OpenPrinting

Vector Printer Driver Application Program Interface (OPVP) Specification

Version-1.0 RC6

(2009-12-24)

OpenPrinting Vector Printer Driver Application Program Interface Specification

Table of Contents

1

1.Notation and Terminology

1.1.Notational Conventions

This section describes the use of font and style in this document.

Font and Style	Description	Examples

`Courier`	Definition of functions, structures, enumerations and constants.	`opvp_result_t opvpClosePrinter( opvp_dc_t printerContext);`
		
		`typedef struct _opvp_point { opvp_fix_t x, y; } opvp_point_t;`
		
		`#define OPVP_OK 0`

	Function parameters .	`printerContext`
	Source code examples.	`#ifndef _OPVP_H_` `#define _OPVP_H_`
Italic	Coordinate values (x, y)	*(x0, y0)*

1.2.Conformance Terminologies

In this document, capitalized terms, such as: MUST, MUST NOT, REQUIRED, SHOULD, SHOULD NOT, MAY, and OPTIONAL, are intended to be interpreted as described in [RFC2119].

2.Introduction

This document specifies an application program interface (API) for printer drivers in the OpenPrinting print environment. The API defined in this specification is meant for use by applications that need to print documents as well as subcomponents of the print system. The API provides an abstraction layer to access the graphics and printing capabilities that a printer supports, either via a printer language or any other means. As a result of this, it becomes possible to use the printer's full potential without the need for detailed knowledge and understanding of each printer model's internals and peculiarities.

The specification defined in this document targets personal devices capable of only raster graphics such as ink-jet printers as well as production systems using high-end laser printers. In particular, this specification provides access to the high level graphics instructions that are available on devices in the latter category, facilitating improved performance.

When an application program prints in the OpenPrinting print environment, the application, either directly or indirectly via supporting libraries, produces “document data”. Normally, this kind of document data uses a device independent page description language such as PostScript or Portable Document Format (PDF). When the printing device can not process such device independent document data directly, the printing system will pass this data to a separate piece of software called the “renderer”. The renderer interprets the device independent document data and transforms it into a sequence of calls to a printer driver using the API entries defined in this document. Based on these calls, the printer driver produces printer commands that the printing device is able to process. The printer commands produced by the printer driver are then sent to the printing device by the print system, finally resulting in printed output.

The previous paragraph merely explains a typical flow of events in the print process but, apart from a renderer placing the API calls defined here, printer drivers may be called on by other software components as well. In the remainder of this document we will refer to any module calling on the printer driver as the “Caller”and simply use “Driver” when talking about the printer driver.

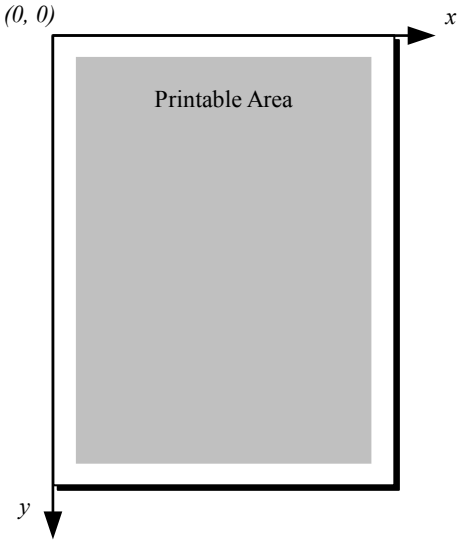
3.Graphic Model

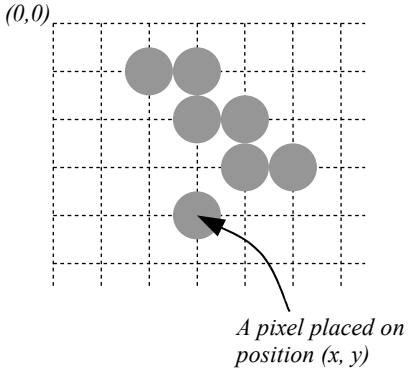
3.1.Coordinate System

The origin of the coordinate system as defined and used in this specification is the physical upper left corner of the media handled by a device. Units of the coordinate system are based on the device resolution. Positive direction on the x-axis is from the origin towards the right side of the media, and positive direction on the y-axis is from the origin towards the bottom of the media.

The type of the coordinate value is a signed fixed point 32 bit value, where the integer part uses 24 bits and the fractional part 8 bits. The type name is `opvp_fix_t`.

Coordinate values *x* and *y* define the horizontal and vertical distance from the origin to a point on the media. A point with integer coordinate values *x* and *y* is on the intersection of the coordinate system grid. Physical device pixels are assumed to be printed on the intersections of the grid (Grid Intersection Model). The relation between coordinate grid and device pixels is shown in the bottom figure on the right.





3.2.Graphics State Object

For each printer, a printer driver MUST maintain a “Graphics State Object” which contains properties and drawing attributes used for drawing graphics for a printer. When calling a printer driver, the caller can specify only one Graphics State Object to the driver. However, caller and driver MAY maintain multiple Graphics State Objects to control different printers, and even save and restore the properties and drawing attributes of each Graphics State Object using the `opvpSaveGS()` and `opvpRestoreGS()` functions.

A graphics State Object MUST provide the following properties and drawing attributes. Please refer to the description of the related functions for more details on the properties and drawing attributes.

Properties	Related functions
CTM	`opvpResetCTM(), opvpSetCTM(), opvpGetCTM()`
Clipping region	`opvpSetClipPath(), opvpResetClipPath()`

Drawing Attributes	Related functions
Color space	`opvpQueryColorSpace(), opvpSetColorSpace(), opvpGetColorSpace()`
Filling mode	`opvpSetFillMode(), opvpGetFillMode()`
Alpha blending constant	`opvpSetAlphaConstant(), opvpGetAlphaConstant()`
Stroke line width	`opvpSetLineWidth(), opvpGetLineWidth()`
Line dash pattern	`opvpSetLineDash(), opvpGetLineDash()`
Line dash pattern offset	`opvpSetLineDashOffset(), opvpGetLineDashOffset()`
Line style	`opvpSetLineStyle(), opvpGetLineStyle()`
Line cap style	`opvpSetLineCap(), opvpGetLineCap()`
Line join style	`opvpSetLineJoin(), opvpGetLineJoin()`
Miter limit value	`opvpSetMiterLimit(), opvpGetMiterLimit()`
Painting mode	`opvpSetPaintMode(), opvpGetPaintMode()`
Stroke line color	`opvpSetStrokeColor()`
Fill color	`opvpSetFillColor()`
Background color	`opvpSetBgColor()`

3.3.CTM

A printer driver MUST maintain a Coordinate Transformation Matrix (CTM) with each Graphics State Object. The CTM is used for transformation from the caller's (renderer) coordinate system to the printer's (device) coordinate system. A CTM transforms the renderer coordinates to device coordinates as follows:

$$\begin{bmatrix} x_{dev} & y_{dev} & 1 \end{bmatrix} = \begin{bmatrix} x_{ren} & y_{ren} & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

To get, set or reset a CTM from the Graphics State Object, use the `opvpGetCTM()`, `opvpSetCTM()`, `opvpResetCTM()` functions.

3.4.Color

The following color spaces are defined by this document.

Color Space Macro	Color Space
`OPVP_CSPACE_BW`	Black and White
`OPVP_CSPACE_DEVICEGRAY`	Gray scale
`OPVP_CSPACE_DEVICECMY`	CMY
`OPVP_CSPACE_DEVICECMYK`	CMY and Black
`OPVP_CSPACE_DEVICERGB`	Device RGB
`OPVP_CSPACE_DEVICEKRGB`	Device KRGB
`OPVP_CSPACE_STANDARDRGB`	sRGB
`OPVP_CSPACE_STANDARDRGB64`	scRGB

A printer driver MUST support at least one of the three color spaces, `OPVP_CSPACE_BW`, `OPVP_CSPACE_DEVICEGRAY` and `OPVP_CSPACE_STANDARDRGB`. Other color spaces are reserved for future extension.

3.5.Graphics Operations

The following graphics operations are defined in this document:

Graphics Operation	Description	Section
Path	Operations for drawing lines, curves, rectangles and other kinds of shapes. A path is maintained as part of a printer context and is shared by all the Graphics State Objects managed by a single printer context. Note: The clipping region is managed by the Graphics State Object, not by the printer context.	4.6
Bitmap Image	Operations for drawing pixel oriented rectangular images.	4.7
Scan Line	Operations for drawing of horizontal lines defined by one or more pairs of begin and end points.	4.8
Raster Image	Operations for drawing whole pages by image instead of using Path Operations and Bitmap Image Operations.	4.9

The color and color space in effect for “Path” and “Scan Line” operations are controlled via the `opvpSetStrokeColor()`, `opvpSetFillColor()` and `opvpSetBgColor()` functions. “Path” and “Scan Line” operations are not affected by the `opvpSetColorSpace()` function.

The image data format for “Bitmap Image” and “Raster Image” operations is dependent on the color space set by the `opvpSetColorSpace()` function.

Image data formats for each color space are defined in the following section.

3.6.Image Data Format

The following image formats are defined in this document:

Color Space	# of Channels	Bits per Channel	Bits per Pixel	Bytes per Pixel	Note
OPVP_CSPACE_BW	1	1	1	1/8	The MSB corresponds to the left and the LSB to the right pixel in the printed result. Padding bits are added to the rightmost byte if necessary. A bit value of "1" indicates white, a value of "0" indicates black.
OPVP_CSPACE_DEVICEGRAY	1	8	8	1	Maximum luminance value is 0xFF. A value of 0xFF indicates white, 0x00 indicates black.
OPVP_CSPACE_STANDARDRGB	3	8	24	3	The order of the channels is R,G,B,R,G,B,R,G,B... The maximum luminance value of each channel is 0xFF.

4.Operations

4.1.Creating and Managing Printer Contexts

This section defines functions to create and delete printer driver contexts. These functions MUST be supported by all drivers.

4.1.1.opvpOpenPrinter

Name

opvpOpenPrinter – Creates a printer context.

Synopsis

```
opvp_dc_t opvpOpenPrinter(
       opvp_int_t outputFD,
       const opvp_char_t *printerModel,
       const opvp_int_t apiVersion[2],
       struct _opvp_api_procs **apiEntry);
```

Arguments

`outputFD` – File descriptor to write the printing data stream to.

`printerModel` – Printer model name (UTF-8 encoded).

