
SetCharScale	0x65
SetCharShear	0x66
SetCharSubMode	0x81
SetClipIntersect	0x67
SetClipMode	0x7f
SetClipRectangle	0x68
SetClipToPage	0x69
SetColorSpace	0x6a
SetCursor	0x6b
SetCursorRel	0x6c
SetHalftoneMethod	0x6d
SetFillMode	0x6e
SetFont	0x6f
SetLineCap	0x71
SetLineDash	0x70
SetLineJoin	0x72
SetMiterLimit	0x73
SetPageDefaultCTM	0x74
SetPageOrigin	0x75
SetPageRotation	0x76
SetPageScale	0x77
SetPathToClip	0x80
SetPatternTxMode	0x78
SetPenSource	0x79
SetPenWidth	0x7a
SetROP	0x7b
SetSourceTxMode	0x7c
Text	0xa8
TextPath	0xa9

Appendix**D. Data Type to Tag Value Table**

D. Data Type to Tag Value Table

The following table outlines the binary stream tag values for specific PCL XL data types.

Tag Name	Tag Value
attr_ubyte	0xf8
attr_uint16	0xf9
embedded_data	0xfa
embedded_data_byte	0xfb
real32	0xc5
real32_array	0xcd
real32_box	0xe5
real32_xy	0xd5
sint16	0xc3
sint16_array	0xcb
sint16_box	0xe3
sint16_xy	0xd3
sint32	0xc4
sint32_array	0xcc
sint32_box	0xe4
sint32_xy	0xd4
ubyte	0xc0
ubyte_array	0xc8
ubyte_box	0xe0
ubyte_xy	0xd0
uint16	0xc1
uint16_array	0xc9
uint16_box	0xe1
uint16_xy	0xd1
uint32	0xc2
uint32_array	0xca
uint32_box	0xe2
uint32_xy	0xd2

Note: Real numbers are represented in standard 32-bit IEEE single-precision floating point format.

Appendix

E. Attribute ID Number to Attribute Name Table

E. Attribute ID Number to Attribute Name Table

The following table outlines the attribute ID numbers for PCL XL attributes in binary streams.

Attribute ID	Attribute Name
1.	
2.	PaletteDepth
3.	ColorSpace
4.	NullBrush
5.	NullPen
6.	PaletteData
7.	
8.	PatternSelectID
9.	GrayLevel
10.	Lightness
11.	RGBColor
12.	PatternOrigin
13.	NewDestinationSize
14.	PrimaryArray
15.	PrimaryDepth
16.	Saturation
17.	ColorimetricColorSpace
18.	XYChromaticities
19.	WhiteReferencePoint
20.	CRGBMixMax
21.	GammaGain
22.	
23.	
24.	
25.	
26.	
27.	
28.	
29.	
30.	
31.	
32.	
33.	DeviceMatrix
34.	DitherMatrixDataType
35.	DitherOrigin
36.	
37.	MediaSize
38.	MediaSource

39.	MediaType
40.	Orientation
41.	PageAngle
42.	PageOrigin
43.	PageScale
44.	ROP3
45.	TxMode
46.	
47.	CustomMediaSize
48.	CustomMediaSizeUnits
49.	PageCopies
50.	DitherMatrixSize
51.	DitherMatrixDepth
52.	SimplexPageMode
53.	DuplexPageMode
54.	DuplexPageSide
55.	
56.	
57.	
58.	
59.	
60.	
61.	
62.	
63.	
64.	
65.	ArcDirection
66.	BoundingBox
67.	DashOffset
68.	EllipseDimension
69.	EndPoint
70.	FillMode
71.	LineCapStyle
72.	LineJoinStyle
73.	MiterLength
74.	LineDashStyle
75.	PenWidth
76.	Point
77.	NumberOfPoints
78.	SolidLine
79.	StartPoint
80.	PointType
81.	ControlPoint1
82.	ControlPoint2
83.	ClipRegion
84.	ClipMode
85.	

Appendix**E. Attribute ID Number to Attribute Name Table**

86.		133.	
87.		134.	Measure
88.		135.	
89.		136.	SourceType
90.		137.	UnitsPerMeasure
91.		138.	
92.		139.	StreamName
93.		140.	StreamDataLength
94.		141.	
95.		142.	
96.		143.	ErrorReport
97.		144.	
98.	ColorDepth	145.	Reserved
99.	BlockHeight	146.	Reserved
100.	ColorMapping	147.	Reserved
101.	CompressMode	148.	Reserved
102.	DestinationBox	149.	Reserved
103.	DestinationSize	150.	Reserved
104.	PatternPersistence	151.	Reserved
105.	PatternDefineID	152.	Reserved
106.		153.	Reserved
107.	SourceHeight	154.	Reserved
108.	SourceWidth	155.	Reserved
109.	StartLine	156.	Reserved
110.	PadBytesMultiple	157.	Reserved
111.	BlockByteLength	158.	Reserved
112.		159.	Reserved
113.		160.	
114.		161.	CharAngle
115.	NumberOfScanLines	162.	CharCode
116.		163.	CharDataSize
117.		164.	CharScale
118.		165.	CharShear
119.		166.	CharSize
120.		167.	FontHeaderLength
121.		168.	FontName
122.		169.	FontFormat
123.		170.	SymbolSet
124.		171.	TextData
125.		172.	CharSubModeArray
126.		173.	
127.		174.	
128.		175.	XSpacingData
129.	CommentData	176.	YSpacingData
130.	DataOrg	177.	CharBoldValue
131.		178.	
132.		179.	

Appendix**F. Attribute Name to Data Types Table****F. Attribute Name to Data Types Table**

The following table outlines the attribute ID numbers for PCL XL attributes in binary streams.

Attribute Name	Valid Value(s)	Data Types	Attribute #	Class.rev #
ArcDirection	ArcDirection Enumeration	ubyte	65	1.1
BlockByteLength	0 - (2 ³² - 1)	uint32	111	2.0
BlockHeight	0 - 65535	uint16	99	1.1
BoldValue	0.0 - 1.0	real32	177	1.1
BoundingBox	Range of data types	ubyte_box uint16_box sint16_box	66	1.1
CharAngle	-360 <= CharAngle <= 360	uint16 sint16 real32	161	1.1
CharCode	Any value within Range of data types	ubyte uint16	162	1.1
CharDataSize	0 - 65535	uint16	163	1.1
CharScale	Two numeric values less than or equal to 32767.0 and greater than or equal to -32768.0, excluding a value of zero.	ubyte_xy uint16_xy real32_xy	164	1.1
CharShear	Two numeric values within Range of data types and less than 32767.0 and greater than -32768.0	ubyte_xy uint16_xy sint16_xy real32_xy	165	1.1
CharSize	Any value within Range of data types and greater than zero.	ubyte uint16 real32	166	1.1
CharSubModeArray	CharSubModeArray Enumeration	ubyte_array	172	1.1
ClipMode	ClipMode Enumeration	ubyte	84	1.1
ClipRegion	ClipRegion Enumeration	ubyte	83	1.1
ColorDepth	ColorDepth Enumeration	ubyte	98	1.1
ColorimetricColorSpace	ColorimetricColorSpace Enum.	ubyte	17	2.0
ColorMapping	ColorMapping Enumeration	ubyte	100	1.1
ColorSpace	ColorSpace Enumeration	ubyte	3	1.1
CommentData	Any data	ubyte_array uint16_array	129	1.1
CompressMode	Compress Mode Enumeration	ubyte	101	1.1

Appendix**F. Attribute Name to Data Types Table**

ControlPoint1	Range of data types	ubyte_xy uint16_xy sint16_xy	81	1.1
ControlPoint2	Range of data types	ubyte_xy uint16_xy sint16_xy	82	1.1
CRGBMinMax	Range of data types	real32_array	20	2.0
CustomMediaSize	dimensions of physical custom media size	uint16_xy real32_xy	47	1.1
CustomMediaSizeUnits	Measure Enumeration	ubyte	48	1.1
DashOffset	Range of data types	ubyte uint16 sint16	67	1.1
DataOrg	DataOrg Enumeration	ubyte	130	1.1
DestinationBox	Four numeric values:	uint16_box	102	1.1
DestinationSize	Two numeric values; x nor y equal to zero	uint16_xy	103	1.1
DeviceMatrix	DitherMatrix Enumeration	ubyte	33	1.1
DitherMatrixDataType	eUByte	ubyte	34	1.1
DitherMatrixDepth	e8Bit	ubyte	51	1.1
DitherMatrixSize	1-256	uint16_xy	50	1.1
DitherOrigin	Range of data types	ubyte_xy uint16_xy sint16_xy	35	1.1
DuplexPageMode	DuplexPageMode enumeration	ubyte	53	1.1
DuplexPageSide	DuplexPageSide enumeration	ubyte	54	1.1
EllipseDimension	Range of data types	ubyte_xy uint16_xy	68	1.1
EndPoint	Range of data types	ubyte_xy uint16_xy sint16_xy	69	1.1
ErrorReport	ErrorReport Enumeration	ubyte	143	1.1
FillMode	FillMode Enumeration	ubyte	70	1.1
FontFormat	0	ubyte	169	1.1
FontHeaderLength	0 - 65535	uint16	167	1.1
FontName	Valid Font Name	ubyte_array uint16_array	168	1.1
GammaGain	range of data types	real32_array	21	2.0
GrayLevel	0 - 1.0 if real or range of integer data type	real32 ubyte	9	1.1
Lightness	range of data types	real32	10	2.0
LineCapStyle	LineCap Enumeration	ubyte	71	1.1

