PCL 6

A White Paper

3000 Hanover Street
Palo Alto, California 94304 USA
Telephone: (415) 857-1501

Hewlett-Packard Company Confidential

***** NOTICE *****

HEWLETT-PACKARD COMPANY MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF 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. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE OR TECHNICAL INFORMATION IS ASSUMED BY YOU. 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 © 1995,1996 Hewlett-Packard Company. All rights reserved.

Table of Contents

1.0 Introduction to PCL 6

1.1 The Challenge of Digital Image Communication

Imagine it is the year 2000. You're an application-developer or application driver-writer. Your goal is to implement the most efficient and highest quality solution to communicate images among applications and digital printing or display devices.

You know the images to be transmitted to the device may be full of: (1) characters needing exact positioning and precise rendering; (2) scanned images with a variety of color and contrast characteristics; and (3) special effects such as gradient-color or gradient gray-scale regions of arbitrary shapes.

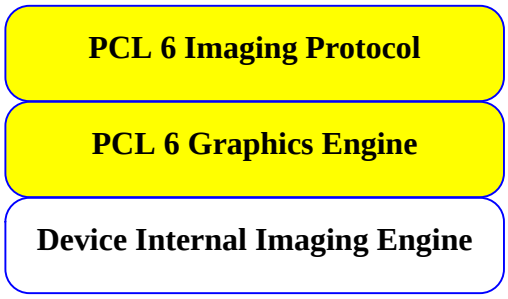
You also know that print quality and performance are key to customer satisfaction for the imaging solution. Customers depending on your solution spend tremendous amounts of money to create documents, pamphlets, and other material for business communication. Loss of image quality in the transmitted image is highly undesirable. You also want to take advantage of the best imaging resolution of the customer's target device.

Some graphics languages such as PostScript and PCL 5e often used for transmitting images were originally developed for the simpler printing needs of nearly two decades ago. To use either of these languages you'll need to emulate and compensate in your application or driver for graphic capabilities or graphics processing power missing in the languages. You hope that your emulations and work-arounds for PostScript and PCL 5e provide the quality you need and eventually match what the user sees in the original soft version of his or her document on the video display. You're concerned about the risk of your undertaking.

1.2 A Solution to the Imaging Challenge

Imagine now that you have a complete solution for imaging your documents. This solution combines the best graphics capabilities of PCL 5e and PostScript with new imaging features for graphical user interfaces in a device-independent and extendible form. This solution you've found for digital imaging is called "PCL 6."

PCL 6 is a system for communicating graphics and text among computers and digital imaging devices. PCL 6 has two components that are built upon a device's primitive imaging engine. These components are depicted in the illustration below.



The two key components of PCL 6 are:

1) *An object-oriented imaging protocol* designed for graphical user interface applications hosted on environments such as Windows 9x, Windows NT, OS/2, Macintosh O/S, etc.

2) And, *a robust graphics engine* for rendering text, vector graphics, and scanned images received through PCL 6 objects.

The PCL 6 imaging protocol is a thin layer of software that translates graphical objects received by the device into a form understood by the PCL 6 graphics engine. The PCL 6 graphics engine is designed for efficient rendering of all graphical objects into a form supported by low-level, more primitive device imaging engines.

This document provides an overview of PCL 6 concepts. All PCL 6 devices conform to the general concepts outlined in this document.

1.3 Design Goals of PCL 6

The dramatic growth of graphics-capable applications has increased the ability of users to create graphics-rich documents for printing and displaying. As the graphics complexity of documents increases, the need for more efficient methods to communicate these documents among applications and devices increases.

PCL 6 is specifically designed to increase the efficiency of communicating electronic images among a wide range of current and future devices, including printers, fax machines, copiers, software on-screen viewers, etc.

The overall design goals of PCL 6 are outlined below.

1.3.1 Make Electronic Imaging Simple

The set of object-oriented imaging tools in PCL 6 almost always provides the application-writer or driver-writer with a *one-to-one match* between the application object to be imaged and a PCL 6 tool to image the object. This reduces the amount of programmer effort required and limits the number of errors made when describing images for a PCL 6 device.

1.3.2 Enable Highest Image Quality and Device Independence

The rich and device-independent set of PCL 6 tools helps reduce the need for ad-hoc application or driver pre-processing and break-down of images due to language imaging deficiencies. Reducing application or driver pre-processing and break-down of images is especially critical for intelligent color or monochrome devices where device-internal processing of the original image produces the best color or grayscale output quality.

1.3.3 Make Electronic Imaging Efficient and Fast

PCL 6 provides select tools such as arbitrary image clipping that may be used to reduce the number of commands required to describe complex graphic images. Applications and drivers using select PCL 6 tools over traditional PCL 5e tools may achieve dramatic increases in imaging speed and significant decreases in memory requirements for complex graphic images.

1.3.4 Provide for Easy and Compatible Future Enhancements

The object-oriented nature of PCL 6 allows commands to be “overloaded” and re-used. Thus, new imaging extensions may be easily implemented for a given command by simply adding and/or changing the structure of data provided to the command. All PCL 6 commands are reusable, extendible, and designed to insure backward compatibility of function.