`apiVersion` – Vector printer driver API version. `apiVersion[0]` is the major and `apiVersion[1]` is the minor value of the version respectively.

`apiEntry` – Pointer to a structure which stores all API entries of the driver.

Description

This function initializes the driver. The caller MUST specify the file descriptor for writing the printing data stream, and the driver MUST write the printing data stream it generates to the file descriptor. The caller may specify a UTF-8 encoded printer model name via `printerModel`. If the caller passes `NULL` for `printerModel`, the driver SHOULD use the default printer model of the driver.

The caller MUST set the API version number that it expects from the driver via the `apiVersion` array. In case the driver does not support the API version that the caller requests, the driver MUST return an error and set the detailed error code to `OPVP_VERSIONERROR`.

The driver may write debugging messages to `stderr`. Therefore `stderr` MUST NOT be passed as `outputFD` by the caller.

The driver MUST allocate the `struct _opvp_api_procs` buffer and store the address of each driver API entry into to corresponding member of `apiEntry`. If the driver does not support some API entries, it MUST store `NULL` into the corresponding `apiEntry` members.

This function is the only function that MUST be exported by the driver library. The caller MUST refer to and call any other API entry via the addresses stored in the `apiEntry` buffer.

The printer driver MUST return a printer context as the return value of this function. The printer context MUST be a unique number which is managed by the driver. The caller passes the printer context as the first argument when calling other API entries.

Return Value

Printer context value (positive value) or -1 in case of error. In the later case, the driver MUST store a detailed error code in `opvpErrorNo`.

4.1.2.opvpClosePrinter

Name

opvpClosePrinter – Deletes a printer context.

Synopsis

```
opvp_result_t opvpClosePrinter(
       opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function terminates the printing process and deletes the printer context from the printer driver. The caller SHOULD close the file descriptor that was passed as `outputFD` to the `opvpOpenPrinter()` function.

The caller should call the `opvpEndJob()` function before calling this function to declare the end of the printing job. If the caller calls this function before a printing job is completed, in other words, if the caller calls this function before calling the `opvpEndJob()` function, the driver should discard all data from the printing data stream for the printing job. In this situation, the driver need not guarantee that the printing job is canceled normally.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store a detailed error code in `opvpErrorNo`.

4.2.Job, Document and Page Operations

This section defines functions to operate job controls. These functions MUST be supported by all drivers. One printing job consists of one or more documents. One document consists of one or more pages. The caller MUST call the `opvpStartJob()` to declare the start of a printing job, and call the `opvpEndJob()` function to declare the end of the printing job for each job. In addition, it MUST call the `opvpStartDoc()` and `opvpEndDoc()` functions for starting and ending of every document, and it also MUST call the `opvpStartPage()` and `opvpEndPage()` functions for starting and ending of every page. However, if a printing job consists of only one document, the caller can omit calling the `opvpStartDoc()` and `opvpEndDoc()` functions.

4.2.1.opvpStartJob

Name

opvpStartJob – Declares the start of a printing job.

Synopsis

```
opvp_result_t opvpStartJob(
        opvp_dc_t printerContext,
        const opvp_char_t *jobInfo);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`jobInfo` – Printing job property string.

Description

This function declares the start of a printing job.

The caller MUST call this function before calling the `opvpStartDoc()` or `opvpStartPage()` functions or any other drawing API entries.

The caller can set printing job properties via `jobInfo`. The printer driver MUST keep the given printing job properties until the `opvpEndJob()` function is called. When the caller calls the `opvpStartJob()` function again after calling the `opvpEndJob()` function, the driver MUST override the former `jobInfo` with the new `jobInfo` specified by the latest `opvpStartJob()` function call.

If the caller passes `NULL` for the `jobInfo`, the printer driver SHOULD use its default printing job properties. The data format of `jobInfo` is described in the section "Attributes for Job, Document and Page Operations" in this document.

It depends on printer and driver capabilities whether nested printing jobs are supported. A nested printing job is one where the `opvpStartJob()` and `opvpEndJob()` functions are called between an enclosing pair of the `opvpStartJob()` and `opvpEndJob()` functions. If a printer or printer driver does not support nested printing jobs, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.2.2.opvpEndJob

Name

opvpEndJob –Declares the end of a printing job.

Synopsis

```
opvp_result_t opvpEndJob(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function declares the end of a printing job.

The caller MUST call this function after it finishes processing a printing job.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.2.3.opvpAbortJob

Name

opvpAbortJob – Terminates a printing job.

Synopsis

```
opvp_result_t opvpAbortJob(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function aborts a printing job and returns both the printer and the driver to their initial state. After this function has been called, the next printing job MUST be accepted normally. Whether the driver creates printing data and whether any printing data is sent to the printer is implementation dependent. As a result, this function can not guarantee that no consumables are used.

If a caller calls the `opvpEndJob()` function after calling this function, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.2.4.opvpStartDoc

Name

opvpStartDoc – Declares the start of a printing document.

Synopsis

```
opvp_result_t opvpStartDoc(
        opvp_dc_t printerContext,
        const opvp_char_t *docInfo);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`docInfo` – Printing document property string.

Description

This function declares the start of a printing document.

The caller should call this function after calling the `opvpStartJob()` function, and before calling the `opvpStartPage()` function or any other drawing API entries. However, if a printing job consists of only one document, the caller can omit calling the `opvpStartDoc()` and `opvpEndDoc()` functions.

The caller can set document properties via `docInfo`. The printer driver MUST keep the given document properties until the `opvpEndDoc()` function is called. When the caller calls the `opvpStartDoc()` function again after calling the `opvpEndDoc()` function, the driver MUST override the former `docInfo` with the new `docInfo` specified by the latest `opvpStartDoc()` function call.

If the caller passes `NULL` for the `docInfo`, the printer driver SHOULD use its default printing document properties. The data format of `docInfo` is described in the section "Attributes for Job, Document and Page Operations" in this document.

It depends on printer and driver capabilities whether nested documents are supported. A nested document is one where the `opvpStartDoc()` and `opvpEndDoc()` functions are called between an enclosing pair of `opvpStartDoc()` and `opvpEndDoc()` functions. If a printer or printer driver does not support nested documents, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.2.5.opvpEndDoc

Name

opvpEndDoc – Declares the end of a printing document

Synopsis

```
opvp_result_t opvpEndDoc(
        opvp_dc_t printerContext);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

Description

This function declares the end of a printing document.

If the caller called the opvpStartDoc() function, it MUST call this function after it finishes processing a document.

If a printing job consists of one document, the caller can omit calling the opvpStartDoc() and opvpEndDoc() functions.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.2.6.opvpStartPage

Name

opvpStartPage – Declares the start of a printing page.

Synopsis

```
opvp_result_t opvpStartPage(
        opvp_dc_t printerContext,
        const opvp_char_t *pageInfo);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

pageInfo – Printing page property string.

Description

This function declares the start of a printing page.

The caller MUST call this function after calling the opvpStartJob() and opvpStartDoc() functions, and before calling any other drawing API entries.

The caller can set page properties via pageInfo. The printer driver MUST keep the page properties until the opvpEndPage() function is called. When the caller calls the opvpStartPage() function again after calling the opvpEndPage() function, the printer driver MUST override the former pageInfo with the new pageInfo specified by the latest opvpStartPage() function call.

If the caller passes NULL as the pageInfo, the printer driver SHOULD use its default page properties. The data format of pageInfo is described in the section "Attributes for Job, Document and Page Operations" in this document.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.2.7.opvpEndPage

Name

opvpEndPage – Declares the end of a printing page.

Synopsis

```
opvp_result_t opvpEndPage(
       opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function declares the end of a printing page.

The caller MUST call this function after it finishes processing a page.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.3.Query Operations

This section defines functions to query device capabilities and information. Driver support for these functions is OPTIONAL.

4.3.1.opvpQueryDeviceCapability

Name

opvpQueryDeviceCapability – Queries for device capabilities.

Synopsis

```
opvp_result_t opvpQueryDeviceCapability(
       opvp_dc_t printerContext,
       opvp_queryinfoflags_t queryflag,
       opvp_int_t *buflen,
       opvp_char_t *infoBuf);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`queryflag` – Flag specifying which device capability to query.

`buflen` – Number of bytes of the buffer pointed to by *`infoBuf`.

`infoBuf` – Pointer to the buffer to store the device capability.

Description

This function queries the capabilities that are supported by the printer or driver.

The caller MUST set one or more bits of `queryflag` to query capabilities. The values in the following enumeration should be supported by the driver:

```
typedef enum _opvp_queryinfoflags {
  OPVP_QF_DEVICERESOLUTION       = 0x00000001,
  OPVP_QF_MEDIASIZE              = 0x00000002,
  OPVP_QF_PAGEROTATION           = 0x00000004,
  OPVP_QF_MEDIANUP               = 0x00000008,
  OPVP_QF_MEDIADUPLEX            = 0x00000010,
  OPVP_QF_MEDIASOURCE            = 0x00000020,
  OPVP_QF_MEDIADESTINATION       = 0x00000040,
  OPVP_QF_MEDIATYPE              = 0x00000080,
  OPVP_QF_MEDIACOPY              = 0x00000100,/* Maximum copy number supported */
  OPVP_QF_PRINTREGION            = 0x00010000 /* only for opvpQueryDeviceInfo use */
} opvp_queryinfoflags_t;
```

The driver MUST return the queried capabilities in ASCII text format via the `infoBuf` buffer, and also return the number of bytes of this text via *`buflen`. If the buffer is too small to store the query result, the driver MUST return the necessary number of bytes to retrieve the query result in *`buflen`, return an error and set the detailed error code to `OPVP_PARAMERROR`. If the caller passes `NULL` as the `infoBuf` buffer, the driver MUST return the number of bytes required to store the query result via *`buflen`.

The format of the capability name-value pairs stored in the `infoBuf` buffer is the same as that of the `jobInfo` used with the `opvpStartJob()` function. For example, when querying for the device resolution, a resolution list similar to the following may be returned. The first name-value pair in the list indicates the default.

```
updf:DeviceResolution=deviceResolution_600x600,deviceResolution_1200x1200
```

In case of `OPVP_QF_MEDIACOPY`, a driver MUST return the maximum MediaCopy number that a printer supports.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.3.2.opvpQueryDeviceInfo

Name

opvpQueryDeviceInfo – Queries for device information.

Synopsis

```
opvp_result_t opvpQueryDeviceInfo(
       opvp_dc_t printerContext,
       opvp_queryinfoflags_t queryflag,
       opvp_int_t *buflen,
       opvp_char_t *infoBuf);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`queryflag` – Flag specifying which device information to query.