Appendix

F. Attribute Name to Data Types Table

LineDashStyle	Range of data types Maximum array size = MAXDASHES (device- dependent)	ubyte_array uint16_array sint16_array	74	1.1
LineJoinStyle	LineJoin Enumeration	ubyte	72	1.1
Measure	Measure Enumeration	ubyte	134	1.1
MediaSize	MediaSize Enumeration	ubyte	37	1.1
MediaSource	MediaSource Enumeration	ubyte	38	1.1
MiterLength	range of data types	ubyte uint16	73	1.1
NewDestinationSize	range of data types; neither x nor y equal to zero	uint16_xy	13	1.1
NullBrush	0	ubyte	4	1.1
NullPen	0	ubyte	5	1.1
NumberOfPoints	Range of data types	ubyte uint16	77	1.1
NumberOfScanLines	Range of data types	uint16	115	1.1
Orientation	Orientation Enumeration	ubyte	40	1.1
PadBytesMultiple	1-255	ubyte	110	2.0
PageAngle	Positive or Negative multiples of 90: -360, -270, -180, -90, 0, 90, 180, 270, 360	uint16 sint16	41	1.1
PageCopies	range of data types; zero causes the page not to be imaged	uint16	49	1.1
PageOrigin	Range of data types	ubyte_xy uint16_xy sint16_xy	42	1.1
PageScale	Two numeric values less than or equal to 32767.0 and greater than or equal to 0.	ubyte_xy uint16_xy real32_xy	43	1.1
PaletteData	Any data in the range of the array elements. If eGray, array lengths of 2, 16, and 256 are allowed. If eRGB, eCRGB or eSRGB array lengths of 6, 48, 768 are allowed.	ubyte_array	6	1.1
PaletteDepth	ColorDepth Enumeration	ubyte	2	1.1

Appendix**F. Attribute Name to Data Types Table**

PatternDefineID	Range of data types	sint16	105	1.1
PatternOrigin	Range of data types	sint16_xy	12	1.1
PatternPersistence	PatternPersistence Enumeration	ubyte	104	1.1
PatternSelectID	Range of data types	sint16	8	1.1
PenWidth	Range of data types: zero or more	ubyte uint16	75	1.1
Point	Two numeric values within Range of data types	ubyte_xy uint16_xy sint16_xy	76	1.1
PointType	DataType Enumeration	ubyte	80	1.1
PrimaryArray	Color data for colorspace	real32_array ubyte_array	14	2.0
PrimaryDepth	Primary Depth Enumeration	ubyte	15	2.0
RGBColor	Three values: 0 - 1.0 if real or range of integer data type	real32_array ubyte_array	11	1.1
ROP3	range of data types	ubyte	44	1.1
Saturation	range of data types	real32	16	2.0
SimplexPageMode	SimplexPageMode Enumeration	ubyte	52	1.1
SolidLine	0	ubyte	78	1.1
SourceHeight	1 - 65535	uint16	107	1.1
SourceType	DataSource Enumeration	ubyte	136	1.1
SourceWidth	1 - 65535	uint16	108	1.1
StartLine	0 - 65535	uint16	109	1.1
StartPoint	Range of data types	ubyte_xy uint16_xy sint16_xy	79	1.1
StreamDataLength	Range of data types	uint32	140	1.1
StreamName	ASCII character string	ubyte_array uint16_array	139	1.1
SymbolSet	SymbolSet Enumeration	uint16	170	1.1
TextData	Any character codes in range of data types	ubyte_array uint16_array	171	1.1
TxMode	TxMode enumeration	ubyte	45	1.1
UnitsPerMeasure: xUnits, yUnits	two positive numeric values maximum of 65535.0	uint16_xy real32_xy	114	1.1
WhitePointReference	range of data types	real32_array	19	2.0
WritingMode	WritingMode enumeration	Ubyte	173	2.0
XSpacingData	Range of data types—must be same size as TextData array	ubyte_array uint16_array sint16_array	175	1.1
YSpacingData	Range of data types—must be same size as TextData array	ubyte_array uint16_array sint16_array	176	1.1
XYChromaticities	range of data types	real32_array	18	2.0

Appendix

G. Attribute Value Enumerations Table

G. Attribute Value Enumerations Table

The following are values used for enumerated data types through out this document. These enumerations are standardized across protocol classes. All values are listed in decimal form.

Attribute Name / Enumeration	Val	Class
ArcDirection		
eClockWise	0	1.1
eCounterClockWise	1	1.1
CharSubModeArray		
eNoSubstitution	0	1.1
eVerticalSubstitution	1	1.1
ClipMode <i>see FillMode Enumeration</i>		
ClipRegion		
eInterior	0	1.1
eExterior	1	1.1
ColorDepth		
e1Bit	0	1.1
e4Bit	1	1.1
e8Bit	2	1.1
ColorimetricColorSpace		
eCRGB	5	2.0
ColorMapping		
eDirectPixel	0	1.1
eIndexedPixel	1	1.1
ColorSpace		
eGray	1	1.1
eRGB	2	1.1
eSRGB	6	2.0
CompressMode		
eNoCompression	0	1.1
eRLECompression	1	1.1

eJPEGCompression	2	2.0
DataOrg		
eBinaryHighByteFirst	0	1.1
eBinaryLowByteFirst	1	1.1
DataSource		
eDefault	0	1.1
DataType		
eUByte	0	1.1
eSByte	1	1.1
eUInt16	2	1.1
eSint16	3	1.1
DitherMatrix		
eDeviceBest	0	1.1
DuplexPageMode		
eDuplexHorizontalBinding	0	1.1
eDuplexVerticalBinding	1	1.1
DuplexPageSide		
eFrontMediaSide	0	1.1
eBackMediaSide	1	1.1
ErrorReport		
eNoReporting	0	1.1
eBackChannel (BackCh)	1	1.1
eErrorPage (ErrPage)	2	1.1
eBackChAndErrPage	3	1.1
eNWBackChannel	4	2.0
eNWErrorPage	5	2.0
eNWBackChAndErrPage	6	2.0
FillMode		
eNonZeroWinding	0	1.1
eEvenOdd	1	1.1
LineCap		
eButtCap	0	1.1
eRoundCap	1	1.1
eSquareCap	2	1.1
eTriangleCap	3	1.1

Appendix

G. Attribute Value Enumerations Table

LineJoin		
eMiterJoin	0	1.1
eRoundJoin	1	1.1
eBevelJoin	2	1.1
eNoJoin	3	1.1
Measure		
eInch	0	1.1
eMillimeter	1	1.1
eTenthsOfAMillimeter	2	1.1
MediaSize Enumerations		
eLetterPaper	0	1.1
eLegalPaper	1	1.1
eA4Paper	2	1.1
eExecPaper	3	1.1
eLedgerPaper	4	1.1
eA3Paper	5	1.1
eCOM10Envelope	6	1.1
eMonarchEnvelope	7	1.1
eC5Envelope	8	1.1
eDLEnvelope	9	1.1
eJB4Paper	10	1.1
eJB5Paper	11	1.1
eB5Envelope	12	1.1
eJPostcard	14	1.1
eJDoublePostcard	15	1.1
eA5Paper	16	1.1
eA6Paper	17	2.0
eJB6Paper	18	2.0
MediaSource		
eDefaultSource	0	1.1
eAutoSelect	1	1.1
eManualFeed	2	1.1
eMultiPurposeTray	3	1.1
eUpperCassette	4	1.1
eLowerCassette	5	1.1
eEnvelopeTray	6	1.1
eThirdCassette	7	2.0
External Trays 1-248	8-255	2.0
MediaDestination		
eDefaultDestination	0	2.0
eFaceDownBin	1	2.0
eFaceUpBin	2	2.0
eJobOffsetBin	3	2.0

External Bins ^{**} 1-251	5-255	
Orientation		
ePortraitOrientation	0	1.1
eLandscapeOrientation	1	1.1
eReversePortrait	2	1.1
eReverseLandscape	3	1.1
PatternPersistence		
eTempPattern	0	1.1
ePagePattern	1	1.1
eSessionPattern	2	1.1
SymbolSet		
<i>See Appendix O.</i>		
SimplexPageMode		
eSimplexFrontSide	0	1.1
TxMode		
eOpaque	0	1.1
eTransparent	1	1.1
WritingMode		
eHorizontal	0	2.0
eVertical	1	2.0

* External input trays 1 through 248 are selected by substituting the values 8 through 255 for the enumerated values. Example, 8 = first external input tray, 9 = second external input tray, etc.

** External output bins 1 through 251 are selected by substituting the values 5 through 255 for the enumerated values. Example, 5 = first external output bin, 6 = second external output bin, etc.

H. HP LaserJet Printer PJJ and PCL XL Streams

All PCL XL sessions on HP LaserJet Printer devices must be explicitly invoked through a PJJ ENTER LANGUAGE command. There may be only one currently executing PCL XL imaging session in HP LaserJet Printers. Following the EndSession in the stream, the PCL XL session must end with a Universal End of Language command (UEL).