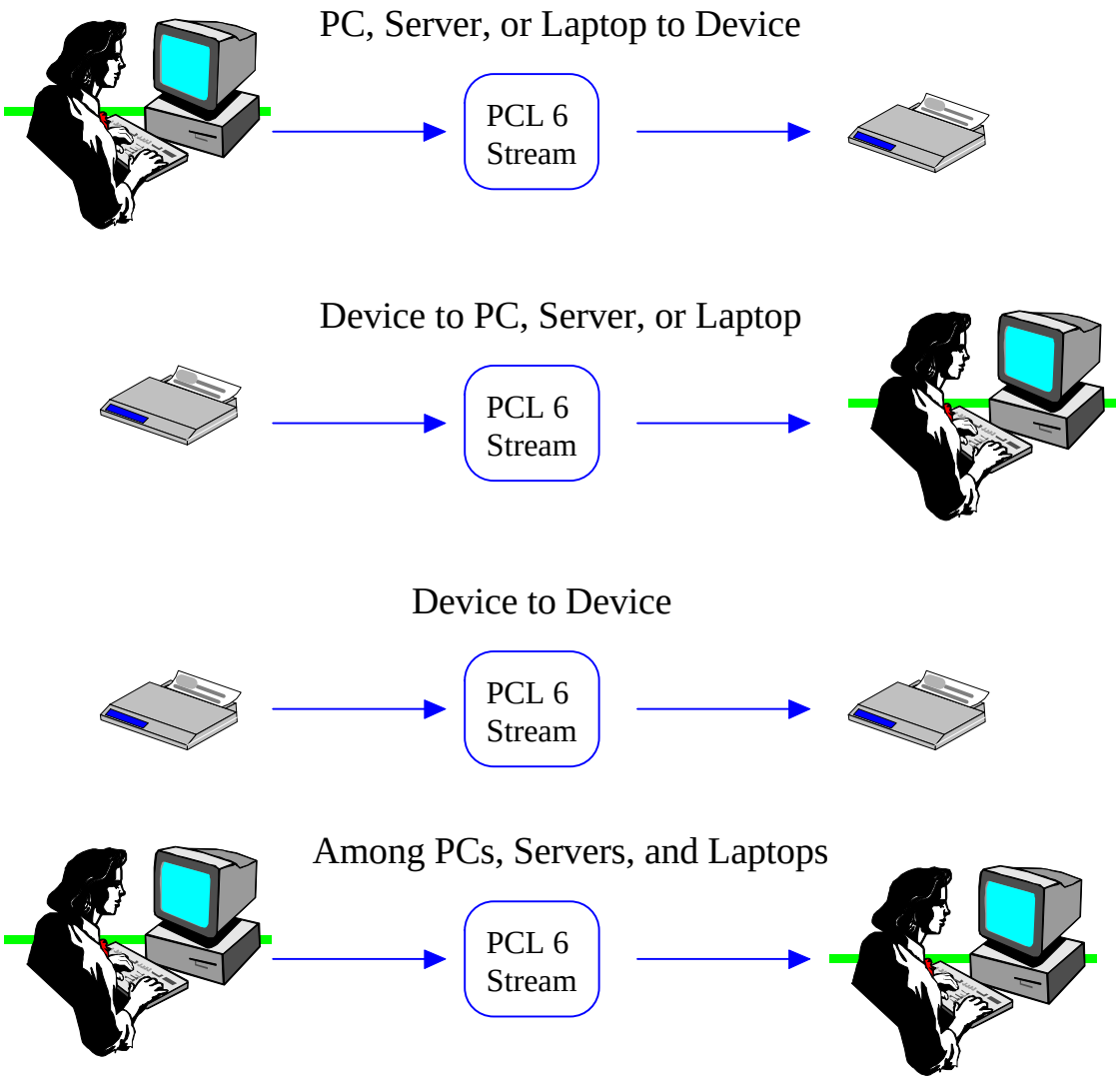
The design goals listed above form the foundation the all current and future design decisions for PCL 6.

2.0 PCL 6 Basic Concepts

2.1 PCL 6 Streams: A Portable Unit of Imaging Work

A PCL 6 stream is a self-describing package of PCL 6 commands and data. PCL 6 streams may be sent to a device to image anything from a single scanned image, a whole document, or an entire set of documents.

A PCL 6 stream is designed to be a universal and portable unit of imaging work. The stream's encapsulated, compact, and device-independent design make it suitable for a wide variety of device imaging communications as illustrated below.



2.2 Stream Object Structure

Most of today's 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. On such devices, PCL 6 imaging is accomplished through an encapsulated sequence of operations called a *stream*. A PCL 6 stream contains a sequence of bytes that may instruct PCL 6 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 6 device to draw one or more of the following:

- A clip-art figure
- A form
- A watermark
- A signature
- An entire page or entire document
- A series of documents

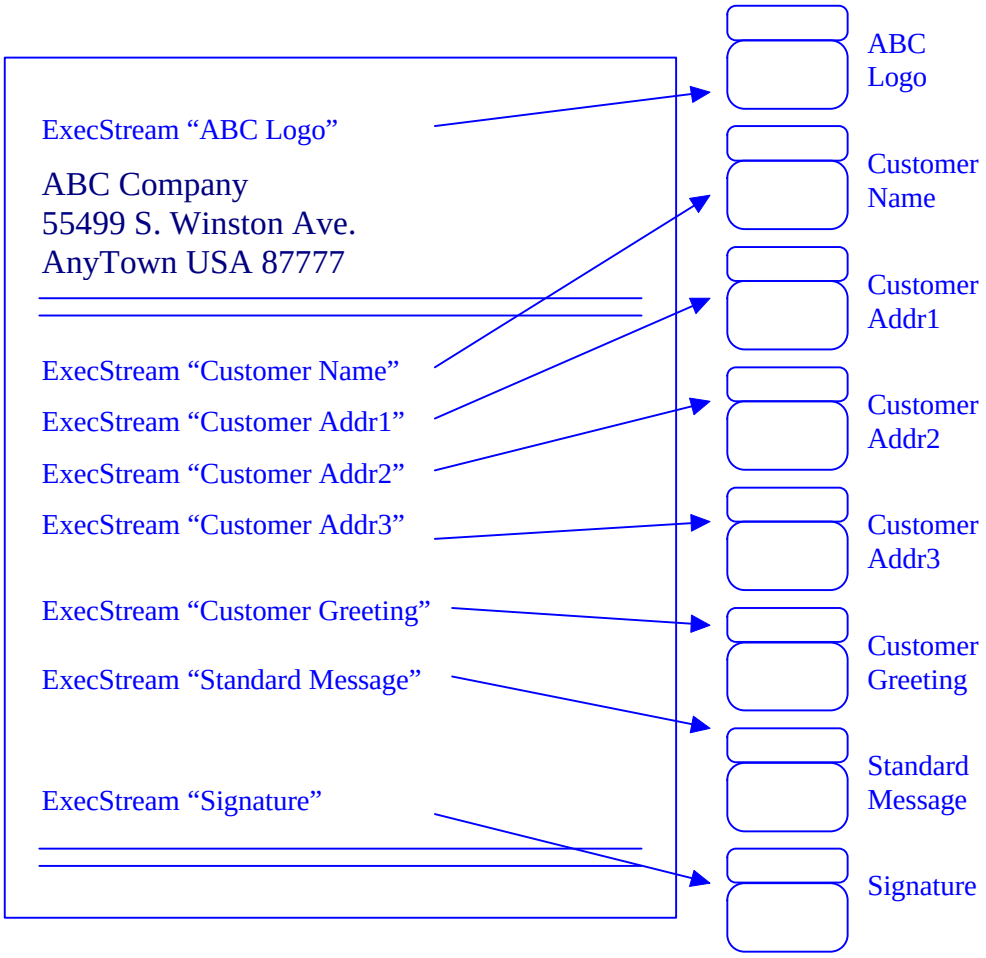
All operators in a *stream body* belong to a specific PCL 6 protocol class. The protocol class specifies the set of operators and the capability of the operators in the stream.