`buflen` – Number of bytes of the buffer pointed to by *`infoBuf`.

`infoBuf` – Pointer to the buffer to store the device information.

Description

This function queries for information about the current settings of the printer or driver.

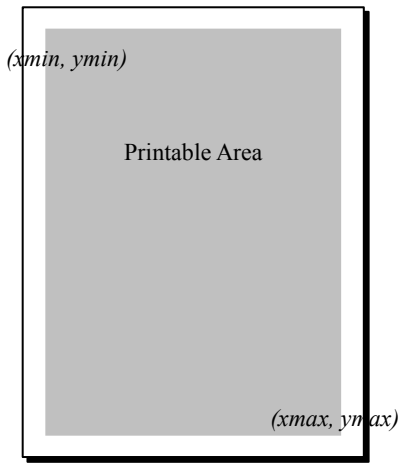
The caller MUST set one or more bits of `queryflag` to query for information. Values of the same enumeration used for the `opvpQueryDeviceCapability()` function MUST be used.

The driver MUST return the queried information in ASCII text format into the `infoBuf` buffer, and also return the number of bytes of this text via *`buflen`. If the buffer is too small to store the query result, the driver MUST return the necessary number of bytes to retrieve the query result in *`buflen`, return an error and set the detailed error code `OPVP_PARAMERROR`. If the caller passes `NULL` as the `infoBuf` pointer, the driver MUST return the number of bytes required to store the query result via *`buflen`.

The format of the information name-value pairs stored in the `infoBuf` buffer is the same as that of the `jobInfo` used with the `opvpStartJob()` function.

When the caller sets the `OPVP_QF_PRINTREGION` bit of the `queryflag`, the driver MUST provide the printable area using the current resolution setting. The printable area format MUST be as given below. The values correspond to the *x* and *y* coordinates of the left top and right bottom corners, respectively, of the current printable area setting as shown in the figure. These values depend on the current media orientation setting.

```
PrintRegion=xmin,ymin,xmax,ymax
```



Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.4.Attributes for Job, Document and Page Operations

Attributes for printing jobs, documents and pages can be passed as arguments to the `opvpStartJob()`, `opvpStartDoc()` and `opvpStartPage()` functions. Supported attributes are provided by a printer driver database file distributed with the driver.

Attribute names and values MUST be specified as ASCII strings in the following format;

<*scheme*>:<*key*>=<*value*>{,<*value*>}*{;<*key*>=<*value*>{,<*value*>}*}*

- <*scheme*>: Name space for the following <*key*> and <*value*>.
- <*key*>: Name of the attribute.
- <*value*>: Value for given <*key*>.

Multiple <*value*>s for a <*key*> MUST be separated by a "," (comma). When multiple values are specified for a single key, the printer driver should search the list of values starting with the first value and use the first value that the driver can use with the current settings.

Multiple key-value pairs, <*key*>=<*value*>{,<*value*>}*, MUST be separated by a ";" (semi-colon).

Conforming drivers MUST support a "updf" <*scheme*> as defined in IEEE-ISTO PWG 5101.4 "Universal Printer Definition Format" (May 2004) developed by the Printer Working Group.

Conforming drivers MUST ignore unknown properties to maintain compatibility with future vendor or standard extensions.

Major UPDF attributes are shown in the table below:

Attribute	Name	Value	Effective in		
			Job	Doc	Page
Orientation	MediaPageRotation	landscape portrait reverse-landscape reverse-portrait	✔	✔	✔
Page Size	MediaSize	iso_a4_210x297mm iso_a3_297x420mm jpn_hagaki_100x148mm ...	✔	✔	✔
Number Up	MediaNUp	nup-1x1 nup-2x1 nup-2x2 ...	✔	✔	✔
Duplex	MediaDuplex	simplex duplex-long-edge duplex-short-edge	✔	✔	✔
Resolution	DeviceResolution	deviceResolution_1200x1200 deviceResolution_600x600 ...	✔	✔	✔
input-bin	MediaSource	manual continuous roll cut-sheet proprietary-value device-setting	✔	✔	✔
output-bin	MediaDestination	standard proprietary-value device-setting	✔	✔	✔
Media Type	MediaType	cardstock continuous stationery stationery-fine ...	✔	✔	✔
Media Copy	MediaCopy	1, 2, ...	✔	✔	
Print Quality	PrintQuality	draft high normal	✔	✔	✔

Between `opvpStartPage()` and `opvpEndPage()` function calls, page attributes take precedence over document attributes. Similarly, between `opvpStartDoc()` and `opvpEndDoc()` function calls, document attributes take precedence over job attributes. For example, a three page job with a landscape orientation attribute passed to the `opvpStartJob()` function and a portrait orientation attribute passed to the `opvpStartPage()` function for its second page will use a landscape orientation for the first page and third page but the second page will use portrait.

4.5.Graphics State Object Operations

This section defines functions that operate on the properties or drawing attributes of a Graphics State Object. Driver support for any of these functions is OPTIONAL.

4.5.1.opvpResetCTM

Name

opvpResetCTM – Initializes the CTM of a Graphics State Object.

Synopsis

```
opvp_result_t opvpResetCTM(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function initializes the CTM of a Graphics State Object. The initial value of a CTM is the identity matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.2.opvpSetCTM

Name

opvpSetCTM – Sets the CTM of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetCTM(
        opvp_dc_t printerContext,
        const opvp_ctm_t *pCTM);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`pCTM` – Pointer to an `opvp_ctm_t` structure holding the six CTM elements a, b, c, d, e and f.

Description

This function sets the CTM of a Graphics State Object.

The printer (device) coordinate system coordinates [x_{dev}, y_{dev}] and the caller (renderer) coordinate system coordinates [x_{ren}, y_{ren}] are related via the CTM as shown in the following equation:

$$\begin{bmatrix} x_{dev} & y_{dev} & 1 \end{bmatrix} = \begin{bmatrix} x_{ren} & y_{ren} & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

Structures

```
typedef struct _opvp_ctm {
        opvp_float_t a, b, c, d, e, f;
} opvp_ctm_t;
```

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.3.opvpGetCTM

Name

opvpGetCTM – Gets the CTM of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetCTM(
        opvp_dc_t printerContext;
        opvp_ctm_t *pCTM);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`pCTM` – Pointer to an `opvp_ctm_t` structure to receive the six CTM elements a, b, c, d, e and f.

Description

This function gets the CTM of a Graphics State Object.

Structures

```
typedef struct _opvp_ctm {
        opvp_float_t a, b, c, d, e, f;
} opvp_ctm_t;
```

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.4.opvpInitGS

Name

opvpInitGS – Initializes the parameters of a Graphics State Object.

Synopsis

```
opvp_result_t opvpInitGS(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function initializes the parameters of the currently active Graphics State Object. The Graphics State is a set of values managed by the printer driver and contains parameters for drawing color, drawing mode and other, drawing related, settings. The driver MUST keep the current parameters of a Graphics State Object unless `opvpInitGS()` or other parameter setting functions are called. Also, the driver MUST save all parameters in the Graphics State Object onto the driver's stack when the caller calls the `opvpSaveGS()` function, and restore all parameters from the stack to the Graphics State Object when the caller calls the `opvpRestoreGS()` function. These two functions are described later in this document. These operations are used to change the parameters in the Graphics State Object temporarily and restore them after drawing operations.

The driver MUST keep the Graphics State Object parameters between `opvpStartJob()` and `opvpEndJob()`, unless `opvpInitGS()` or other parameter setting functions are called. When the caller calls the `opvpStartJob()` function, the driver MUST set the same parameters as the `opvpInitGS()` sets.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.5.opvpSaveGS

Name

opvpSaveGS – Saves the Graphics State Object parameters.

Synopsis

```
opvp_result_t opvpSaveGS(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function saves the Graphics State Object parameters onto the driver's stack.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.6.opvpRestoreGS

Name

opvpRestoreGS – Restores Graphics State Object parameters.

Synopsis

```
opvp_result_t opvpRestoreGS(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function retrieves Graphics State Object parameters from the driver's stack and restores them into the current Graphics State Object.

This function MUST return error and set the error code `OPVP_BADREQUEST` in `opvpErrorNo` if the stack is empty.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.7.opvpQueryColorSpace

Name

opvpQueryColorSpace – Queries the driver for supported color spaces.