An example is illustrated below (see Hewlett-Packard's [Printer Job Language Technical Reference Manual](#) for more information).

Example:

Invoke a PCL XL Job that will consist of a single, three page session:

```
<esc>%-12345X
@PJJ ENTER LANGUAGE = PCLXL <CR><LF>
...<PCL XL stream header>...
... begin session ...
... page 1 ...
... page 2 ...
... page 3 ...
... end session ...
<esc>%-12345X
```

The PCL XL imaging protocol provides commands that define the context of a session, or a series of page descriptions. The purpose of the session commands is two-fold. First, the **BeginSession** operator provides a mechanism to specify attributes that will span all pages. Second, the **Begin/EndSession** concept provides an encapsulation of the page descriptions so that the PCL XL protocol may be implemented in an environment other than HP LaserJet Printer PJJ, where job definition is not explicitly defined.

As stated above, all PCL XL sessions must be invoked through PJJ. Immediately following the PJJ ENTER LANGUAGE command (that is terminated with a line feed character) is the PCL XL stream header. When all pages have been described, the PCL XL session is ended with an EndSession operator. A UEL should follow immediately after the **EndSession** operator, thus ending the job. Note: if several sessions are wrapped with a single pair of UEL strings and the first job causes an error, all successive sessions wrapped by the UEL will be flushed.

Appendix**H. LaserJet Printer Job Language and PCL XL Streams**

The PCL XL language in HP LaserJet Printers will read and honor the following variables that can be set through PJJL.

DEFAULT_COPIES	Number of copies of each page that will be printed.
DEFAULT_PLEX_MODE	Duplexing enabled or disabled.
DEFAULT_PLEX_BINDING	Duplex binding mode of a job.
DEFAULT_MANUAL_FEED	Manual Feed mode.

I. HP LaserJet Printer Engine/Paper Handling Features and PCL XL

The attributes that influence how media is selected and moved through a printer can be divided into two groups. These groups are: attributes that have meaning to the entire print job, and attributes that have meaning to the current page. These two groups of attributes are listed below. Note that some of the attributes listed below may not be supported by all print engines.

Job (PJL) Attributes:

COPIES Sets the number of uncollated copies for each page of the job (may be overridden by the Copies attribute of the EndPage operator).

RESOLUTION Sets the print resolution to be used to render each page of the job.

DUPLEX Sets the default mode to enable/disable printing on both sides of the paper (may be overridden by the duplexing attributes of the BeginPage operator).

BINDING Sets the default relationship of the front and back images on the pages printed in duplex mode (may be overridden by the duplexing attributes of the BeginPage operator).

JOB OFFSET Sets the current Job Offset mode.

MANUALFEED Sets the manual feed mode. Note: Manual feed is a special case, see below.

Page Attributes:

Orientation Sets the orientation of the page.

MediaSize Sets the requested MediaSize for the page.

MediaSource Sets the requested Source from which to draw media for the page.

MediaDestination Sets the requested Destination at which to place the media following page rendering.

MediaType Sets the requested Type of media to be used for the page.

In general, if a user desires to set or change Job Attributes through the data stream, this should be done through PJL, prior to the invocation of the PCL XL personality. The special case to this rule is MANUALFEED, which is discussed below.

Appendix**I. LaserJet Engine/Paper Handling Features and PCL XL**

The user is required to specify certain Page Attributes within the PCL XL protocol, and if desired, request certain other attributes. The specification of Page Attributes should be envisioned as requests to the printer engine. Every attempt will be made to satisfy the users request. The following is a list of conditions that will be analyzed for each page to be printed.

1. If the printer engine can satisfy the requests, the page will be printed according to the users page attribute specifications.
2. If the user makes a valid request for a given set of page attributes, but the printer engine can not satisfy the request (due to hardware or front panel configuration), the user will be prompted to satisfy the request (by inserting other media, sources, etc.).
3. The user is required to specify **Orientation** and **MediaSize** for each page to be printed. If the user specifies invalid requests, an error or warning will be raised.
4. The user may optionally specify **MediaSource**, **MediaDestination**, **MediaType**, and **PageCopies** (if the printer engine supports these features). If the user does not specify these attributes, or if the user specifies invalid request for these attributes, the printer defaults will be used and a warning issued. Note that the user may specify **MANUALFEED** as a page attribute, by requesting **MANUALFEED** as the desired media source.
5. If the user specifies Media Selection attributes (i.e. **MediaSource**, **MediaDestination**, and **MediaType**) for a given page, all future pages of that PCL XL session will inherit these attributes until the Media Selection attribute(s) are specified again. Note that it is possible for the default state of all optional page attributes to be explicitly requested by the user.
6. If the user does not specify a **MediaSource**, and if **MANUALFEED** has been enabled through PJI (or the front panel), the job will be manually fed. **MANUALFEED** will be ignored if the user specifies any value for **MediaSource**.
7. If the user specifies the **PageCopies** attribute, that value will be used to determine how many copies of the current page to print. Only the current page will be effected by the **PageCopies** attribute. If the **PageCopies** attribute is not specified, a single page will be printed. If the **PageCopies** attribute is set to zero, the page will not print.

Appendix**I. LaserJet Engine/Paper Handling Features and PCL XL****Duplex on LaserJet PCL XL Devices**

Duplex nomenclature:

Page: A series of marks or commands that define a single side of a printed page.

Sheet: A physical sheet of media, that contains a front side and a back side.

The behavior of duplex operation is described here:

1. The duplex feature can only be enabled if the duplexing hardware is installed.
2. Duplex is enabled through the front panel setting, or by setting the PJI DUPLEX variable, or by associated BeginPage attributes.
3. The duplex binding mode is enabled through the front panel setting, or by setting the PJI BINDING variable, or by BeginPage attributes.
4. A media source requested that is made with duplex enabled will be processed the same as if duplex is not enabled.
5. If duplex is enabled, the MediaSource, MediaDestination, MediaSize, and MediaType for both the front and back pages of a sheet should be identical. If these attributes for two adjacent pages differ, such that the second page was bound for the back side of the first, the first page will be ejected as a separate sheet (with the back side blank). In this condition, the next page (previously bound for the back side) would be printed on the front side of the next sheet.
6. Not all Custom Media Size pages can be duplexed. If a page is received that utilizes the CustomMediaSize setting that the engine cannot duplex; it will be printed simplex.
7. If a session is ending and if the front side of a duplex sheet has been printed. The sheet will be printed with a blank back side.

See the **BeginPage** operator for more information.

J. PCL XL Device Error Reporting

Errors can be reported to the user during the course of a PCL XL session. These error reports can take on several forms. The error report can be sent to the user through the I/O stream (sometimes referred to as “Back-Channel”) if bi-directional I/O is supported. The user can also be informed about the error through the use of a printed error report.

The user is allowed to select the type of error reporting desired at **BeginSession** time. The user can set the **ErrorReport** attribute for the **BeginSession** operator to one of the selections in the **errorReportEnumeration** (see **BeginSession** operator description). Warnings may also be reported. Warnings are described at the end of this section.

An error report will show the following information:

1. The subsystem that indicated the error.
2. A textual error code describing the error condition.
3. The name of the operator being processed at the time of the error.
4. The operator position in the stream where the error occurred. “1” is the position of the first operator in a stream, “2” is the position of the second operator in the stream, and so on. Attribute/Value pairs do not increment the position number.

The following is an example of an error report:

PCL XL Error

```

Subsystem:   KERNEL
Error:      IllegalOperatorSequence
Operator:   SetColorSpace
Position:   52

```

Textual error codes such as “IllegalOperatorSequence” are reported when the most common user or simple user errors are detected. Numeric error codes (in hex form) are reported for the more complex error conditions. When numeric codes are reported they will be preceded by the text “InternalError.” Internal error codes must be resolved through communication with factory support personnel. All font and character download errors involving problems with data format are reported as internal errors.

Appendix**J. PCL XL Error and Warning Codes**

When a failure occurs during a user-defined stream, there will be two position numbers reported, separated by a semicolon (“;”). The first position number is the position of the **ExecStream** operator in the parent stream. The second position number following the semicolon is the location of the operator in error in the user-defined stream.

When a Fatal error occurs in a HP LaserJet Printer device, the entire job will be flushed. The job will be flushed until a <UEL> is encountered. When multiple errors occur during a session, the error with the highest priority will be stored; all lower priority errors will be lost.

Warnings are logged internally by PCL XL at the time they occur and the job will be allowed to continue. At the end of the session, a list of warnings will be reported to the user. The form of the warning report is similar to the standard error report format except that operator positions are not logged and reported.

If an error occurs after warnings have been logged, only the error will be reported.

K. PCL XL Error and Warning Codes

GENERIC OPERATOR ERRORS

PCL XL reports several error codes that apply to more than one operator in a general way. Generic operator error codes are described in this section.

Generic Errors

IllegalOperatorSequence	PCL XL read an operator that is out-of-sequence according to the legal sequencing of operators defined by the protocol. For example, a <i>ReadImage</i> occurring prior to a <i>BeginImage</i> yields this error.
IllegalTag	PCL XL had expected to read a data tag or operator tag and instead read something that is undefined for the current version of PCL XL.
InsufficientMemory	The amount of memory required to complete the current operation is unavailable.
InternalOverflow	Completing the current operation requires an amount of one or more internal resources that exceeds the maximum allowed.