The protocol class and revision number for the class are defined in the *stream header*, preceding the stream body. Prior to executing the stream body, the device reads the information in the stream header to understand if it has the ability to execute the class of operators and data to come.

Some PCL 6 devices may be designed to accept streams with different protocol classes. For example, one could envision a high-volume printer accepting a redirected stream from a less-capable fax machine or from a stream produced by a scanner.

2.3 User-Defined Streams

The "*parent*" or top-level stream in PCL 6 is similar to a traditional print job. Lower-level streams may be defined and referenced by name. These lower-level streams, called "user-defined" streams, may contain any set of PCL 6 imaging operators and data and may be stored and referenced in the device in RAM, flash, or disk.

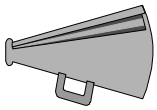


The illustration shows how user-defined streams may be used in the construction of forms. The form is depicted as a large rectangular area on the left of the illustration. At the right of the form there is a stream defined for each field in the form except for the address of the originating organization (ABC Company).

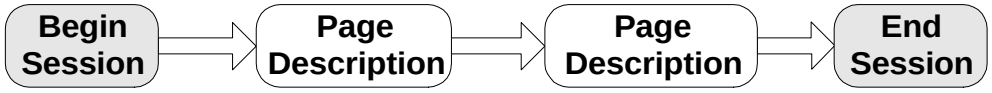
The logo, signature, and standard message fields are captured as user-defined streams since they will be used over and over again during forms imaging. New customer information streams are downloaded prior to imaging each form. The field data in these streams are imaged when the PCL 6 "ExecStream" operator is encountered for each named stream.

Some versions of PCL 6 do not allow streams to be nested in order to avoid recursion.

2.4 Sessions



The next level of structure in PCL 6 streams is the *session*. A session defines the world coordinate system and other default attributes for each one of the pages described during the session. The user instructs PCL 6 to begin a new session in order to perform imaging in the device. The user may only cause imaging to occur on a PCL 6 device during an active session. There may be multiple sessions per parent stream.



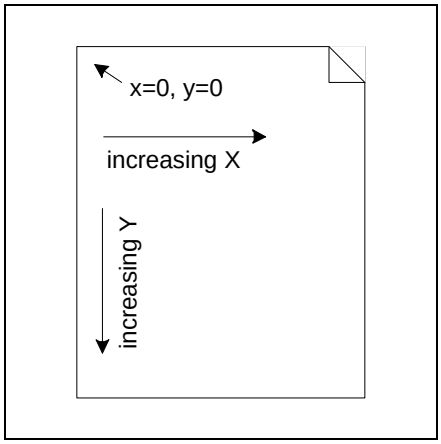
The illustration shows the relationship between sessions and page descriptions. The user may only describe pages within session boundaries.

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 status or error reporting.

The operator to start a session has both *required* and *optional* attributes. PCL 6 devices require a user resolution attribute to be given when a new session is started. This attribute identifies the *world coordinate system* in which the user prefers to describe pages during the session. The page coordinate system may be described in English units (inches) or in Metric units (millimeters). The user units resolution for a session may differ from the internal device resolution. This is because the user coordinate system of a session is device-independent. See the *Coordinate System* section below.

2.5 User Coordinate System

PCL 6 devices have a two-dimensional world coordinate system to specify the location at which graphical objects are placed and painted.



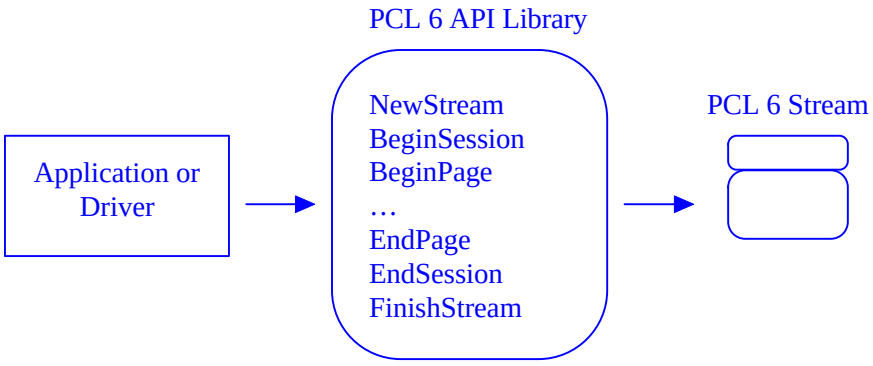
The user coordinate system defaults are depicted in the illustration. These defaults are as follows:

- The origin (x = 0, y = 0) is the “physical” upper left hand corner of 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

Section 3 explains how the user coordinate system may be modified for special needs during page description (see section on Rotate and Transform Concepts).