Synopsis

```
opvp_result_t opvpQueryColorSpace(
        opvp_dc_t printerContext,
        opvp_int_t *pnum,
        opvp_cspace_t *pcspace);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`pnum` – Number of elements in `pcspace`.

`pcspace` – Pointer to an array of supported color spaces.

Description

This function queries the driver for a list of supported color spaces.

To query for the supported color spaces , the caller MUST set the number of `pcspace` elements into *`pnum`. In this case, the driver MUST return the supported color spaces in the `pcspace` array and also return the number of supported color spaces retrieved in `*pnum`.

If the caller sets `NULL` in `*pcspace`, the driver MUST return only the number of supported color spaces in `*pnum`.

If the number of supported color spaces exceeds the number of `pcspace` array elements prepared by the caller, the driver MUST return the necessary number to retrieve all supported color spaces in `*pnum`, return an error and set the detailed error code to `OPVP_PARAMERROR`.

The driver SHOULD return the supported color spaces in order of preference. The first supported color spaces is the most preferred color space for the driver.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.8.opvpSetColorSpace

Name

opvpSetColorSpace – Sets the color space of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetColorSpace(
       opvp_dc_t printerContext,
       opvp_cspace_t cspace);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`cspace` – Color space value.

Description

This function sets the color space of a Graphics State Object.

The color space passed by the caller via `cspace` MUST be one of the color spaces returned by the `opvpQueryColorSpace()` function.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.9.opvpGetColorSpace

Name

opvpGetColorSpace – Gets the current color space from a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetColorSpace(
       opvp_dc_t printerContext,
       opvp_cspace_t *pcspace);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`pcspace` – Pointer to the color space value to be returned.

Description

This function gets the color space that is currently set in a Graphics State Object.

The initial color space of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.10.opvpSetFillMode

Name

opvpSetFillMode – Sets the filling mode of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetFillMode(
       opvp_dc_t printerContext,
       opvp_fillmode_t fillmode);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`fillmode` – Filling mode enumeration value. `OPVP_FILLMODE_EVENODD` (even-odd rule) and `OPVP_FILLMODE_WINDING` (non-zero winding number rule) can be set.

Description

This function sets the filling mode of a Graphics State Object.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.11.opvpGetFillMode

Name

opvpGetFillMode – Gets the filling mode from a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetFillMode(
       opvp_dc_t printerContext,
       opvp_fillmode_t *pfillmode);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`pfillmode` – Pointer to the fill mode enumeration value to be returned.

Description

This function gets the filling mode that is currently set in a Graphics State Object.

The initial filling mode of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.12.opvpSetAlphaConstant

Name

opvpSetAlphaConstant – Sets the alpha blending constant of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetAlphaConstant(
       opvp_dc_t printerContext,
       opvp_float_t alpha);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`alpha` – Alpha blending constant. This MUST be a value between 0.0 and 1.0.

Description

This function sets the alpha blending constant, which is transparent ratio, of a Graphics State Object. The value of `alpha` MUST be between 0.0 and 1.0. If the value specified by the caller is outside of this range, the driver SHOULD use the nearest value that is within range.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.13.opvpGetAlphaConstant

Name

opvpGetAlphaConstant – Gets the alpha blending constant from a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetAlphaConstant(
       opvp_dc_t printerContext,
       opvp_float_t *palpha);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`palpha` – Pointer to the alpha constant value to be returned.

Description

This function gets the alpha constant value that is currently set in a Graphics State Object.

The initial alpha blending constant of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.14.opvpSetLineWidth

Name

opvpSetLineWidth – Sets the line width of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetLineWidth(
       opvp_dc_t printerContext,
       opvp_fix_t width);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`width` – Line width value.

Description

This function sets line width for stroke operations by a Graphics State Object. The line width MUST be set in caller coordinate system unit.

Treatment of line widths less than one depends on the device or driver implementation. Similarly, the maximum line width that can be set also depends on the device or driver implementation.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.15.opvpGetLineWidth

Name

opvpGetLineWidth – Gets the line width from a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetLineWidth(
       opvp_dc_t printerContext,
       opvp_fix_t *pwidth);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`pwidth` – Pointer to the line width value to be returned.

Description

This function gets the line width for stroke operations from a Graphics State Object. The line width value MUST be in caller coordinate system unit.

The initial line width of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.16.opvpSetLineDash

Name

opvpSetLineDash – Sets the line dash pattern of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetLineDash(
       opvp_dc_t printerContext,
       opvp_int_t num,
       const opvp_fix_t *pdash);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`num` – Number of elements in `pdash`.

`pdash` – Pointer to the line dash pattern array.

Description

This function sets a line dash pattern for use by a Graphics State Object.

When the painting mode is `OPVP_PAINTMODE_OPAQUE` (opaque mode), the odd numbered (1st, 3rd, 5th, 7th ...) elements of the `pdash` array indicate the length for dashes in the color set by the `opvpSetStrokeColor()` function and the even numbered (2nd, 4th, 6th, 8th...) elements indicate the length for dashes in the color set by the `opvpSetBgColor()` function.

When the painting mode is `OPVP_PAINTMODE_TRANSPARENT` (transparent mode), the odd numbered (1st, 3rd, 5th, 7th ...) elements of the `pdash` array indicate the length for dashes in the color set by the `opvpSetStrokeColor()` function and the even numbered (2nd, 4th, 6th, 8th...) elements indicate the length of line segments that are not painted.

The lengths set in the `pdash` array MUST be in caller coordinate system unit.

If the number of elements of the `pdash` array is odd, the dash pattern is created as if the number of elements is double the number of elements of the array. In this case, the first element of the `pdash` array in the second cycle is treated as the length for background color dashes in case of opaque mode and for non-painted line segments in case of transparent mode.

The maximum number of elements that can be set depends on the device or driver implementation.

If the caller passes zero for `num`, the driver MUST draw solid lines.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.5.17.opvpGetLineDash

Name

opvpGetLineDash – Gets the line dash pattern of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetLineDash(
       opvp_dc_t printerContext,
       opvp_int_t *pnum,
       opvp_fix_t *pdash);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

pnum – Pointer to the number of elements in pdash.

pdash – Pointer to the buffer to store the line dash pattern array.

Description

This function gets the line dash pattern from a Graphics State Object.

If the caller sets NULL in *pdash, the driver MUST return only the number of the dash pattern elements in *pnum.

If the caller sets a non-NULL value in *pdash, the caller MUST also set the number of pdash elements into *pnum. In this case, the driver MUST return the dash pattern in the pdash array and also return the number of dash pattern elements retrieved in *pnum.

If the number of dash pattern elements exceeds the number of pdash array elements prepared by the caller, the driver MUST return the necessary number to retrieve all the dash pattern elements in *pnum, return an error and set the detailed error code to OPVP_PARAMERROR.

The initial line dash pattern of a Graphics State Object is driver dependent.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.5.18.opvpSetLineDashOffset

Name

opvpSetLineDashOffset – Sets the line dash pattern offset of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetLineDashOffset(
       opvp_dc_t printerContext,
       opvp_fix_t offset);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

offset – Offset value of the line dash pattern.

Description

This function sets the offset value of the line dash pattern for stroke operations in caller coordinate system unit.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.5.19.opvpGetLineDashOffset

Name

opvpGetLineDashOffset – Gets the line dash pattern offset of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetLineDashOffset(
       opvp_dc_t printerContext,
       opvp_fix_t *poffset);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`poffset` – Pointer to the offset value of the line dash pattern to be returned.

Description

This function gets the offset value of the line dash pattern from a Graphics State Object.

The initial line dash pattern offset of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.20.opvpSetLineStyle

Name

opvpSetLineStyle – Sets the line style of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetLineStyle(
       opvp_dc_t printerContext,
       opvp_linestyle_t linestyle);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`linestyle` – Line style enumeration value. `OPVP_LINESTYLE_SOLID` (solid line) and `OPVP_LINESTYLE_DASH` (dashed line) can be set.

Description

This function sets the line style of a Graphics State Object.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.21.opvpGetLineStyle

Name

opvpGetLineStyle – Gets the line style of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetLineStyle(
       opvp_dc_t printerContext,
       opvp_linestyle_t *plinestyle);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`plinestyle` – Pointer to the line style enumeration value to be returned.

Description

This function gets the line style from a Graphics State Object.

The initial line style of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.22.opvpSetLineCap

Name

opvpSetLineCap – Sets the line cap style of a Graphics State Object.

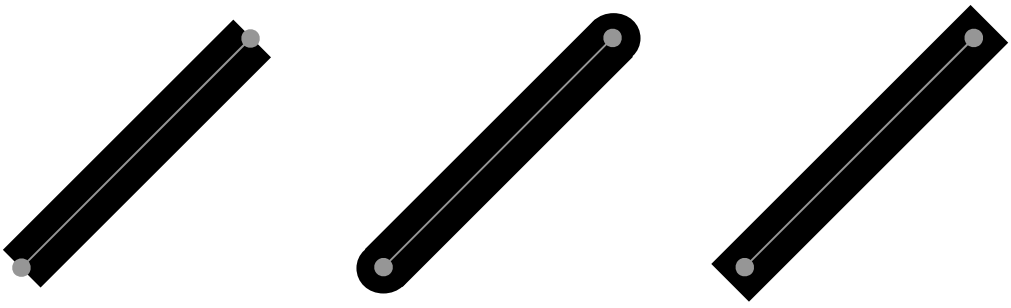
Synopsis

```
opvp_result_t opvpSetLineCap(
       opvp_dc_t printerContext,
       opvp_linecap_t linecap);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`linecap` – Line cap style enumeration value. `OPVP_LINECAP_BUTT` (butt cap), `OPVP_LINECAP_ROUND` (round cap) and `OPVP_LINECAP_SQUARE` (square projection cap) can be set.



Butt cap Round cap Square projection cap

Description

This function sets the line cap style of a Graphics State Object.

The line cap style specifies the shape at the end of opened paths, and also specifies the shapes at the end of each dashes.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.23.opvpGetLineCap

Name

opvpGetLineCap – Gets the line cap style of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetLineCap(
       opvp_dc_t printerContext,
       opvp_linecap_t *plinecap);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`plinecap` – Pointer to the line cap style enumeration value to be returned.

Description

This function gets the line cap style of a Graphics State Object.

The initial line cap style of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.24.opvpSetLineJoin

Name

opvpSetLineJoin – Sets the line join style of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetLineJoin(
       opvp_dc_t printerContext,
       opvp_linejoin_t linejoin);
```

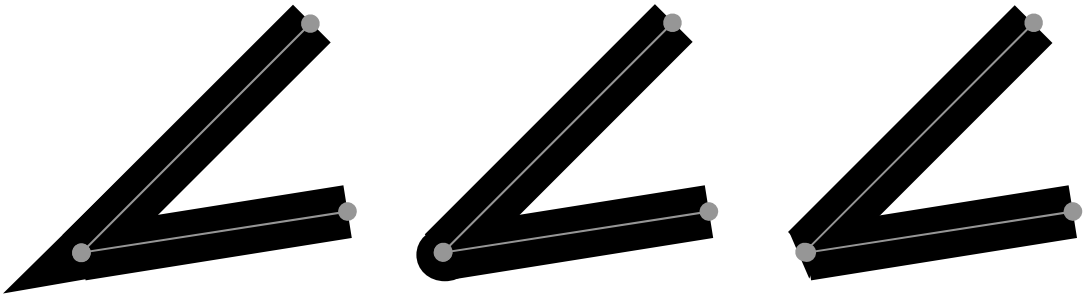
Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`linejoin` – Line join style enumeration value. `OPVP_LINEJOIN_MITER` (miter join), `OPVP_LINEJOIN_ROUND` (round join) and `OPVP_LINEJOIN_BEVEL` (bevel join) can be set.

Description

This function sets the line join style of a Graphics State Object.



Miter join Round join Bevel join

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.25.opvpGetLineJoin

Name

opvpGetLineJoin – Gets the line join style of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetLineJoin(
       opvp_dc_t printerContext,
       opvp_linejoin_t *plinejoin);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`plinejoin` – Pointer to the line join style enumeration value to be returned.

Description

This function gets the line join style of a Graphics State Object.

The initial line join style of a Graphics State Object is driver dependent.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.5.26.opvpSetMiterLimit

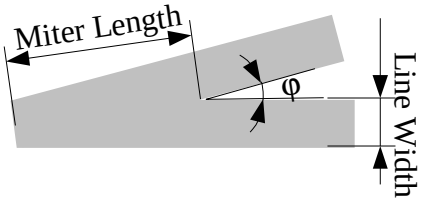
Name

opvpSetMiterLimit – Sets the miter limit value of a Graphics State Object.

$$Miter\ Limit = \frac{Miter\ Length}{Line\ Width} = \frac{1}{\sin(\frac{\varphi}{2})}$$

Synopsis