Generic Attribute Errors

IllegalArraySize	PCL XL may yield this error for any array that cannot be zero through infinity in length (see the Attribute ID table in the appendix for valid array sizes).
IllegalAttribute	An attribute provided to an operator is not valid for that operator. For example, the <i>BeginPage</i> operator given a <i>Point</i> attribute yields this error.
IllegalAttributeCombination	An operator has two or more attributes that either conflict or do not make sense when presented together. For example, providing <i>BeginPage</i> with both the <i>MediaSize</i> and <i>CustomMediaSize</i> attribute yields this error.
IllegalAttributeDataType	The data type of the data value provided to an attribute is not valid for that attribute. For example, the single-value “ <i>sint16</i> ” data type is not valid for the <i>Point</i> attribute which requires two values: one each for the x and y coordinates of the point. The “ <i>sint16_xy</i> ” data type is valid for the <i>Point</i> attribute.
IllegalAttributeValue	PCL XL will yield this error if an attribute value given is out-of-range for the attribute value expected.

Appendix**K. PCL XL Error and Warning Codes*****Generic Attribute Errors (continued)***

MissingAttribute	An operator with required attributes is missing one or more of the required attributes. For example, <i>SetCursor</i> without the <i>Point</i> attribute yields this error.
------------------	---

Generic Cursor Errors

CurrentCursorUndefined	An operator needing the current cursor position in the current path found that there was no current path and therefore no current cursor position. For example, <i>SetCursorRel</i> yields this error if performed immediately following a <i>NewPath</i> operator.
------------------------	---

Generic Font Errors

NoCurrentFont	An operator needing a valid current font found that no font is set in the current graphics state. For example, attempting to perform the <i>Text</i> or <i>TextPath</i> operators without a successful <i>SetFont</i> operation for the current graphics state yields this error.
BadFontData	The font and/or character description data for the current font operation is incompatible with the current font operation. For example, attempting to perform character scaling on a bitmap characters by setting <i>CharScale</i> to something other than 1.0 will yield this error.

Generic Data Source Errors

DataSourceNotOpen	An operator that requires data from the data source yields this error when no data source is open.
ExtraData	An operator reading the <i>default</i> data source and finding that there is more data than required for the operation yields this error.
IllegalDataLength	An operator reading the <i>default</i> data source and finding that the length of data is not either a required constant length nor a required multiple of a constant length yields this error. For example, the <i>BezierPath</i> operator yields this error if the length of data in the default data source is not a multiple of 3.
IllegalDataValue	An operator that reads the data source and finds that a field in the data source contains an illegal value for that operator yields this error. For example, an illegal value in the x-pair type field for the <i>ScanLineRel</i> operator yields this error.
MissingData	An operator reading the data source and finding that there is less data than is required to complete the operation yields this error.

Appendix

K. PCL XL Error and Warning Codes

OPERATOR-SPECIFIC ERRORS

PCL XL reports several error codes that are specific to an operator. Specific operator errors are described in this section.

BeginChar Errors

CannotReplaceCharacter	An attempt was made to download a character to an internal font or to a font on a mass-storage device.
FontUndefined	The user is attempting to start a character definition with <i>BeginChar</i> , but the font name specified is undefined.

BeginFontHeader Errors

FontNameAlreadyExists	The font name given already exists. The font must be removed before re-defining the header.
-----------------------	---

BeginImage/BeginRastPattern Errors

ImagePaletteMismatch	The palette is too small or too large for the ColorDepth specified for the indexed image or pattern.
MissingPalette	There is no palette for the indexed image/pattern.

BeginPage Warnings

IllegalMediaSize	The values for MediaSize or CustomMediaSize or CustomMediaSizeUnits are out of range for the current device.
IllegalMediaSource	The value for MediaSource is out of range for the current device or device configuration.
IllegalMediaDestination	The value for MediaDestination is out of range for the current device or device configuration.
IllegalOrientation	The value for page orientation is out of range for the current device.

ExecStream Errors

StreamUndefined	An attempt was made to execute an undefined stream.
-----------------	---

OpenDataSource Errors

DataSourceNotClosed	A data source is already open (not closed).
---------------------	---

PushGS Errors

MaxGSLevelsExceeded	The maximum number of graphics state save levels has been exceeded (max = 19).
---------------------	--

Appendix**K. PCL XL Error and Warning Codes****ReadChar Errors**

FSTMismatch	The value of the character format field is inconsistent with the value of the Font Scaling Technology field in the corresponding font header.
UnsupportedCharacter Class	The value of the character class field represents a character class that is not supported by the current version of PCL XL.
UnsupportedCharacter Format	The value of the character format field represents a character format that is not supported by the current version of PCL XL.
IllegalCharacterData	The character data is in a format unrecognized by PCL XL.

ReadFontHeader Errors

IllegalFontData	The font header data is in a format unrecognized by PCL XL.
IllegalFontHeaderFields	The font header Font Format, Font Scaling Technology, Variety, and/or Orientation fields contain invalid values.
IllegalNullSegmentSize	The size of the Null segment in the font header is invalid.
IllegalFontSegment	A segment included in the font header is not defined for the corresponding font scaling technology.
MissingRequired Segment	A segment required by the corresponding font scaling technology is missing.
IllegalGlobalTrueType Segment	The global TrueType segment is in a format unrecognized by PCL XL.
IllegalGalleyCharacter Segment	The galley character segment is in a format unrecognized by PCL XL.
IllegalVerticalTx Segment	The vertical transformation segment is in a format unrecognized by PCL XL.
IllegalBitmapResolution Segment	The bitmap resolution segment is in a format unrecognized by PCL XL.

RemoveFont Warnings

UndefinedFontNot Removed	An attempt was made to remove an undefined font.
InternalFontNot Removed	An attempt was made to remove an internal font.
MassStorageFontNot Removed	An attempt was made to remove a font on a mass storage device.

Appendix**K. PCL XL Error and Warning Codes****RemoveStream Warnings**

UndefinedStreamNot Removed	An attempt was made to remove an undefined stream.
InternalStreamNot Removed	An attempt was made to remove an internal stream.
MassStorageStreamNot Removed	An attempt was made to remove a stream on a mass storage device.

SetBrushSource/SetPenSource Errors

ColorSpaceMismatch	The color attribute for SetPen/SetBrush (i.e. RGBColor or GrayLevel) does not match the ColorSpace at the current graphics state level. The ColorSpace was either set by default (at BeginPage) or by the most recent SetColorSpace operator.
RasterPatternUndefined	The value given for PatternSelectID does not identify a raster pattern.

SetClipReplace/SetClipIntersect/SetClipRectangle Errors

ClipModeMismatch	The ClipRegion attribute is “eExterior” and the ClipMode is “eNonZeroWinding.” When the ClipRegion is “eExterior” the ClipMode must be “eEvenOdd.”
------------------	--

SetFont Errors

FontUndefined NoSubstituteFound SymbolSetRemap Undefined	An attempt was made to set a font that was not defined in the device. A font substitution could not be found by PCL XL. The font is defined, but unusable with the requested symbol set.
---	--

SetFont Warnings

“font name” substituted for “font name”	An attempt was made to set a font that was undefined in the device. A substitution was found and made by PCL XL.
--	--

Stream Header Errors

UnsupportedBinding	PCL XL does not support the binding requested in the stream header.
UnsupportedClassName	The class name is unknown to PCL XL (e.g. “PCL-XY”).
UnsupportedProtocol	The protocol class number and/or revision is unsupported by PCL XL.
IllegalStreamHeader	The stream header is in a format unrecognized by PCL XL.

L. PCL XL Font Formats and Font Definition Operators

PCL XL supports TrueType and bitmap font support for protocol class 1.1.

For non-volatile fonts, class 1.1 supports HP LaserJet Printer SX, 300 and 305 format font entities that are stored in mass storage or ROM.

PCL XL supports TrueType and bitmap fonts downloaded by applications on an on-demand basis.

PCL XL supports a single TrueType and bitmap font header format, leveraged from PCL Format 16 for downloading fonts through the language operators.

Refer to the PCL Technical Reference Manual and the PCL Implementor's Guide and Draft Appendix for background information to the information in this appendix.

The PCL XL Font Format 0 Header

The PCL XL font header is a modified form of PCL Format 16. This allows support of TrueType and bitmap characters.

The font header for PCL XL Format 0 is shown below:

Byte	MSB	LSB	Byte
0	Format = 0	Orientation	1
2	Mapping		3
4	Font Scaling Technology	Variety = 0	5
6	Number Of Characters		7

PCL XL does not support Intellifont rendering in the printer. Intellifont characters should be rendered into bitmap characters on the host and downloaded as bitmap glyphs.

Appendix**L. PCL XL Font Formats and Font Definition Operators*****Definitions*****Format (UBYTE)**

PCL XL download format flag. Value = 0.

Orientation (UBYTE)

Specifies the orientation of characters in a bitmap font. PCL XL supports the following values:

Value	Orientation
0	Portrait
1	Landscape
2	Reverse-Portrait
3	Reverse-Landscape

For TrueType fonts, this field should be set to 0.

Font Scaling Technology (UBYTE)

This field specifies the font technology to be used to scale the characters. PCL XL supports the following values:

Value	Font Scaling Technology
1	TrueType
254	Bitmap

Variety (UBYTE)

Meaning depends on the value of Font Scaling Technology. For TrueType fonts this field must be zero.