2.6 Stream Software Developers Kit

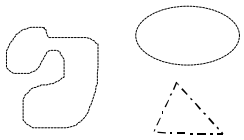
A goal for PCL 6 is to provide a software developer's kit (SDK) compatible with all current versions of PCL 6. The SDK includes a library that allows a developer to create PCL 6 streams via simple application programming interfaces (APIs). This allows the PCL 6 developer to concentrate on functionality and the building blocks of images instead of language syntax and intricate byte sequences. The library also allows improvements to the library's use of the PCL 6 imaging protocol without affecting driver or application code.



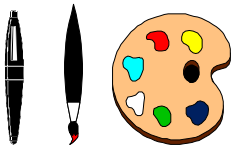
As depicted in the illustration, the PCL 6 API library may be called directly by application or driver software to create PCL 6 streams. In most cases, there is a PCL 6 stream API matching every imaging requirement of the application or graphical user interface.

Section 3 explains the basic PCL 6 toolbox for text and graphics imaging.

3.0 The PCL 6 Toolbox



Path Object

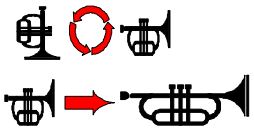


Pen and Brush Objects

Text Operators

Bitmap and Pattern Objects

Clip Path Object

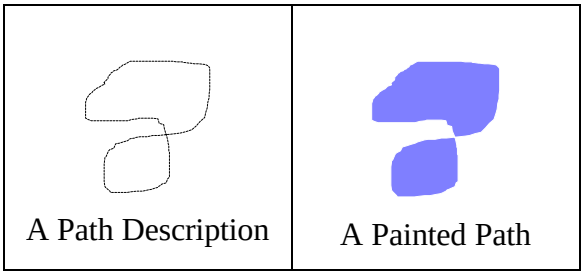


Rotate and Transform Operators

PCL 6 provides the image-creator with a comprehensive graphics toolbox. The illustration above depicts the graphics tools available in PCL 6. Each tool and object in the toolbox is described in the following sections.

3.1 The Path Object

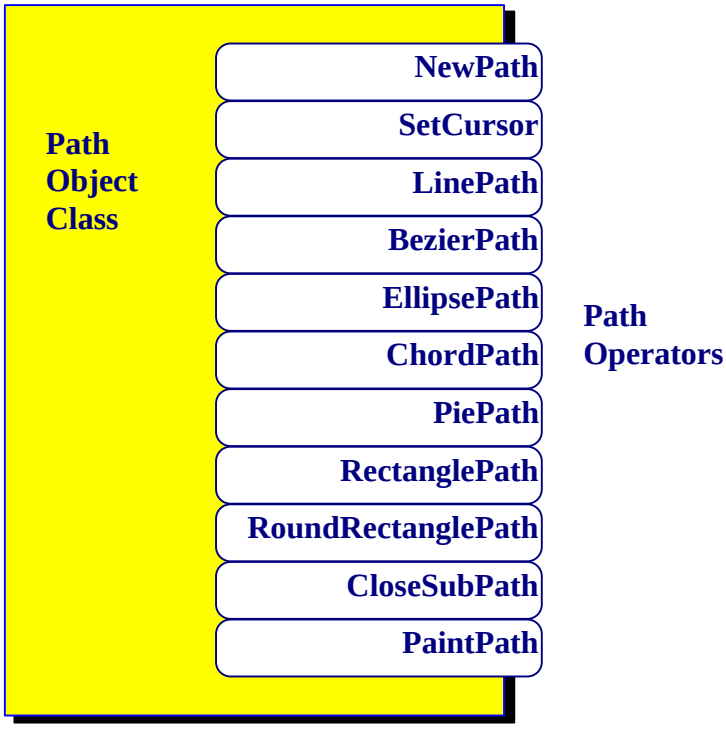
Most images used for business communications includes one or more lines and/or curves. These regions are typically called vector regions. Vector regions are described in PCL 6 using the *Path Object*. The path object is designed for easy, fast, and efficient imaging for any vector region, simple or complex.



The dashed-line symbol on the left of the illustration depicts a path object having a series of connected lines and beziers to describe a closed vector region. The elements of the path object are not visible on a physical page or display until the path object is painted. A connected series of one or more lines and curves form a *sub-path*. A path object contains, zero, one, or many sub-paths.

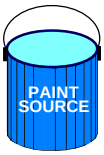
The symbol on the right in the illustration depicts a painted path where a brush was used to fill the path with a pattern. When a path is painted, the device fills inside the path's lines and curves with user-selected paint and strokes paint along the lines and curves. Filling and stroking sub-paths is controlled by pen and brush object settings in the current graphics state (pens and brushes are described later).

3.2 Path Object Operator Listing



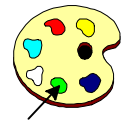
The illustration depicts some of the operators for the path object. The PCL 6 developer is not limited to simple line and bezier descriptions for paths. PCL 6 provides a rich set of path primitives, matching the needs of graphical user interface applications for WYSIWYG, efficiency, and performance.

3.3 Paint Sources



A paint source defines the color or graphical pattern used to paint a path 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.

3.3.1 Color Objects

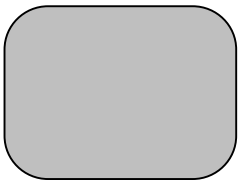

Color Object

Color objects are conceptually a single color in a specific color space (i.e. RGB, Gray, etc.). In a PCL 6 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.

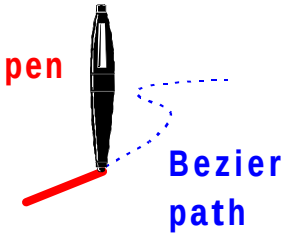
A user-defined mapping into a color space may also be constructed using palettes.