```
opvp_result_t opvpSetMiterLimit(
        opvp_dc_t printerContext,
        opvp_fix_t miterlimit);
```



Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

miterlimit – Maximum miter length.

Description

This function sets the maximum length of the miter created by a Graphics State Object. The miter limit is effective only when the line join style is OPVP_LINEJOIN_MITER. The length MUST be set in caller coordinate system unit.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.5.27.opvpGetMiterLimit

Name

opvpGetMiterLimit – Gets the miter limit value of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetMiterLimit(
        opvp_dc_t printerContext,
        opvp_fix_t *pmiterlimit);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

pmiterlimit – Pointer to the maximum miter length to be returned.

Description

This function gets the maximum length of miter of a Graphics State Object.

The initial miter limit value of a Graphics State Object is driver dependent.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.5.28.opvpSetPaintMode

Name

opvpSetPaintMode – Sets the background color painting mode of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetPaintMode(
        opvp_dc_t printerContext,
```

```
        opvp_paintmode_t paintmode);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`paintmode` – Painting mode enumeration value. `OPVP_PAINTMODE_OPAQUE` (opaque mode) and `OPVP_PAINTMODE_TRANSPARENT` (transparent mode) can be set.

Description

This function sets the background color painting mode of a Graphics State Object.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.29.opvpGetPaintMode

Name

opvpGetPaintMode – Gets the background color painting mode of a Graphics State Object.

Synopsis

```
opvp_result_t opvpGetPaintMode(
        opvp_dc_t printerContext,
        opvp_paintmode_t *ppaintmode);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`ppaintmode` – Pointer to the painting mode enumeration value to be returned.

Description

This function gets the background color painting mode of a Graphics State Object.

The initial painting mode of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.30.opvpSetStrokeColor

Name

opvpSetStrokeColor – Sets the stroke color and pattern for the `opvpStrokePath()` and `opvpStrokeFillPath()` operations of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetStrokeColor(
        opvp_dc_t printerContext,
        const opvp_brush_t *brush);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`brush` – Pointer to the `opvp_brush_t` structure data.

Description

This function sets the color or pattern to be used by the Graphics State Object when drawing strokes as instructed by the `opvpStrokePath()` and `opvpStrokeFillPath()` operations.

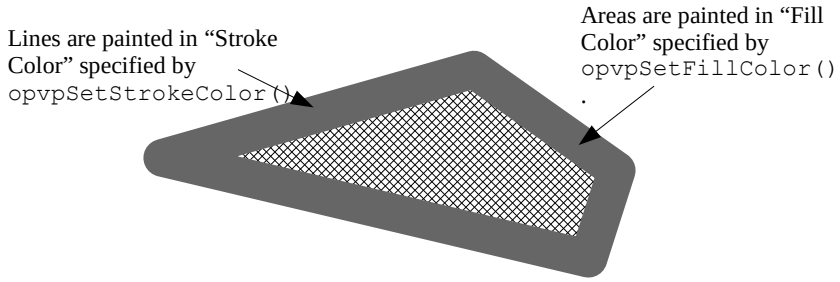
If the `pbrush` member of the `*brush` `opvp_brush_t` structure is `NULL`, the driver MUST treat it as a solid brush. In this case, the brush's color value is specified in the `color[]` array of the *`brush` `opvp_brush_t` structure. The `color[]` array is interpreted according to the brush's `colorSpace`.

If the `pbrush` member of the `*brush` `opvp_brush_t` structure points to a `opvp_brushdata_t` structure, the driver MUST treat it as a pattern brush. The `opvp_brushdata_t` structure's `width` and `height` members are specified in pixels,

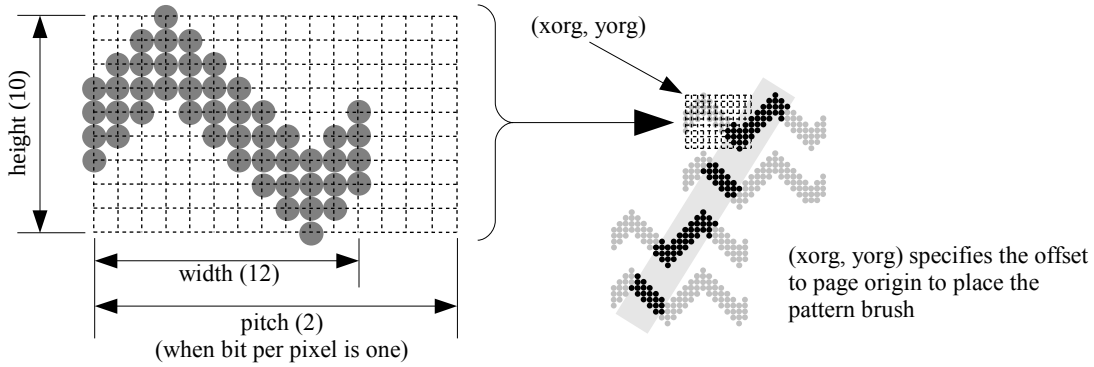
and the brush pattern's horizontal repetition `pitch` is specified as a number of bytes. The actual pattern data is specified in the `data` array. The pattern is interpreted according to the brush's `colorSpace`, and the `color[]` array MUST be ignored.

The `type` member of the `opvp_brushdata_t` structure is intended for future extensions. To maintain compatibility between the current and future versions of the specification, the caller MUST set this member to `OPVP_BDTYPE_NORMAL` for the current version of this specification.

The initial stroke color of a Graphics State Object is driver dependent.



“Stroke Color” vs. “Fill Color”



Pattern Brush Example

Structures

```
typedef struct _opvp_brushdata {
       opvp_bdtype_t type;
       opvp_int_t width, height, pitch;
       opvp_byte_t data[];

} opvp_brushdata_t;

typedef struct _opvp_brush {
       opvp_cspace_t colorSpace;
       opvp_int_t color[4];
       opvp_int_t xorg, yorg;
       opvp_brushdata_t *pbrush;
} opvp_brush_t;

typedef enum _opvp_bdtype {OPVP_BDTYPE_NORMAL = 0} opvp_bdtype_t;
```

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.31.opvpSetFillColor

Name

opvpSetFillColor – Sets the fill color and pattern for the `opvpFillPath()` and `opvpStrokeFillPath()` operations of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetFillColor(
```

```
        opvp_dc_t printerContext,
        const opvp_brush_t *brush);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`brush` – Pointer to brush structure data.

Description

This function registers the color or pattern to be used by the Graphics State Object when filling areas as instructed by the `opvpFillPath()` and `opvpStrokeFillPath()` operations.

The definition and behavior of the `opvp_brush_t` structure `*brush` is identical to that specified for the `opvpSetStrokeColor()` function.

The initial fill color of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.32.opvpSetBgColor

Name

opvpSetBgColor – Sets the background color for the `opvpStrokePath()`, `opvpFillPath()` and `opvpStrokeFillPath()` operations of a Graphics State Object.

Synopsis

```
opvp_result_t opvpSetBgColor(
        opvp_dc_t printerContext,
        const opvp_brush_t *brush);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`brush` – Pointer to brush structure data.

Description

This function sets the background color for the `opvpStrokePath()`, `opvpFillPath()` and `opvpStrokeFillPath()` operations.

The caller MUST set a solid brush via the `*brush` parameter. This is achieved by setting the `pbrush` member of the `opvp_brush_t` structure to `NULL`. If the caller fails to do so, the driver MUST return and error and set the detailed error code to `OPVP_BADREQUEST`.

The driver MAY refer only to the `colorSpace` and `color[4]` members of the `opvp_brush_t` structure, and MUST ignore the `xorg`, `yorg` and `pbrush` members.

The initial background color of a Graphics State Object is driver dependent.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.Path Operations

This sections defines functions for path operations.

A path is used for drawing commands such as “stroke”, “fill” and “stroke and fill”, as well as for defining a clipping area. Any printer context maintains only one path at any one time, and all Graphics State Objects managed by a printer context share the same path. Paths are specified in the caller's coordinate system when passed to path operation functions. The driver transforms the given path to the printer's coordinate system by means of the CTM. The resulting path is registered with the printer context.

Driver support for any of the path operation functions is OPTIONAL.

4.6.1.opvpNewPath

Name

opvpNewPath – Starts a new path.

Synopsis

```
opvp_result_t opvpNewPath(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function deletes the printer context's current path and starts a new path. The new path MUST be empty.

The initial path of a printer context MUST be empty.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.2.opvpEndPath

Name

opvpEndPath – Declares the end of the current path.

Synopsis

```
opvp_result_t opvpEndPath(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function declares the end of the printer context's current path.

The printer context MUST retain the path as the current path.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.3.opvpStrokePath

Name

opvpStrokePath – Draws lines along the current path.

Synopsis

```
opvp_result_t opvpStrokePath(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function draws lines along the current path according to the drawing attributes registered with the Graphics State Object.

The current path MUST be retained in the printer context after calling this function.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.4.opvpFillPath

Name

opvpFillPath – Fills the interior of the current path.

Synopsis

```
opvp_result_t opvpFillPath(
       opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function fills the interior of the current path according to the drawing attributes registered with the Graphics State Object.

When the path is not closed, the starting point and end point of the current path are connected by a straight line to determine the interior of the path.

The current path MUST be retained in the printer context after calling this function.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.5.opvpStrokeFillPath

Name

opvpStrokeFillPath – Draws lines along the current path and fills the interior of the current path.

Synopsis

```
opvp_result_t opvpStrokeFillPath(
       opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function draws lines along the current path, and fills the interior of the current path according to the drawing attributes registered with the Graphics State Object.

When the path is not closed, the starting point and end point of the current path are connected by a straight line. This line is used solely to determine the interior of the path and not stroked.

The current path MUST be retained in the printer context after calling this function.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.6.opvpSetClipPath

Name

opvpSetClipPath – Sets the current path as the clipping region.