Mapping (UINT16)

Specifies the native mapping for the font in the PCL-style symbol set format.

Unlike PCL, where fonts are considered bound or unbound, in PCL XL every font is considered to have a native mapping. A font that PCL considers unbound, PCL XL considers as having a native mapping of Unicode.

Appendix**L. PCL XL Font Formats and Font Definition Operators**

A font can be remapped if there is such a mapping in the printer. For example, a font whose native mapping is Unicode could be remapped to Roman-8 if a Roman-8 to Unicode mapping is available. (This is what PCL calls ‘binding’ an unbound font to Roman-8.)

Number of Characters (UINT16)

Specifies the number of characters that will be downloaded for this font. Used by Font Manager to allocate cache memory for downloaded characters.

Note: This value excludes ‘special’ glyphs, such as glyph pieces and vertical substitutes.

Segmented Font Data

The segmented font data immediately follows the PCL XL Font Header. Each segment contains three parts: **Segment Identifier (UINT16)**, **Segment Size (UINT16)** and **Data Segment**.

Byte	15(MSB)	8	7	(LSB)0	Byte
0	1 st segment, segment ID				1
2	1 st segment, segment size				3
4					5
6	data segment		...		7
...
6 + 1 st segment size	2 nd segment, segment ID				
	2 nd segment, segment size				
	data segment		...		
		
	MORE...				
	...SEGMENTS				
	NULL segment ID = 0xFFFF				
	NULL segment size = 0				

Bitmap Supported Segments

PCL XL supports the following font segments for bitmap Fonts:

Bitmap Resolution Segment (required)

Leveraged from PCL font format 16 without modification.

Vendor Information Segment (optional)

New for PCL XL format 0. An all-purpose segment that can be used by ISV's to store any extraneous font information (copyright, etc.).

NULL Segment (required)

Leveraged from PCL font format 16 without modification.

TrueType Supported Segments

PCL XL supports the following font segments for TrueType Fonts:

Global TrueType Segment (required)

Leveraged from PCL font format 16 without modification.

Galley Character Segment (optional)

Leveraged from PCL font format 16 without modification.

Vertical Transformation (Substitution) Segment (optional)

Leveraged from PCL font format 16 without modification.

Vendor Information Segment (optional)

New for PCL XL format 0. An all-purpose segment that can be used by ISV's to store any extraneous font information (copyright, etc.).

NULL Segment (required)

Leveraged from PCL font format 16 without modification.

Appendix**L. PCL XL Font Formats and Font Definition Operators***PCL XL Font Header Operators***BeginFontHeader**

This operator signals the beginning of the font header download. It provides information about the font that is used to prepare PCL XL's font database: format, name. A ReadFontHeader operator must follow this operator.

ReadFontHeader

Allows arbitrary chunks of the PCL XL font header and segmented font data to be downloaded. With this operator, host memory requirements can be optimized (the entire font header need not be buffered on the host). Either another ReadFontHeader operator or an EndFontHeader operator must follow this operator.

EndFontHeader

Signals the end of font header download. Although this seems to serve the same purpose as the NULL segment, it provides easier fault recovery and fits well into the structure of the PCL XL language.

Appendix**L. PCL XL Font Formats and Font Definition Operators*****PCL XL Font Header Operator Sequence***

The following illustration shows how the above operators are to be used:

BeginFontHeader		
ReadFontHeader	PCL XL Font Header	
	1 st Font Segment	
ReadFontHeader	2 nd Font Segment	
	3 rd Font Segment	
ReadFontHeader	4 th Font Segment	
	NULL Segment	
EndFontHeader		

Note that the data sent with each ReadFontHeader operator need not be one complete segment, although this may be the simplest implementation for our driver.

Appendix

L. PCL XL Font Formats and Font Definition Operators

*PCL XL Character Formats**Bitmap Character Format*

PCL XL's bitmap character format is a modified version of PCL5e's Bitmap character format. The descriptor size field has been removed. The continuation field has also been removed, since continuation blocks are not supported. The orientation field has been removed, since that is sent in the font header. The delta-x field has been removed, since text is escape encapsulated. The format field must equal 0. The class field must equal 0 for uncompressed bitmaps. (Another class value may be added later for compressed bitmaps.)

PCL XL's bitmap character format is shown below:

Byte	15(MSB)	8	7 (LSB)	0	Byte
0	Format =0		Class=0		1
2	Left Offset				3
4	Top Offset				5
6	Character Width				7
18	Character Height				9
10	Bitmap Character Data				11
...	...				

Using the Format 1 Class 0 character definition assumes that the TrueType 'hmtx' and 'hhea' tables have already been downloaded as part of the GT font segment of the downloaded font header. These tables store left side bearing and advance width values for all glyphs in the font. For optimized RAM usage, this format should only be used if most or all of the font glyphs are actually downloaded and used.

For better use of RAM when only printing a few of a downloaded TrueType font's characters, remove the 'hmtx' and 'hhea' tables from the GT segment of the font's header and use the Format 1 Class 1 character description. This configuration includes fields for the glyphs left side bearing and advance width values.

PCL XL's TrueType character Format 1 Class 1 is shown below:

Byte	15(MSB)	8	7 (LSB)	0	Byte
0	Format=1		Class=1		1
2	Character Data Size				3
4	Left Side Bearing				5

Appendix	L. PCL XL Font Formats and Font Definition Operators	
6	Advance Width	7
8	Glyph ID	9
10	TrueType Glyph Data	
...	...	

When printing with the eVerticalRotated WritingMode and using a downloaded proportional font with Format 1 Class 0 or 1 character definitions, it is assumed that the TrueType ‘vmtx’ and ‘vhea’ tables have already been downloaded as part of the GT font segment of the downloaded font header. These tables store the topside bearing values for all glyphs in the font. For optimized RAM usage, this format should only be used if most or all of the font glyphs are actually downloaded and used.

For better use of RAM when only printing a few of a downloaded TrueType font’s characters, remove the ‘vmtx and ‘vhea tables from the GT segment of the font’s header and use the Format 1 Class 2 character description. This configuration includes fields for the glyphs left side bearing and advance width values. . (Note: Advance height values are not used by PCL-XL; therefore, they are not part of the Class 2 definition).

PCL XL’s TrueType character Format 1 Class 2 is shown below:

Byte	15(MSB)	8	7 (LSB)0	Byte
0	Format=1		Class=2	1
2	Character Data Size			3
4	Left Side Bearing			5
6	Advance Width			7
8	Top Side Bearing			9
10	Glyph ID			11
12	TrueType Glyph Data			
...	...			

Appendix**L. PCL XL Font Formats and Font Definition Operators*****TrueType Character Format***

PCL XL's TrueType character format is a modified version of PCL's TrueType character format. The descriptor size field has been removed, since variable length descriptors are not supported. The continuation field has also been removed, since continuation blocks are not supported. The format field must equal 1. The class field must equal 0, 1, or 2.

PCL XL's TrueType character format is shown below:

Byte	15(MSB)	8	7 (LSB)0	Byte
0	Format=1		Class=0	1
2	Character Data Size			3
4	Glyph ID			5
6	TrueType Glyph Data			7
...	...			

PCL XL Character Definition Operators***BeginChar***

Specifies, by name, the download font with which to associate the following download characters. A ReadChar operator must follow this operator.

ReadChar

This operator is used to download a single character. One of its parameters specifies the character code. Embedded data containing the definition of a single character in one of the above-described formats **must** be sent with this operator. Using a character code of 0xFFFF will specify 'Special' glyphs. Either another ReadChar operator or an EndChar operator must follow this operator.

EndChar

Signals the end of character downloading.

Appendix**L. PCL XL Font Formats and Font Definition Operators*****Character Definition Operator Sequence***

The following illustration shows how the above operators are to be used:

BeginChar		
ReadChar	Character Data	
ReadChar	Character Data	
...	...	
ReadChar	Character Data	
EndChar		

Note that each ReadChar operator *must* be followed by the data for one complete character.