3.3.2 Device-Independent Raster Patterns

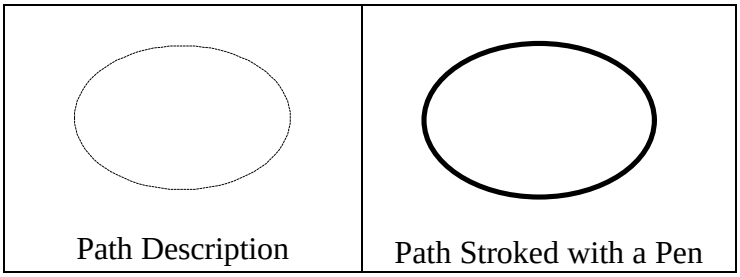


Raster patterns are rectangular NxM source pixel regions used for constructing primitive patterns. The pattern may be specified as a device-independent bitmap. Once created, raster patterns may be associated with a pen or a brush. A raster pattern used to fill the elements of a path object is associated with a brush and tiled across the inside edges of the object's sub-paths. As raster pattern associated with a pen is stroked along the edges of the sub-paths in a path object. Raster patterns are provided for compatibility with graphical user interface environments such as Windows 3.1, Windows 95/NT, and OS/2.

3.4 The Pen Object



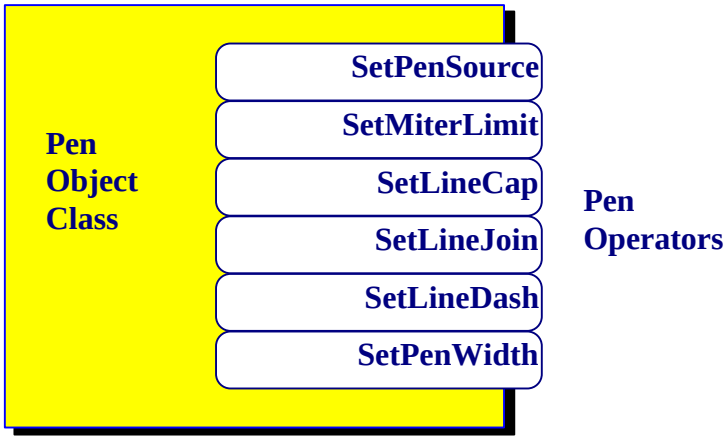
The *Pen Object* is used by the Path Object to stroke paint (primary colors or patterns) along the path's lines and curves.



The illustration depicts the results of painting a path object with the pen set to a moderately wide line width and a black paint source. The following is a summary of the key concepts behind the pen object.

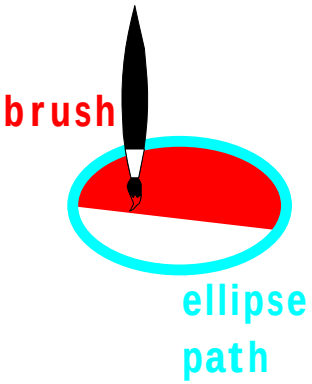
- There is one pen defined to stroke the contents of a path object
- The color or pattern stroked along the edges of the object depends upon the *paint source* currently associated with the pen
- If the pen is not associated with a paint source when an object is painted, no stroking is performed
- The sub-paths in the path object are *not* destroyed when stroked, allowing the path to be reused for later filling or raster operations

3.5 Pen Object Operator Listing



The illustration shows some of the operators for the pen object.

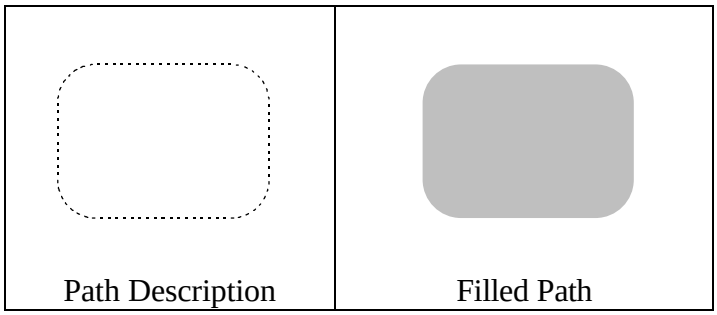
3.6 The Brush Object



The Brush Object is used by the Path Object to fill the inside region of the path's lines and curves with paint. The filling operation is executed when the path object is painted.

The brush object is also used as an operand for raster operations (ROPs) during the process of imaging text, path, and raster objects.

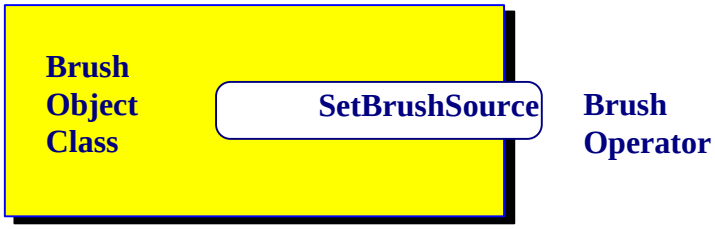
.



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

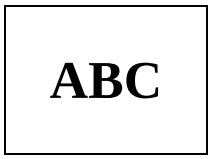
- There is one brush defined to paint path, text, and raster objects
- The color or pattern filled or colored within the object depends upon the *paint source* currently associated with the brush
- If the brush is not associated with a paint source when an object is painted, no filling is performed
-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

3.7 Brush Object Operator Listing



The illustration shows the operator for the brush object.

3.8 Text Operators



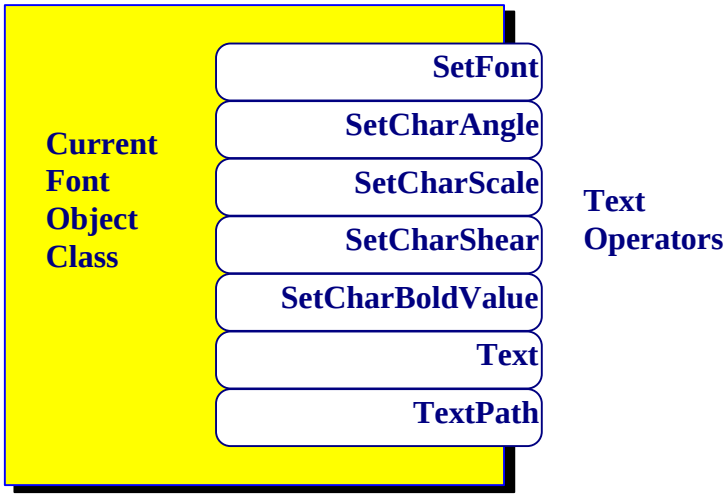
A PCL 6 device allows the user to place characters of a font anywhere on the page and at any angle. The actual font technology available in the device (TrueType, Bitmap, etc.) to render each character is protocol class-dependent.