Synopsis

```
opvp_result_t opvpSetClipPath(
        opvp_dc_t printerContext,
        opvp_cliprule_t clipRule);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`clipRule` – Clipping rule enumeration value. `OPVP_CLIPRULE_EVENODD` (even-odd rule) and `OPVP_CLIPRULE_WINDING` (non-zero winding number rule) can be set.

Description

This function sets the current path as a clipping region for the Graphics State Object.

When the path is not closed, the starting point and end point of the current path are connected by a straight line and closed.

The current path MUST be retained in the printer context after calling this function.

The clipping region also MUST be retained until the caller invokes one of: `opvpSetClipPath()`, `opvpResetClipPath()`, `opvpRestoreGS()` or `opvpStartPage()`. When the caller calls the `opvpStartPage()` function, the driver MUST reset the clipping region as if the `opvpResetClipPath()` function had been called.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.7.opvpResetClipPath

Name

opvpResetClipPath – Resets the clipping region.

Synopsis

```
opvp_result_t opvpResetClipPath(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function resets the clipping region of a Graphics State Object to cover all of the printable area of the media.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.8.opvpSetCurrentPoint

Name

opvpSetCurrentPoint – Sets the current point.

Synopsis

```
opvp_result_t opvpSetCurrentPoint(
        opvp_dc_t printerContext,
        opvp_fix_t x,
        opvp_fix_t y);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`x` – value of the *x* coordinate of the new current point.

`y` – value of the *y* coordinate of the new current point.

Description

This function set the current point of the printer context to *(x, y)*.

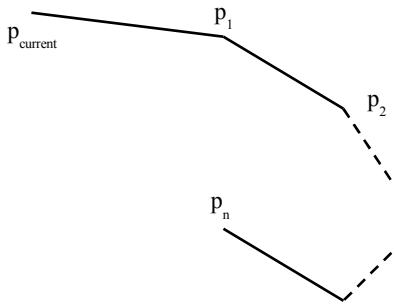
Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.9.opvpLinePath

Name

opvpLinePath – Adds multiple, connected line segments to the current path.



Synopsis

```
opvp_result_t opvpLinePath(
        opvp_dc_t printerContext,
        opvp_pathmode_t flag;
        opvp_int_t npoints,
        const opvp_point_t *points);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`flag` – Path open/close flag. One of `OPVP_PATHCLOSE` and `OPVP_PATHOPEN` can be set.

`npoints` – Number of elements in `points`.

`points` – Pointer to an `opvp_point_t` structure array.

Description

This function adds the line segments specified by the `points` array starting at the current point.

When the caller passes `OPVP_PATHOPEN` as `flag`, the driver MUST append the line segments from the current point and make the last point of the `points` array the new current point. When the caller passes `OPVP_PATHCLOSE` as `flag`, the driver MUST append the line segments from the current point and make the first point of the `points` array the new current point.

Structures

```
typedef struct _opvp_point {
        opvp_fix_t x, y;
} opvp_point_t;
```

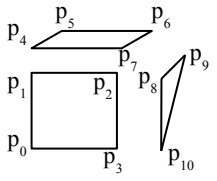
Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.10.opvpPolygonPath

Name

opvpPolygonPath – Adds polygons to the current path.



For the above case, each of the arguments is as follows:
npolygons=3
*nvertices={4, 4, 3}
*points={p0, p1, … p10}

Synopsis

```
opvp_result_t opvpPolygonPath(
        opvp_dc_t printerContext,
        opvp_int_t npolygons,
        const opvp_int_t *nvertices,
        const opvp_point_t *points);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`npolygons` – Number of polygons to add.

`nvertices` – Pointer to an array with the number of points of each polygon.

`points` – Pointer to an `opvp_point_t` structure array. The number of points in this array MUST be equal to the sum total of the values in the `nvertices` array.

Description

This function adds the polygons specified via its arguments to the current path.

After the polygons have been added, the driver MUST make the last point of the `points` array the new current point.

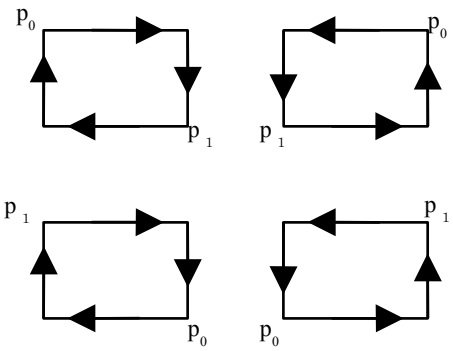
Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.11.opvpRectanglePath

Name

opvpRectanglePath – Adds rectangles to the current path.



Synopsis

```
opvp_result_t opvpRectanglePath(
       opvp_dc_t printerContext,
       opvp_int_t nrectangles,
       const opvp_rectangle_t *rectangles);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`nrectangles` – Number of rectangles to add.

`rectangles` – Pointer to an `opvp_rectangle` structure array.

Description

This function adds rectangles to the current path.

After the rectangles have been appended, the driver MUST make the starting point of the last rectangle appended the new current point.

The direction of the paths of each rectangle are specified by the starting and diagonal points as illustrated in the figure above. The path is appended in order *(x0, y0)-(x1, y0)-(x1, y1)-(x0, y1)-(x0, y0)* where the starting point p0 is *(x0,y0)* and the diagonal point p1 is *(x1,y1)*.

Structures

```
typedef struct _opvp_rectangle {
       opvp_point_t p0;                /* starting point */
       opvp_point_t p1;                /* diagonal point */
} opvp_rectangle_t;
```

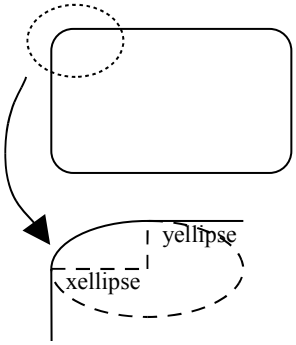
Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.12.opvpRoundRectanglePath

Name

opvpRoundRectanglePath – Adds rounded rectangles to the current path.



Synopsis

```
opvp_result_t opvpRoundRectanglePath(
       opvp_dc_t printerContext,
       opvp_int_t nrectangles,
       const opvp_roundrectangle_t *rectangles);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`nrectangles` – Number of rounded rectangles to add.

`rectangles` – Pointer to an `opvp_roundrectangle_t` structure array.

Description

This function adds rounded rectangles to the current path.

Each corner of a rounded rectangle is connected by an elliptic arc defined by the values of the `xellipse` and `yellipse` members of the `opvp_roundrectangle_t` structure. After the rounded rectangles have been appended, the driver MUST make the starting point of the last rounded rectangle appended the new current point.

The direction of the paths of each rounded rectangle MUST follow the conventions as specified for the `opvpRectanglePath()` function.

Structures

```
typedef struct _opvp_roundrectangle {
        opvp_point_t p0;                /* starting point */
        opvp_point_t p1;                /* diagonal point */
        opvp_fix_t xellipse, yellipse;
} opvp_roundrectangle_t;
```

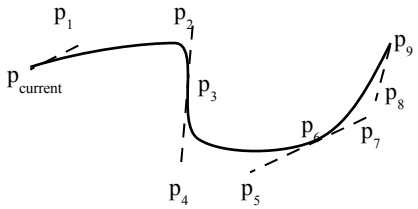
Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.13.opvpBezierPath

Name

opvpBezierPath – Adds Bézier path segments to the path.



In the above case, each arguments are as following:
npoints = 9
*points = {p_1, p_2, … p_9}

Synopsis

```
opvp_result_t opvpBezierPath(
        opvp_dc_t printerContext,
        opvp_int_t npoints,
        const opvp_point_t *points);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`npoints` – Number of elements in `points`. It MUST be multiple of 3.

`points` – Pointer to the array of end and control points of Bézier curves.

Description

This function adds multiple Bézier path segments to the current path. Each path segment is specified by four points. The current point is the start point of the segment and the first point in the `points` array is used to define the start tangent vector. The second point in the `points` array is used to define the end tangent vector and the third point defines the segment's end point. The segment's end point becomes the path's new current point. The next path segment is constructed in identical fashion using the next three points in the `points` array. This process repeats until all points of the `points` array have been used.

After the driver has appended the Bézier path segments, the driver MUST set the last point of the `points` array as the current point.

If `npoints` is not a multiple of 3, this function MUST return an error and set the detailed error code to `OPVP_PARAMERROR`.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.14.opvpArcPath

Name

opvpArcPath – Adds arcs, chords, or pies to the current path.

Synopsis

```
opvp_result_t opvpArchPath(
       opvp_dc_t printerContext,
       opvp_arcmode_t kind,
       opvp_arcdir_t dir,
       opvp_fix_t bbx0, opvp_fix_t bby0, opvp_fix_t bbx1, opvp_fix_t bby1,
       opvp_fix_t x0, opvp_fix_t y0,
       opvp_fix_t x1, opvp_fix_t y1);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

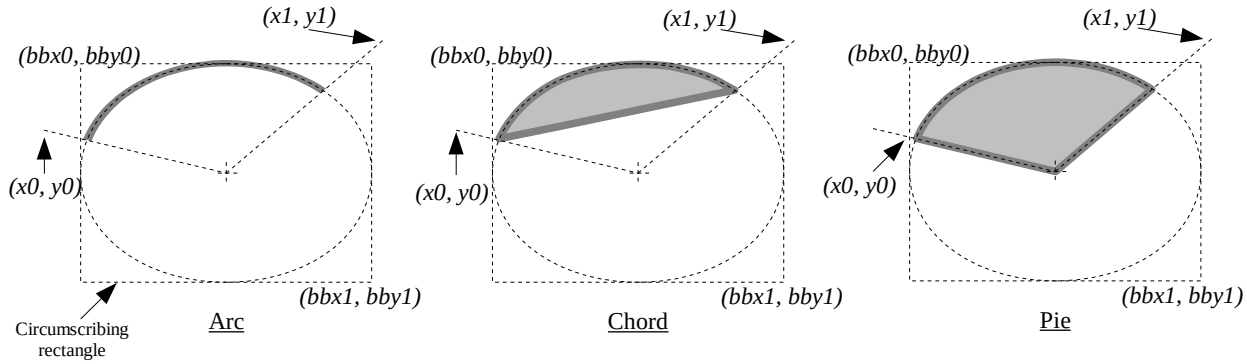
`kind` – Type of arc path. `OPVP_ARC`(Arc), `OPVP_CHORD`(Chord), `OPVP_PIE`(Pie) can be set.

`dir` – Direction of the path. `OPVP_CLOCKWISE`(Clockwise), `OPVP_COUNTERCLOCKWISE`(Counter-clockwise) can be set.

`bbx0`, `bby0`, `bbx1`, `bby1` – Circumscribing rectangle.

`x0`, `y0` – Starting point.

`x1`, `y1` – End point.



Description

This function adds an arc, a chord or a pie to the current path. The center point of ellipse is the middle point of the circumscribe rectangle. The direction of the path is specified by `dir`. When `OPVP_ARC` is set into `kind` and the same point is set into both start and end points, driver MUST append ellipse into the path. If the circumscribe rectangle is a square, driver MUST adds circle to the path.

After driver adds paths, in case of arc, driver MUST set the end point of the arc as the current point. In case of chord or pie, driver MUST set the left-top point of the circumscribe rectangle as the current point.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.7.Bitmap Image Operations

This section defines functions for bitmap operations.

A bitmap is a pixel oriented image format for rectangular images. A bitmap is drawn using the drawing attributes registered with a Graphics State Object.

The typical bitmap drawing sequence is as follows:

- `opvpStartDrawImage()` – Declares the start of a bitmap drawing sequence and sets up the Graphics State Object.
- `opvpTransferDrawImage()` – Transfers the actual image data. The caller typically invokes this function one or more times.
- `opvpEndDrawImage()` – Declares the end of a bitmap drawing sequence.

If the caller calls any functions other than the `opvpTransferDrawImage()` or `opvpAbortJob()` function between the `opvpStartDrawImage()` and `opvpEndDrawImage()` functions, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

The `opvpDrawImage()` convenience function performs the `opvpStartDrawImage()`, `opvpTransferDrawImage()` and `opvpEndDrawImage()` function calls as a single function call.

All bitmap operation functions are OPTIONAL.

4.7.1.opvpDrawImage

Name

opvpDrawImage – Draws a bitmap image

Synopsis