Appendix M. Table of Raster Operations
M. Table of ROP3 Operations

The following table shows the mapping between input values and their logical operations. Note that the logical operations are specified as RPN (reverse polish, “postfix” notation) equations. Note that the operations assume RGB color space where white=1 and black=0. Here is a key to understand what the ROP values mean:

S	=	Source	a	=	AND
P	=	Paint	o	=	OR
D	=	Destination	n	=	NOT
			x	=	EXCLUSIVE OR

ROP #	Function				
0	0	46	PSDPxox	95	DPan
1	DPSoon	47	PSDnoan	96	PDSxa
2	DPSona	48	PSna	97	DSPDSaoxxn
3	PSon	49	SDPnaon	98	DSPDoax
4	SDPona	50	SDPSoox	99	SDPnox
5	DPon	51	Sn	100	SDPSoax
6	PDSxon	52	SPDSaox	101	DSPnox
7	PDSaon	53	SPDSxnox	102	DSx
8	SDPnaa	54	SDPox	103	SDPSonox
9	PDSxon	55	SDPoan	104	DSPDSonooxxn
10	DPna	56	PSDPOax	105	PDSxxn
11	PSDnaon	57	SPDnox	106	DPsax
12	SPna	58	SPDSxox	107	PSDPSoaxxn
13	PDSnaon	59	SPDnoan	108	SDPax
14	PDSonon	60	PSx	109	PDSPDoaxxn
15	Pn	61	SPDSonox	110	SDPSnoax
16	PDSona	62	SPDSnaox	111	PDSxnan
17	DSon	63	PSan	112	PDSana
18	SDPxnon	64	PSDnaa	113	SSDXPDxaxn
19	SDPaon	65	DPSxon	114	SDPSxox
20	DPsxon	66	SDxPDxa	115	SDPnoan
21	DPSaon	67	SPDSanaxn	116	DSPDxox
22	PSDPSanaxx	68	SDna	117	DSPnoan
23	SSPxDSxaxn	69	DPSnaon	118	SDPSnaox
24	SPxPDxa	70	DSPDaox	119	DSan
25	SDPSanaxn	71	PSDPxaxn	120	PDSax
26	PDSPaox	72	SDPxa	121	DSPDSoaxxn
27	SDPSxaxn	73	PDSPDaooxxn	122	DPSPnoax
28	PSDPAox	74	DPSPdoax	123	SDPxnan
29	DSPDxaxn	75	PDSnox	124	SPDSnoax
30	PDSox	76	SDPana	125	DPsxnan
31	PDSoan	77	SSPxDSxoxn	126	SPxDSxo
32	DPSnaa	78	PDSPxox	127	DPSaan
33	SDPxon	79	PDSnoan	128	DPSaa
34	DSna	80	PDna	129	SPxDSxon
35	SPDnaon	81	DSPnaon	130	DPsxnna
36	SPxDSxa	82	DPSPdaox	131	SPDSnoaxn
37	PDSPanaxn	83	SPDSxaxn	132	SDPxna
38	SDPSaox	84	DPSONon	133	PDSPnoaxn
39	SDPSxnox	85	Dn	134	DSPDSoaxx
40	DPSxa	86	DPSox	135	PDSaxn
41	PSDPSaoxxn	87	DPSoan	136	DSa
42	DPsana	88	PDSPoax	137	SDPSnaoxn
43	SSPxPDxaxn	89	DPSnox	138	DSPnoa
44	SPDSoax	90	DPx	139	DSPDxoxn
45	PSDnox	91	DPSPDonox	140	SDPnoa
		92	DPSPDxox	141	SDPSxoxn
		93	DPSnoan		
		94	DPSPDnaox		

Appendix

M. Table of Raster Operations

142	SSDxPDxax	202	DPSDxax
143	PDSanan	203	SPDSaoxn
144	PDSxna	204	S
145	SDPSnoaxn	205	SDPono
146	DPSPoaxx	206	SDPnao
147	SPDaxn	207	SPno
148	PSDPSoaxx	208	PSDnoa
149	DPSaxn	209	PSDPxoxn
150	DPSxx	210	PDSnax
151	PSDPSonoxx	211	SPDSaoxn
152	SDPSonoxn	212	SSPxPDxax
153	DSxn	213	DPSanan
154	DPSnax	214	PSDPSaoxx
155	SDPSoaxn	215	DPSxan
156	SPDnax	216	PDSPxax
157	DSPDoaxn	217	SDPSaoxn
158	DSPDSaoxx	218	DPSDanax
159	PDSxan	219	SPxDSxan
160	DPa	220	SPDnao
161	PDSPnaoxn	221	SDno
162	DPSnoa	222	SDPxo
163	DPSDxoxn	223	SDPano
164	PDSPonoxn	224	PDSoa
165	PDxn	225	PDSoxn
166	DSPnax	226	DSPDxax
167	PDSPoaxn	227	PSDPaoxn
168	DPSoa	228	SDPSxax
169	DPSoxn	229	PDSPaoxn
170	D	230	SDPSanax
171	DPSono	231	SPxPDxan
172	SPDSxax	232	SSPxDSxax
173	DPSDaoxn	233	DSPDSanaxxn
174	DSPnao	234	DPSao
175	DPno	235	DPSxno
176	PDSnoa	236	SDPao
177	PDSPxoxn	237	SDPxno
178	SSPxDSxox	238	DSo
179	SDPanax	239	SDPnoo
180	PSDnax	240	P
181	DPSDoaxn	241	PDSono
182	DPSPaoxx	242	PDSnao
183	SDPxan	243	PSno
184	PSDPxax	244	PSDnao
185	DSPDaoxn	245	PDno
186	DPSnao	246	PDSxo
187	DSno	247	PDSano
188	SPDSanax	248	PDSao
189	SDxPDxan	249	PDSxno
190	DPSxo	250	DPo
191	DPSano	251	DPSnoo
192	PSa	252	PSo
193	SPDSnaoxn	253	PSDnoo
194	SPDSonoxn	254	DPSoo
195	PSxn	255	1
196	SPDnoa		
197	SPDSxoxn		
198	SDPnax		
199	PSDPoaxn		
200	SDPoa		
201	SPDoxn		

Appendix**N. Modified Backus-Naur Form****N. Modified Backus-Naur Form**

The legal operation sequence of the PCL XL Imaging Protocol is described using a modified Backus-Naur form (BNF). The form is used in this document to specify the legal execution sequence of imaging protocol operators and the format of attribute lists.

BNF allows the user to provide a keyword that defines a class of syntax tokens. This token-class can then be further defined by a set of rules specified by BNF. An example showing how to use BNF to further define the keyword *digit* is shown below:

$$\text{digit} ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$$

The rule for replacing *digit* above may be read as follows “a keyword digit may be replaced by the symbol 0, the symbol 1,..., or the symbol 9.” The vertical bars in the example represent the “or’s” in the previous description.

The various symbols that can be used in BNF rule are listed in the table below:

symbol	meaning
::=	may be replaced by
	choose one among the vertical bars for replacement
&	choose all among the ampersands for replacement in any order
{ }	choose the enclosed items for replacement one time
{ } _n	choose the enclosed items for replacement n times
{ } ₀₊	choose the enclosed items for replacement zero or more times
{ } ₁₊	choose the enclosed items for replacement one or more times
{ } _{opt}	this item is optional for replacement (not required)

There are also special meanings for the tokens contained within the rules as shown below:

token format	meaning
first character is a lower-case letter	the token may be replaced by another rule (is not a terminal token)
first character is an upper-case letter	the token is a terminal value where further meaning is specified by the system or other documentation
first two characters are “e_”	the token is an enumerated type that is one value selected from a sequence of integer values greater than or equal to zero
“any text within double quotes”	a terminal value expressed in natural language
digits (1,2,3...)	any digit (whole or fractional) are terminal tokens

Appendix

O. Symbol Set Selection Table

O. Symbol Set Selection

The PCL XL **SetFont** operator requires the user to select the symbol set to be used for the font. This section provides a listing of some of the standard symbol sets available. **Users should reference *The Book of Characters* (available from HP: P/N 5091-5154E) for the most up-to-date symbol set listings and descriptions. For the symbol sets available in a specific product, see that product's *Software Application Notes*.**

PCL XL uses an integer number to select a symbol set. This number maps to the code used by legacy PCL. The code used by legacy PCL is a number followed by a capital letter A through Z (i.e. "19U"). The following formula may be used to convert the legacy PCL code to the PCL XL symbol set number:

PCL XL symbol set number =

(PCL symbol set number * 32) + (ordinal of letter... where 'A'=1, 'B'=2, etc.)

Example for converting PCL symbol set code "19U" to PCL XL symbol set number:

$$(19 * 32) + (21) = 629$$

Symbol Set Selection Table

Symbol Set Name	PCL Symbol Set Code	PCL XL Symbol Set Number
3 of 9 Barcode	0Y	25
Alternate	11P	368
Alternate Caps	23J	746
Astrology 1	24L	780
Bit Paired APL	1P	48
Carta	20L	652
Chess	23L	748
CODABAR Barcode	5Y	185
Code 11 Barcode	7Y	249
Code Page 864 Latin/Arabic	10V	342
Cyrillic	1R	50
Cyrillic ASCII (8859/5-1986)	0R	18
DeskTop	7J	234
Devanagari	2D	68
Document	8J	266
Expert	10P	336
Fractions	19J	618

Appendix

O. Symbol Set Selection Table

Fraktur	12P	400
Greek-8	8G	263
Hebrew-7	0H	8
Hebrew-8	8H	264
HP Block Characters	1L	44
HP European Spanish	7S	243
HP German	0G	7
HP Large Characters (264x)	0C	3
HP Latin Spanish	8S	275
HP Spanish	1S	51
HP-GL Download	16S	531
HP-GL Drafting	17S	563
HP-GL Special Symbols	18S	595
HPL	5U	181
Industrial 2 of 5 Barcode	1Y	57
Interleaved 2 of 5 Barcode	4Y	153
ISO 2: Int'l Reference Version	2U	85
ISO 4: United Kingdom	1E	37
ISO 6: ASCII	0U	21
ISO 10: Swedish	3S	115
ISO 11: Swedish	0S	19
ISO 13: Katakana	1K	43
ISO 14: JIS ASCII	0K	11
ISO 15: Italian	0I	9
ISO 16: Portuguese	4S	147
ISO 17: Spanish	2S	83
ISO 21: German	1G	39
ISO 25: French (obsolete)	0F	6
ISO 57: Chinese	2K	75
ISO 60: Danish/Norwegian	0D	4
ISO 61: Norwegian Version 2	1D	36
ISO 69: French	1F	38
ISO 84: Portuguese	5S	179
ISO 85: Spanish	6S	211
ISO 8859/1 Latin 1	0N	14
ISO 8859/2 Latin 2	2N	78
ISO 8859/3 Latin 3	3N	110
ISO 8859/4 Latin 4	4N	142
ISO 8859/5 Latin/Cyrillic	10N	334