In PCL 6, 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.

PCL 6 imaging protocol allows many characters to be placed on the page in 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 6 where the cursor should be relocated for placement of each character in succession. The character spacings may be provided for both the current x- and y-axis in a single text operator.

Character sizes are specified in user units. Painted characters are scaled and rotated according to page coordinate transformation matrix (CTM) manipulations by the user (see Rotate and Transform Concepts). Characters may also be rotated, scaled, and sheared in an additive manner to the current page CTM by setting a separate character CTM. 8- and 16-bit character codes are supported.

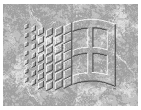
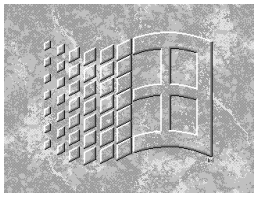
3.9 Current Font Operator Listing



The illustration shows some of the operators for imaging text objects via the current font.

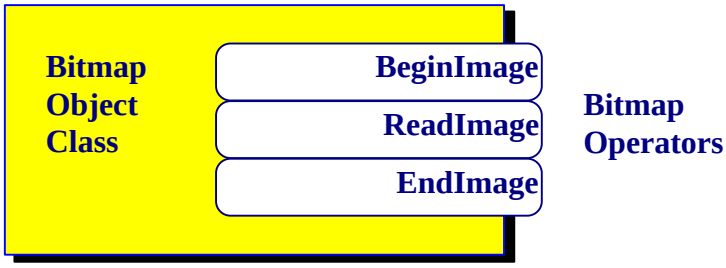
3.10 Device-Independent Bitmap Objects

Bitmaps are rectangular raster regions, including scanned images. Bitmaps may be single- or multi-bit-per-pixel, including color and contone formats. Bitmap formats are device-independent and thus do not have to be transmitted at device-resolution. Bitmaps may be scaled and halftoned by PCL 6.

 Size Submitted by User @ 4 bits-per-pixel 72 pixels-per-inch	 Desired Size On Page @ 600 pixels-per-inch

The illustration depicts an example where a 72 pixels-per-inch image is submitted to a 600 pixels-per-inch PCL 6 device. Assume the user also wants the image to be enlarged to fit the desired area on the physical page. Two things must happen in the device: (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. 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-up bitmaps in a device, instead of the application or driver, off-loads unnecessary work from the user's computer by freeing-up the computer's and I/O channels for useful work other than scaling and half-toning the image. Allowing the device to scale and halftone bitmap images assures device-best print quality for the image.

3.11 Device-Independent Bitmap Operator Listing

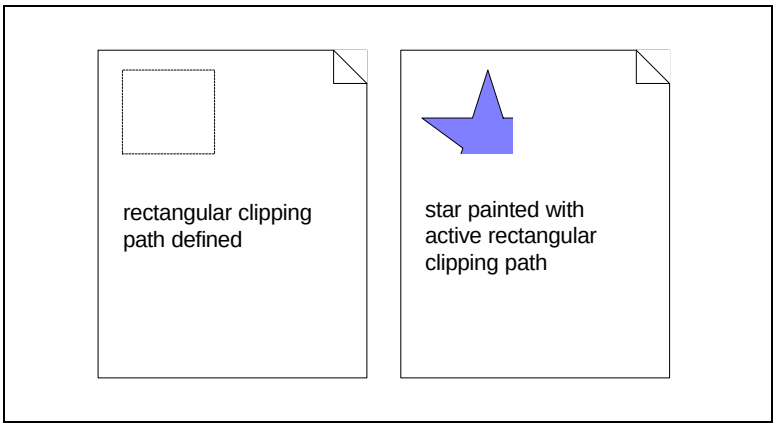


The illustration shows some of the operators for imaging bitmap objects.

3.12 The Clip Path Object

The *Clip Path Object* allows the application or driver to constrain areas in which marks may be imaged for path, text, and raster objects. The clip path object may be set to any arbitrary set of lines and curves defined by the current path object. The default clip path object is the imagable area of a page.

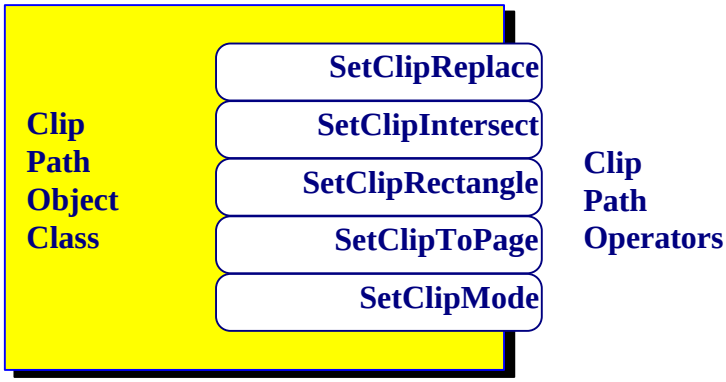
The user may use clip path operators to confine painting operations to only the inside region(s) or only the outside region(s) defined by the current clip path.



The illustration depicts how defining a clip path affects painting operations. In this case, a rectangular clipping region is defined for the clip path object. After the clip path is defined, a star object is painted to the page near the origin. The only elements of the diagonally-striped star painted on the page are those lying within the clip path.

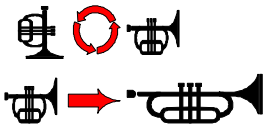
Flexible clip path definition allows the driver or application to efficiently image complex graphics without using compute and memory expensive raster operations (ROPs).