```
opvp_result_t opvpDrawImage(
       opvp_dc_t printerContext,
       opvp_int_t sourceWidth,
       opvp_int_t sourceHeight,
       opvp_int_t sourcePitch,
       opvp_imageformat_t imageFormat,

       opvp_int_t destinationWidth,
       opvp_int_t destinationHeight,
       const void *imageData);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`sourceWidth` – Width (in pixels) of the source image.

`sourceHeight` – Height (in pixels) of the source image.

`sourcePitch` – Length (in bytes) of each line of image data.

`imageFormat`– Image format of the source image.

`destinationWidth` – Destination drawing width (in caller coordinate system units).

`destinationHeight` – Destination drawing height (in caller coordinate system units).

`imageData` – Pointer to the source image data.

Description

This function is a convenience function to transfer a bitmap image in a single function call. It is functionally equivalent to the following:

```
opvpStartDrawImage (printerContext, sourceWidth, sourceHeight, sourcePitch,
       imageFormat, destinationWidth, destinationHeight);
opvpTransferDrawImage(printerContext, sourceHeight * sourcePitch, imageData);
opvpEndDrawImage(printerContext);
```

See the respective functions for more details.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.7.2.opvpStartDrawImage

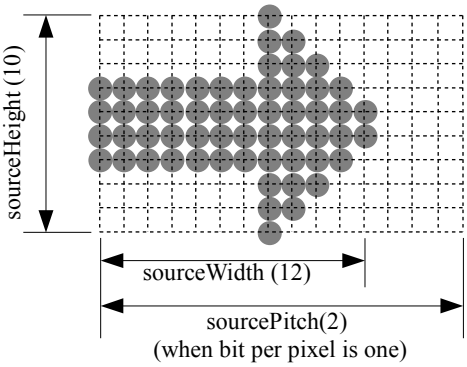
Name

opvpStartDrawImage – Declares the start of a bitmap drawing sequence.

Synopsis

```
opvp_result_t opvpStartDrawImage(
        opvp_dc_t printerContext,
        opvp_int_t sourceWidth,
        opvp_int_t sourceHeight,
        opvp_int_t sourcePitch,
        opvp_imageformat_t imageFormat,

        opvp_int_t destinationWidth,
        opvp_int_t destinationHeight
        );
```



Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`sourceWidth` – Width (in pixels) of the source image.

`sourceHeight` – Height (in pixels) of the source image.

`sourcePitch` – Length (in bytes) of each line of image data.

`imageFormat`– Image format of the source image.

`destinationWidth` – Destination drawing width (in caller coordinate system units).

`destinationHeight` – Destination drawing height (in caller coordinate system units).

Description

This function declares the start of a bitmap image drawing operation. The destination image drawing area is specified by means of the current point kept in the printer context and `destinationWidth` and `destinationHeight`. The upper-left corner of the image MUST be drawn on the current point and the bottom-right corner of the image MUST be drawn at $(x_{current} + destinationWidth - 1, y_{current} + destinationHeight - 1)$ where $x_{current}$ is the current point *x* coordinate value and $y_{current}$ is the current point *y* coordinate value.

If the width or height of source and destination are not equal, the image are expected to be drawn scaled. If the driver does not support image scaling, this function MAY return an error and set the detailed error code to `OPVP_NOTSUPPORTED`.

If the current point coordinate value is not an integer, the reference point for drawing the image MAY be rounded to an integer.

After the caller calls this function, the caller MUST call the `opvpTransferDrawImage()` function once or more to transfer the actual image data to the driver, or call the `opvpAbortJob()` to abort the printing job. If the caller calls any other function between the `opvpStartDrawImage()` and `opvpEndDrawImage()` function, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

The following image formats are defined in this document;

Image Format	Description
`OPVP_IFORMAT_RAW`	Raw image data. The image data format corresponds to that for the current color space. The caller MUST call the `opvpSetColorSpace()` function before calling this function. The current color space value MUST be one of the color spaces supported by the driver.
`OPVP_IFORMAT_MASK`	Mask. The image data format corresponds to that of the `OPVP_CSPACE_BW` color space. When the painting mode is `OPVP_PAINTMODE_OPAQUE` (opaque mode) a “1” bit in the image data MUST be drawn using the current fill color, and a “0” bit MUST be drawn using the color set by the `opvpSetBgColor()` function. When the painting mode is `OPVP_PAINTMODE_TRANSPARENT` (transparent mode), a “1” bit in the image data MUST be drawn using the current fill color, and a “0” bit MUST NOT be drawn
`OPVP_IFORMAT_JPEG`	JPEG image format. Reserved for future extension.
`OPVP_IFORMAT_PNG`	PNG image format. Reserved for future extension.

OPVP_IFORMAT_RLE	RLE format. Reserved for future extension.

The image data format for each color space is defined in the section "Image Data Format".
A driver MUST support both OPVP_IFORMAT_RAW and OPVP_IFORMAT_MASK.

After drawing an image, the driver MUST NOT change the current point.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.7.3.opvpTransferDrawImage

Name

opvpTransferDrawImage – Transfers a chunk of bitmap image data

Synopsis

```
opvp_result_t opvpTransferDrawImage(
       opvp_dc_t printerContext,
       opvp_int_t count,
       const void *imageData);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

count – Number of image bytes to transfer.

imageData – Pointer to the source image data.

Description

This function transfers a chunk of bitmap image data. A caller MUST call the opvpStartDrawImage() function before calling this function.

When this function returns an error, the caller MUST call the opvpEndDrawImage() function before calling any other function.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.7.4.opvpEndDrawImage

Name

opvpEndDrawImage – Declares the end of a bitmap drawing sequence

Synopsis

```
opvp_result_t opvpEndDrawImage(
       opvp_dc_t printerContext);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

Description

This function declares the end of a bitmap drawing sequence.

If a caller does not transfer a sufficient image data by calling the opvpTransferDrawImage() function, this function MAY return an error. In this case, the driver MUST set the detailed error code to OPVP_BADREQUEST.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.8.Scan Line Operations

This section defines functions for scan line operations.

Scan line operations provide support for drawing of horizontal lines defined by one or more pairs of begin and end points. Scan lines are drawn with the drawing attributes registered with a Graphics State Object. Specifically, scan lines are drawn using the current fill color.

Driver support for any of the scan line operations is OPTIONAL.

If the caller calls any API entry other than `opvpScanline()` or `opvpAbortJob()` between the `opvpStartScanline()` and `opvpEndScanline()` functions, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

4.8.1.opvpStartScanline

Name

opvpStartScanline – Declares the start of a scan line draw operation.

Synopsis

```
opvp_result_t opvpStartScanline(
       opvp_dc_t printerContext,
       opvp_int_t yposition);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`yposition` – Vertical position to draw the scan line.

Description

This function declares the start of a scan line draw operation. Scan lines are drawn using three functions. The `opvpStartScanline()` function declares the start of a scan line draw operation, the `opvpScanline()` function transfers the actual scan line data, and the `opvpEndScanline()` function terminates the scan line draw operation. The driver MUST NOT change the current point after these operations.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.8.2.opvpScanline

Name

opvpScanline – Draws the scan lines.

Synopsis

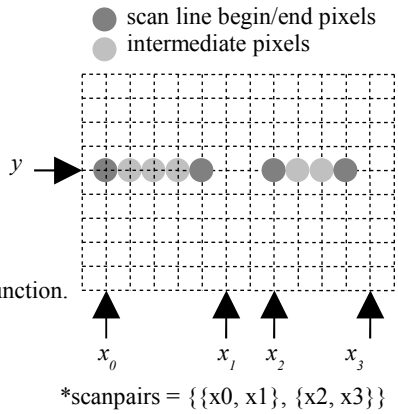
```
opvp_result_t opvpScanline(
       opvp_dc_t printerContext,
       opvp_int_t nscanpairs,
       const opvp_int_t *scanpairs);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`nscanpairs` – Number of begin and end point pairs

`scanpairs` – Pointer to an array of begin and end point pairs



*scanpairs = {{x0, x1}, {x2, x3}}

Description

This function draws the scan lines. After the caller calls this function, the driver MUST increment the y coordinate value of the current scan line by 1. The y coordinate of the current scan line is independent from y coordinate of the current point. The driver MUST maintain the current scan line's y coordinate in the printer context between calls to the `opvpStartScanLine()` and

opvpEndScanLine() functions. The driver MUST NOT change the current point when this function is called.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.8.3.opvpEndScanline

Name

opvpEndScanline – Terminates a scan line draw operation.

Synopsis

```
opvp_result_t opvpEndScanline(
        opvp_dc_t printerContext);
```

Arguments

printerContext – Printer context value returned by the opvpOpenPrinter() function.