Appendix**O. Symbol Set Selection Table**

ISO 8859/6 Latin/Arabic	11N	366
ISO 8859/7 Latin/Greek	12N	398
ISO 8859/8 Latin/Hebrew	7H	232
ISO 8859/9 Latin 5	5N	174
ITC Zapf Dingbats Series 100	11L	364
ITC Zapf Dingbats Series 200	12L	396
ITC Zapf Dingbats Series 300	13L	428
JIS Kanji-1	50K	1611
JIS Kanji-2	51K	1643
Kana-8	8K	267
Korean-8	9K	299
Legal	1U	53
Line Draw-7	0L	12
Line Draw-7 (same as 0L)	0B	2
Line Draw-8	8L	268
Lining Figures	21J	682
Math-7	0M	13
Math-7 (same as 0M)	0A	1
Math-8	8M	269
Matrix 2 of 5 Barcode	2Y	89
MC Text	12J	394
MICR	10O	335
Microsoft Publishing	6J	202
MSI/Plessey Barcode	6Y	217
OCR-A	0O	15
OCR-B	1O	47
OCR-M	2O	79
OEM-1	7U	245
Old Style Figures	18J	586
Ornaments	21L	684
PC Cyrillic	3R	114
PC-1004	9J	298
PC-8 Code Page 437	10U	341
PC-8 D/N - Danish/Norwegian	11U	373
PC-8 Latin/Greek	12G	391
PC-8 T Turkish	9T	308
PC-850 - Multilingual	12U	405
PC-851 Latin/Greek	10G	327
PC-852 Latin 2	17U	565

Appendix**O. Symbol Set Selection Table**

PC-857 Turkish	16U	533
PC-860 Portugal	20U	661
PC-861 Iceland	21U	693
PC-862 Latin/Hebrew	15H	488
PC-863 Canada French	23U	757
PC-865 Norway	25U	821
Pi Font	15U	501
Pi Set #1	31L	1004
Pi Set #2	32L	1036
Pi Set #3	33L	1068
Pi Set #4	34L	1100
Pi Set #5	35L	1132
Pi Set #6	36L	1164
PS ISO Latin 1	11J	362
PS ITC Zapf Dingbats	10L	332
PS Math	5M	173
PS Text	10J	330
reserved (Specials)	***Q	n/a
Roman Extension	0E	5
Roman-8	8U	277
Small Caps and Lining Figures	22J	714
Small Caps and Old Style Figures	17J	554
Sonata	20S	659
Swash Characters	16J	522
Symbol	19M	621
Tax Line Draw	2L	76
Tech-7	1M	45
Teletex	10T	340
TEX Math Extension	11M	365
TEX Math Italic	13M	429
TEX Math Symbol	12M	397
Thai-8	0T	20
Turkish-8	8T	276
Typewriter Paired APL	0P	16
Universal Greek & Math Pi	10M	333
Universal News & Commercial Pi	22L	716
UPC/EAN Barcode	8Y	281
USPS ZIP	15Y	505
Ventura International	13J	426

Appendix**O. Symbol Set Selection Table**

Ventura ITC Zapf Dingbats	9L	300
Ventura Math	6M	205
Ventura US	14J	458
Windows 3.0 Latin 1	9U	309
Windows 3.1 Latin 1	19U	629
Windows 3.1 Latin 2	9E	293
Windows 3.1 Latin 5	5T	180
Windows 3.1 Latin/Cyrillic	9R	306
Windows 3.1 Latin/Greek	9G	295
Wingdings	579L	18540

Appendix**P. HP LaserJet Printer Internal Font Selection****P. HP LaserJet Printer Internal Font Selection**

The PCL XL **SetFont** operator requires the user to name the font to be used in all following Text operators for the current graphics state level. The font name for downloaded fonts is user-defined. However, this font name must not be the same as the device's internal fonts. **For the internal fonts available in a specific product, see that product's *Software Application Notes*.**

In HP LaserJet Printers, the internal font name is a multi-byte field of fixed length. The multi-byte field contains two sub-fields. The first sub-field contains the base font name and is left-justified in the internal font name field. The second field contains the character enhancements (i.e. bold, italic, condensed, etc.). The second sub-field is optional, occupies the bytes unused by the base font name, and is right-justified in the internal font name field.

The following table provides examples of the internal font name field format used in HP LaserJet Printers:

Typeface	field length	Base Font Name	Enhance
Albertus Extra Bold	16 bytes	Albertus	Xb
Albertus Medium	16 bytes	Albertus	Md
Antique Olive	16 bytes	AntiqOlive	
Antique Olive Bold	16 bytes	AntiqOlive	Bd
<i>Antique Olive Italic</i>	16 bytes	AntiqOlive	It
Arial	16 bytes	Arial	
Arial Bold	16 bytes	Arial	Bd
<i>Arial Bold Italic</i>	16 bytes	Arial	BdIt
<i>Arial Italic</i>	16 bytes	Arial	It
CG Omega	16 bytes	CG Omega	
CG Omega Bold	16 bytes	CG Omega	Bd
<i>CG Omega Bold Italic</i>	16 bytes	CG Omega	BdIt
<i>CG Omega Italic</i>	16 bytes	CG Omega	It
CG Times	16 bytes	CG Times	
CG Times Bold	16 bytes	CG Times	Bd
<i>CG Times Bold Italic</i>	16 bytes	CG Times	BdIt
<i>CG Times Italic</i>	16 bytes	CG Times	It
Clarendon Bold Condensed	16 bytes	Clarendon	CdBd
Coronet	16 bytes	Coronet	
Courier	16 bytes	Courier	
Courier Bold	16 bytes	Courier	Bd
<i>Courier Bold Italic</i>	16 bytes	Courier	BdIt
<i>Courier Italic</i>	16 bytes	Courier	It
Garamond Antiqua	16 bytes	Garamond Antiqua	
Garamond Halbfett	16 bytes	Garamond	Hlb

Appendix**P. HP LaserJet Printer Internal Font Selection**

Garamond Kursiv	16 bytes	Garamond	Krsv
Garamond Kursiv Halbfett	16 bytes	Garamond	KrsvHlb
Letter Gothic	16 bytes	LetterGothic	
Letter Gothic Bold	16 bytes	LetterGothic	Bd
<i>Letter Gothic Italic</i>	16 bytes	LetterGothic	It
Line Printer 0N	16 bytes	Line Printer	0N
Line Printer 10U	16 bytes	Line Printer	10U
Line Printer 11U	16 bytes	Line Printer	11U
Line Printer 12U	16 bytes	Line Printer	12U
Line Printer 1U	16 bytes	Line Printer	1U
Line Printer 2N	16 bytes	Line Printer	2N
Line Printer 5N	16 bytes	Line Printer	5N
Line Printer 8U	16 bytes	Line Printer	8U
Marigold	16 bytes	Marigold	
Symbol (αβχδεφ)	16 bytes	Symbol	
Times New	16 bytes	TimeNewRmn	
Times New Bold	16 bytes	TimeNewRmn	Bd
<i>Times New Bold Italic</i>	16 bytes	TimeNewRmn	BdIt
<i>Times New Italic</i>	16 bytes	TimeNewRmn	It
Univers Bold	16 bytes	Univers	Bd
<i>Univers Bold Condensed Italic</i>	16 bytes	Univers	CdBdIt
<i>Univers Bold Italic</i>	16 bytes	Univers	BdIt
Univers Medium	16 bytes	Univers	Md
Univers Medium Condensed	16 bytes	Univers	CdMd
<i>Univers Medium Condensed Italic</i>	16 bytes	Univers	CdMdIt
<i>Univers Medium Italic</i>	16 bytes	Univers	MdIt
Wingdings (☺☻☹☼☽☾☿)	16 bytes	Wingdings	

Q. Compression Methods

The PCL XL **BeginImage** and **BeginRastPattern** operators allow the user to specify compression methods for the image and pattern data. A form of run-length encoding compression is supported for protocol class 1.1 of PCL XL. JPEG compression has been added to protocol class 2.0 of PCL XL.

PCL XL Run Length Encoding Compression Method (eRLECompression)

The PCL XL RLE compression method employs control bytes followed by data bytes. Each control byte in the compressed data sequence is a signed, two's complement byte.

If bit 7 of the control byte is zero ($0 \leq \text{control byte} \leq 127$) the bytes following are *literal*. Literal bytes are simply uncompressed data bytes. The number of literal bytes following a control byte is one plus the value of the control byte. Thus, a control byte of 0 means 1 literal byte follows; a control byte of 6 means 7 literal bytes follow; and so on.

If bit 7 of the control byte is 1 ($-127 \leq \text{control byte} \leq -1$), the byte following the control byte will occur two or more times as decompressed data. A byte following a control byte in this range is called a *repeat* byte. The control byte's absolute value plus one is the number of times the byte following will occur in the decompressed sequence of bytes. For example, a control byte of -5 means the subsequent byte will occur 6 times as decompressed data.