3.13 Clip Path Object Operator Listing

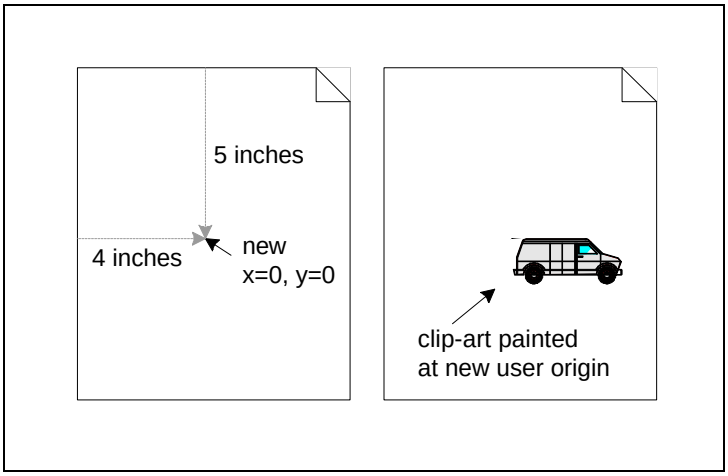


The illustration shows some of the operators for the clip path object.

3.14 Rotate and Transform Concepts



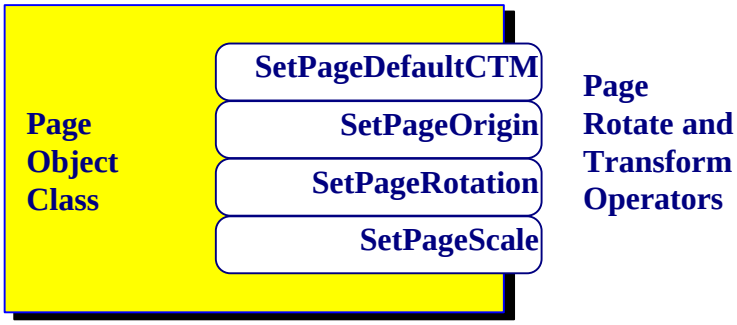
The component used to map user coordinate space to the page surface coordinate space is called a coordinate transformation matrix (CTM). The user may manipulate values in the CTM to rotate and transform the way in which text, vector objects, and raster pixels are painted on the page surface. Initially, the CTM is set such that the origin of a page is the top left corner (whether the page or portrait or landscape).



The illustration depicts 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 the illustration, 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.

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. 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 line edges, text shapes, and raster objects. A driver or application with specific knowledge of a device may be written to set the user session resolution to the target device's internal resolution to achieve the best possible performance.

3.15 Page Rotate and Transform Operator Listing



The illustration shows some of the operators for the page coordinate system settings.

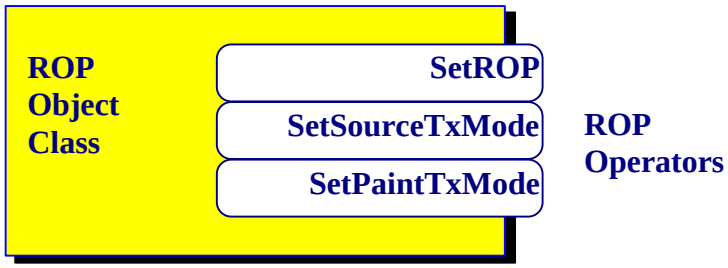
3.16 Raster Operations

10000001
OR 10101110
= 10101111

PCL 6 allows raster operations (ROPs). ROPs 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 ROPs supports at least all standard ROP3 operations.

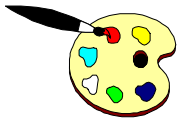
ROPs are particularly useful for applications not supporting paths and clipping to achieve WYSIWYG for complex graphics effects. However, the document description size when using ROP operations for clipping is often much larger than using path clipping and should be avoided whenever possible.

3.17 Raster Operator Listing



The illustration shows the operators for altering the current ROP.

3.18 Graphics State Object



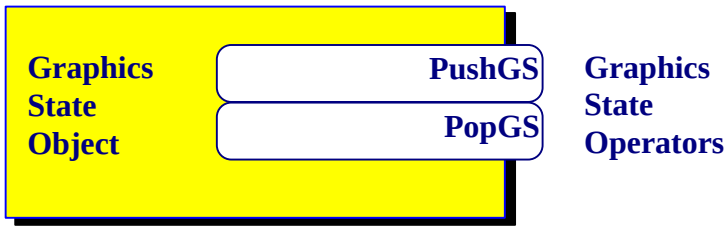
BRUSH PAINT
SOURCE = RED

All PCL 6 graphical objects read and set some attributes in the *Graphics State Object* during definition and painting operations. For example, the path object maintains its current list of components in the graphics state. When the path is painted, the elements to be painted are read from the graphics state object.

Another example graphics state attribute is the current raster operation or ROP. When an object is painted, the ROP stored in the graphics state is read prior to painting. The ROP affects how the paint for the pen and brush are combined with the object being painted and with images already on the page.

The graphics state is modeled as a stack such that the attributes may be pushed and popped in a last-saved, first-restored manner. The current values in the graphic state may be pushed (saved) and popped (restored) at any point in time. The "active" graphics state elements read and written by graphical objects are always the elements at the top of the graphics state stack.

3.19 Graphics State Object Operator Listing



The illustration shows the operators for pushing (saving) or popping (restoring) the current graphics state.

4.0 PCL 6 Operator and Attribute List Design

4.1 Operators

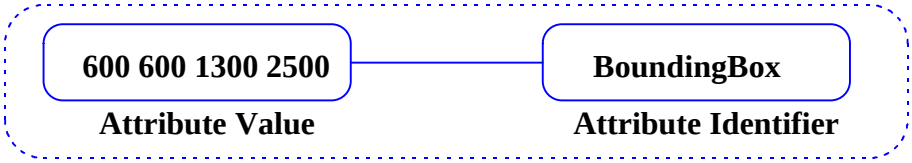
PCL 6 commands are called *operators*. PCL 6 is specifically designed to have a minimum number of operators for ease-of-use, efficiency, and best performance. All operators create, modify, or paint PCL 6 objects. Examples of objects are sessions, pages, the current path, bitmap images, and the current font.

Parameters associated with operators are used to set an object's attributes. These parameters are stored in an operator's attribute list.

Large amounts of data needed by an operator may be obtained for sources external to the attribute list (such as raw data placed in-line in the stream body).