A control byte of -128 is ignored and is not included in the decompressed data. The byte following a control byte of 128 is treated as the next control byte.

It is more efficient to code two consecutive identical bytes as a repeated byte, unless these bytes are preceded and followed by literal bytes. Three-byte repeats should always be encoded using a repeat control byte.

The following is an example of PCL XL RLE compression encoding where tokens within single quotes are single byte character values and the numeric tokens are single byte numbers.

Compressed Bytes	-5 'I' 0 'S' -1 'E'
Uncompressed Bytes	'I' 'I' 'I' 'I' 'I' 'I' 'S' 'E' 'E'

PCL XL JPEG Compression Method (eJPEGCompression).

JPEG is an acronym for “**Joint Photographic Experts Group.**” This group has been working under the auspices of three major international standards organizations – the **ISO, CCITT and IEC** – for the purpose of developing a standard for color image compression. This JPEG software is based in part on the work of the Independent JPEG Group.

Appendix**Q. Compression Methods**

The JPEG implementation in PCL XL protocol class 2.0 decodes gray-scale or color image data in JPEG baseline encoded format. All information required for decoding the image is contained within the JPEG signaling parameters, which accompany the encoded data in the compressed data stream.

Index**PCL XL Feature Reference Index****Index**

- ArcPath operator, 145
- attribute ID, 33
- attribute ID number to attribute name table, 210
- attribute ID tag, 33
- Attribute Lists
 - format specification, 27
- attribute name to attribute ID number table, 212
- attribute value enumerations table, 216
- Backus-Naur
 - modified form, 242
- BeginChar operator, 65
- BeginFontHeader operator, 61
- BeginImage operator, 171
- BeginPage operator, 45
- BeginRastPattern operator, 178
- BeginScan operator, 185
- BeginSession operator, 41
- BeginStream operator, 196
- BezierPath operator, 147
- BezierRelPath operator, 149
- binary operator and data syntax, 36
- binary stream operator example, 37
- bolding characters, 93
- Brush
 - defining the paint source for the brush, 83
- Chord operator, 152
- ChordPath operator, 153
- Clip Path
 - intersecting the clip path, 109, 110
 - setting a clipping rectangle, 37, 111
 - setting the clip mode for clip path construction, 114
 - setting the clip path to the imagable area on the page, 112
- CloseDataSource operator, 59
- CloseSubPath operator, 140
- closing a subpath, 140
- Color Space
 - setting the color space, 80
- Comment operator, 55
- Comments
 - adding a comment, 55
- Compression
 - format, 250
- copies, 219
- CTM
 - reverting to the default CTM for a page, 134
- Cursor
 - setting the absolute cursor location, 77
 - setting the relative cursor location, 78
- Data Sources
 - closing a data source, 59
 - opening a data source, 57
- data type tag, 33
- data type to tag value table, 209
- Data Type to Tag Value Table
 - tag values for data types, 209
- Destination Operand, 120
- Duplex, 219, 222
- Duplex binding mode, 219
- Ellipse operator, 155
- EllipsePath operator, 156
- EndFontHeader operator, 64
- EndImage operator, 175
- EndPage operator, 52
- EndRastPattern operator, 183
- EndScan operator, 187
- EndSession operator, 43
- EndStream operator, 198
- Engine Features, 220
- error and warning codes, 225
- Error Reporting, 223
- ExecStream operator, 199
- fill mode, setting the, 100
- font format
 - bitmap, 230
 - TrueType, 230
- Fonts
 - adding characters to a path, 191
 - beginning a character definition for a font, 65
 - beginning a font header definition, 61
 - ending a font header definition, 64
 - internal font selection, 248
 - painting characters to a page, 189
 - reading character definitions, 66, 68
 - reading font headers, 62
 - selecting the current font, 97
 - setting the angle for character rotation, 90
 - setting the bold value for characters, 93
 - setting the scaling factor for characters, 91
 - setting the shearing factor for characters, 92
 - setting the substitution mode for characters, 95

Index	PCL XL Feature Reference Index
-------	--------------------------------

<ul style="list-style-type: none"> setting the WritingMode for characters, 96 Graphics State <ul style="list-style-type: none"> popping (restoring) the graphics state, 74 pushing (saving) the graphics state, 75 Halftoning <ul style="list-style-type: none"> setting the dither matrix, 131 Images <ul style="list-style-type: none"> beginning an image definition, 171 ending an image definition, 175 reading an image definition, 173 setting the dither matrix, 131 line cap, setting the, 103 line dash style, setting the, 105 line join, setting the, 104 line width, setting the, 107 LinePath operator, 158 LineRelPath operator, 160 Manual Feed, 219 miter limit, setting the, 102 NewPath operator, 141 oblique characters, 92 OpenDataSource operator, 57 operator name to tag value table, 208 operator tag, 34 Operators <ul style="list-style-type: none"> format of specifications, 28, 29, 30 legal sequences for operators, 26 origin, defining the page origin, 135 Pages <ul style="list-style-type: none"> beginning the definition of, 45 defining the origin on a page, 135 ending the definition of, 52 setting the page rotation, 136 setting the scale of the page coordinate system, 137 Paint Operand, 119 PaintPath operator, 142 Palette <ul style="list-style-type: none"> setting the palette in the color space, 80 PaletteData, 214 Paths <ul style="list-style-type: none"> adding a bezier to a path, 147 adding a chord to a path, 153 adding a pie to a path, 164 adding a rectangle to a path, 167 adding a relative bezier to a path, 149 adding a rounded rectangle to a path, 169 adding an arc to a path, 145 adding an ellipse to a path, 156 	<ul style="list-style-type: none"> adding characters to a path, 191 adding lines to a path, 158 adding relative lines to a path, 160 closing a subpath, 140 constructing a new path, 141 painting a path, 142 replacing the current path with the clip path, 113 setting the dash style, 105 setting the fill mode for paths, 100 setting the line cap shape, 103 setting the line join shape, 104 setting the miter limit, 102 setting the pen width, 107 Patterns <ul style="list-style-type: none"> beginning a raster pattern definition, 178 downloading a raster pattern definition, 181 ending a raster pattern definition, 183 Pen <ul style="list-style-type: none"> setting the paint source for the pen, 86 setting the width, 107 Pie operator, 163 PiePath operator, 164 PJL <ul style="list-style-type: none"> and PCL XL Streams, 218 PopGS operator, 74 PushGS operator, 75 ReadChar operator, 66, 68 ReadFontHeader operator, 62 ReadImage operator, 173 ReadRastPattern operator, 181 ReadStream operator, 197 Rectangle operator, 166 RectanglePath operator, 167 RemoveStream operator, 200 RLE Compression <ul style="list-style-type: none"> format, 250 ROP2 Equivalents in the ROP3 Set, 125 ROPs <ul style="list-style-type: none"> destination operand, 120 introduction, 115, 118 paint operand, 119 setting a ROP3, 128 source image operand, 118 table of ROP3s, 240 rotating characters, 90 rotating the page coordinate system, 136 RoundRectangle operator, 168 RoundRectanglePath operator, 169 scaling characters, 91
--	---

Index	PCL XL Feature Reference Index
-------	--------------------------------

scaling the page coordinate system, 137
 Scan Lines
 beginning a scan line definition, 185
 ending a scan line definition, 187
 reading a scan line, 186
 ScanLineRel operator, 186
 Sessions
 beginning a session, 41
 ending a session, 43
 SetBrushSource operator, 83
 SetCharAngle operator, 90
 SetCharAttributes, 96
 SetCharAttributes operator, 96
 SetCharBoldValue operator, 93
 SetCharScale operator, 91
 SetCharShear operator, 92
 SetCharSubMode operator, 95
 SetClipIntersect operator, 109, 110
 SetClipMode operator, 114
 SetClipRectangle operator, 37, 111
 SetClipToPage operator, 112
 SetColorSpace operator, 80
 SetCursor operator, 77
 SetCursorRel operator, 78
 SetFillMode operator, 100
 SetFont operator, 97
 SetHalftoneMethod operator, 131
 SetLineCap operator, 103
 SetLineDash operator, 105
 SetLineJoin operator, 104
 SetMiterLimit operator, 102
 SetPageDefaultCTM operator, 134
 SetPageOrigin operator, 135
 SetPageRotation operator, 136
 SetPageScale operator, 137
 SetPaintTxMode operator, 126
 SetPathToClip operator, 113
 SetPenSource operator, 86
 SetPenWidth operator, 107
 SetROP operator, 128
 SetSourceTxMode operator, 127
 shearing characters, 92
 Source Image Operand
 definitions, 118
 Symbol Set Selection, 243
 syntax for binary operators and data, 36
 tag values for operators, 208
 tag values for operators and data types, 203
 tags, 203
 Text operator, 189
 TextPath operator, 191
 Transparency Mode
 setting the paint transparency mode, 126
 setting the source transparency mode, 127
 Transparency Modes, 121
 User-Defined Streams
 beginning a stream definition, 196
 ending a stream definition, 198
 executing a stream, 199
 reading a stream definition, 197
 removing a stream definition, 200
 vertical characters, 96
 warning and error codes, 225
 white space, 35