4.2 PCL 6 Attribute Lists

Attribute Lists hold parameter data for PCL 6 operators. Every operator has one attribute list upon execution. An attribute list contains a set of one or more attribute value / identifier pairs. Each pair in an attribute list contains an attribute identifier and a set of one or more values for the attribute.



The attribute list in the illustration has one attribute value/identifier pair. This attribute value/identifier pair is used to define a bounding box with two points (two x, y values). A bounding box attribute pair may be used to define the size of an ellipse or a rectangle.

Each operator knows the valid data type(s) and number of values expected for each attribute identifier. The attribute identifiers in the illustration are represented by a name. The actual attribute identifier is a unique number from a set of attribute identifiers defined for the PCL 6 imaging protocol.

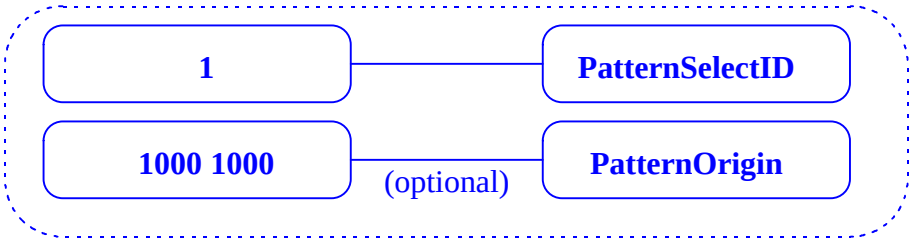
Attribute lists may hold zero, one, or more attribute value/identifier pairs. Many operators need only one attribute value/identifier pair each time they are executed such as the pair in the illustration.

Attribute value/identifier pairs defined for an operator are *not* saved following the execution of that operator. Immediately following the execution of an operator, the attribute list is emptied.

If multiple attribute value/identifier pairs of the same attribute identifier are entered by mistake into the attribute list, the attribute value for the first attribute identifier entered is used by the operator.

4.3 Variable-Length Attribute Lists

Variable-length attribute lists are used when the exact number and type of attributes may vary for an operator.



Variable Length Attribute List

The illustration shows an example of two attribute value/identifier pairs for an operator. One or both of these two attribute value/identifier pairs may be provided to the **SetBrushSource** operator to select a pattern for filling elements in a path object. The **PatternSelectID** attribute value/identifier pair is required for the operator. The **PatternOrigin** attribute value/identifier pair is optional. If the **PatternOrigin** attribute value/identifier pair is not supplied, the origin for tiling the pattern will be the top left corner of the current page.

Variable-type and variable-length attribute lists are allowed through the "operator-overloading" feature of PCL 6. A simple example of this type of operator flexibility is the plus ("+") operator in most programming languages. The same plus operator may be used to add two integer numbers one time and two fractional numbers another time. The operator is always known to the user as plus ("+") even though different data types may be used. This principle of "operator overloading" is often employed in object-oriented systems.

When an operator is to be executed, an 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 parameters in a function call.

When multiple attribute value/identifier pairs are added to an attribute list, the order in which pairs are added is not significant. PCL 6 will extract attribute value/identifier pairs in the order in which they are needed for the operator.

The attribute list mechanism allows PCL 6 to be easily modified 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.

4.4 Simple Operator Examples

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

Step 1: Build the 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
```

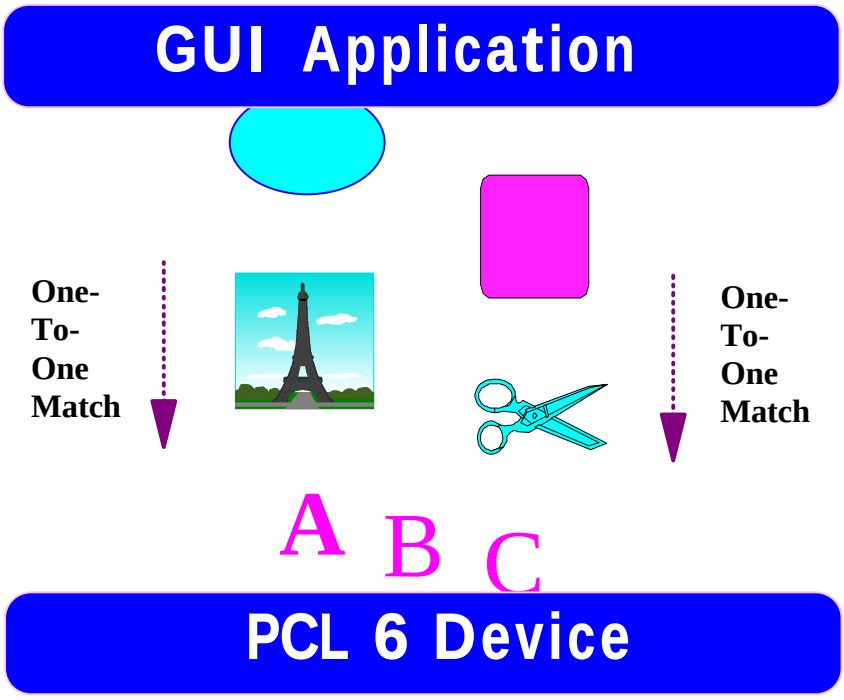
The following PCL 6 sample session paints a single two-inch line on a page:

```
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 at 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 entire imaging session illustrated above may be described in only 40 bytes in the binary stream format accepted by PCL 6 devices.

The attribute values that begin with “e” such as “eInch” are actually names for enumerated integer values. For example, eInch is simply a name for the value 0 for the Measure attribute.

5.0 PCL 6: A Match for Complex Graphics Imaging



PCL 6 imaging tools and objects are designed for the demanding requirements of images produced from today's and future graphical user interface-based applications.

PCL 6 tools provide efficient and compact methods for communicating complex images among devices in a device-independent form.

PCL 6 provides an excellent foundation for future graphics features due to built-in operator extendibility.

PCL 6 is an appropriate vehicle for insuring the highest quality and best performance for digital imaging.