Description

This function terminates a scan line draw operation.

Return Value

OPVP_OK or -1 in case of error. In the latter case, the driver MUST store the detailed error code in opvpErrorNo.

4.9.Raster Image Operations

This section defines functions for raster image operations.

Raster image operations provide the support for raster image drawing. Driver support for any of the raster image operations is OPTIONAL. However, in case the driver supports neither bitmap image nor path operations, raster image operations MUST be supported.

Compressed raster data is not supported by this specification. However, a driver MAY perform raster image data compression if desirable.

If the caller calls any API entry other than `opvpTransferRasterData()`, `opvpSkipRaster()` or `opvpAbortJob()` between the `opvpStartRaster()` and `opvpEndRaster()` functions, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

4.9.1.opvpStartRaster

Name

`opvpStartRaster` – Declares the start of a raster image drawing operation.

Synopsis

```
opvp_result_t opvpStartRaster(
       opvp_dc_t printerContext,
       opvp_int_t rasterWidth);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`rasterWidth` – Width (in pixels) of the raster image (MUST NOT include padding data)

Description

This function declares the start of a raster data drawing operation.

The driver MUST draw the raster image starting at the current point.

The driver MUST use the color space registered with the current Graphics State Object when interpreting the raster image data.

Raster data color spaces and their corresponding formats are defined in the section “Image Data Format”.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.9.2.opvpTransferRasterData

Name

`opvpTransferRasterData` – Transfers raster image data.

Synopsis

```
opvp_result_t opvpTransferRasterData(
       opvp_dc_t printerContext,
       opvp_int_t count,
       const opvp_byte_t *data);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`count` – Number of pixel data elements.

`data` – Pointer to pixel data array.

Description

This function transfers pixel data elements of a single row of raster data. The driver MUST only transfer the lesser of `count` and `rasterWidth` pixels. If `count` is less than `rasterWidth`, the driver MUST NOT transfer “filler” pixels. In case `count` exceeds `rasterWidth`, the driver MUST ignore the additional pixels.

After drawing the raster data, the driver MUST increment the value of the *y* coordinate of the current point by 1.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.9.3.opvpSkipRaster

Name

`opvpSkipRaster` – Skips lines during raster image drawing.

Synopsis

```
opvp_result_t opvpSkipRaster(
        opvp_dc_t printerContext,
        opvp_int_t count);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`count` – Number of lines to be skipped.

Description

This function skips `count` lines in the vertical direction during raster image drawing. After skipping these lines, the driver MUST increment the value of the *y* coordinate of the current point by `count`.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.9.4.opvpEndRaster

Name

`opvpEndRaster` – Terminates a raster image draw operation.

Synopsis

```
opvp_result_t opvpEndRaster(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function terminates a raster image draw operation.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.10.Stream Data Operations

This section defines functions for stream data operations.

Stream data operations provide support for applications that create printer native PDL(Printer Description Language) data and sends this data to the device directly. Driver support for any of the stream data operations is OPTIONAL

If the caller calls any API entries other than `opvpTransferStreamData()` or `opvpAbortJob()` between the `opvpStartStream()` and `opvpEndStream()` functions, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`.

4.10.1.opvpStartStream

Name

`opvpStartStream` – Declares the start of a stream data transfer.

Synopsis

```
opvp_result_t opvpStartStream(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function declares the start of a stream data transfer. Stream data is data that is sent to the printer device directly without any processing by the driver.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.10.2.opvpTransferStreamData

Name

`opvpTransferStreamData` – Sends stream data.

Synopsis

```
opvp_result_t opvpTransferStreamData(
        opvp_dc_t printerContext,
        opvp_int_t count,
        const void *data);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

`count` – Number of data bytes to transfer.

`data` – Pointer to the stream data.

Description

This function sends stream data directly to the printer device.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

4.10.3.opvpEndStream

Name

opvpEndStream – Declares the end of a stream data transfer.

Synopsis

```
opvp_result_t opvpEndStream(
        opvp_dc_t printerContext);
```

Arguments

`printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.

Description

This function terminates a stream data transfer.

Return Value

`OPVP_OK` or -1 in case of error. In the latter case, the driver MUST store the detailed error code in `opvpErrorNo`.

5.Macros, Types, Enumerations and Structures

5.1.Return Values

```
#define OPVP_OK           0      /* -1 for errors */
```

5.2.Error Codes

```
#define OPVP_FATALERROR   -1     /* error: cannot be recovered */
#define OPVP_BADREQUEST   -2     /* error: called where it should not be called */
#define OPVP_BADCONTEXT   -3     /* error: invalid printer context */
#define OPVP_NOTSUPPORTED -4     /* error: combination of parameters are set */
                                 /*   that cannot be handled by driver or printer */
#define OPVP_JOBCANCELED  -5     /* error: job has been canceled by some cause */
#define OPVP_PARAMERROR   -6     /* error: invalid parameter */
#define OPVP_VERSIONERROR -7     /* error: invalid API version */
```

5.3.Basic Types

```
typedef int opvp_dc_t;                  /* driver/device context */
typedef int opvp_result_t;              /* return value */
typedef unsigned char opvp_byte_t;      /* BYTE */
typedef unsigned char opvp_char_t;      /* character (string) */
typedef int opvp_int_t;                 /* integer */
typedef int opvp_fix_t;                 /* fixed integer */
typedef float opvp_float_t;             /* float */
typedef unsigned int opvp_flag_t;       /* flags */

typedef struct _opvp_point {
       opvp_fix_t x, y;
} opvp_point_t;

typedef struct _opvp_rectangle {
       opvp_point_t p0;                 /* start point */
       opvp_point_t p1;                 /* diagonal point */
} opvp_rectangle_t;

typedef struct _opvp_roundrectangle {
       opvp_point_t p0;                 /* start point */
       opvp_point_t p1;                 /* diagonal point */
       opvp_fix_t xellipse, yellipse;
} opvp_roundrectangle_t;
```

5.4.Image Formats

```
typedef enum _opvp_imageformat {
       OPVP_IFORMAT_RAW          = 0,
       OPVP_IFORMAT_MASK         = 1,
       OPVP_IFORMAT_RLE          = 2,
       OPVP_IFORMAT_JPEG         = 3,
       OPVP_IFORMAT_PNG          = 4
} opvp_imageformat_t;
```

5.5.Color Presentation

```
typedef enum _opvp_cspace {
       OPVP_CSPACE_BW            = 0,
       OPVP_CSPACE_DEVICEGRAY    = 1,
       OPVP_CSPACE_DEVICECMY     = 2,
       OPVP_CSPACE_DEVICECMYK    = 3,
       OPVP_CSPACE_DEVICERGB     = 4,
       OPVP_CSPACE_DEVICEKRGB    = 5,
       OPVP_CSPACE_STANDARDRGB   = 6,
```

```
        OPVP_CSPACE_STANDARDRGB64 = 7
} opvp_cspace_t;
```

5.6.Fill, Paint, Clip

```
typedef enum _opvp_fillmode {
        OPVP_FILLMODE_EVENODD           = 0,
        OPVP_FILLMODE_WINDING           = 1
} opvp_fillmode_t;

typedef enum _opvp_paintmode {
        OPVP_PAINTMODE_OPAQUE           = 0,
        OPVP_PAINTMODE_TRANSPARENT      = 1
} opvp_paintmode_t;

typedef enum _opvp_cliprule {
        OPVP_CLIPRULE_EVENODD           = 0,
        OPVP_CLIPRULE_WINDING           = 1
} opvp_cliprule_t;
```

5.7.Line

```
typedef enum _opvp_linestyle {
        OPVP_LINESTYLE_SOLID      = 0,
        OPVP_LINESTYLE_DASH       = 1
} opvp_linestyle_t;

typedef enum _opvp_linecap {
        OPVP_LINECAP_BUTT         = 0,
        OPVP_LINECAP_ROUND        = 1,
        OPVP_LINECAP_SQUARE       = 2
} opvp_linecap_t;

typedef enum _opvp_linejoin {
        OPVP_LINEJOIN_MITER       = 0,
        OPVP_LINEJOIN_ROUND       = 1,
        OPVP_LINEJOIN_BEVEL       = 2
} opvp_linejoin_t;
```

5.8.Brush

```
typedef struct _opvp_brushdata {
        opvp_bdtype_t type;
        opvp_int_t width, height, pitch;
        opvp_byte_t data[];

} opvp_brushdata_t;

typedef struct _opvp_brush {
        opvp_cspace_t colorSpace;
        opvp_int_t color[4];
        opvp_int_t xorg, yorg;    /* brush origin ,ignored for opvpSetBgColor   */
        opvp_brushdata_t *pbrush; /* pointer to brush data */
} opvp_brush_t;

typedef enum _opvp_bdtype {OPVP_BDTYPE_NORMAL = 0} opvp_bdtype_t;
```

5.9.Miscellaneous Flags

```
typedef enum _opvp_arcmode {
        OPVP_ARC                  = 0,
        OPVP_CHORD                = 1,
        OPVP_PIE                  = 2
} opvp_arcmode_t;

typedef enum _opvp_arcdir {
```

```
        OPVP_CLOCKWISE            = 0,
        OPVP_COUNTERCLOCKWISE     = 1
} opvp_arcdir_t;

typedef enum _opvp_pathmode {
        OPVP_PATHCLOSE            = 0,
        OPVP_PATHOPEN             = 1
} opvp_pathmode_t;
```

5.10.CTM

```
typedef struct _opvp_ctm {
        opvp_float_t a, b, c, d, e, f;
} opvp_ctm_t;
```

5.11.Device Information and Capabilities

```
typedef enum _opvp_queryinfoflags {
  OPVP_QF_DEVICERESOLUTION = 0x00000001,
  OPVP_QF_MEDIASIZE        = 0x00000002,
  OPVP_QF_PAGEROTATION     = 0x00000004,
  OPVP_QF_MEDIANUP         = 0x00000008,
  OPVP_QF_MEDIADUPLEX      = 0x00000010,
  OPVP_QF_MEDIASOURCE      = 0x00000020,
  OPVP_QF_MEDIADESTINATION = 0x00000040,
  OPVP_QF_MEDIATYPE        = 0x00000080,

  OPVP_QF_MEDIACOPY        = 0x00000100,/* Maximum copy number supported */
  OPVP_QF_PRINTREGION      = 0x00010000 /* only for opvpQueryDeviceInfo use */
} opvp_queryinfoflags_t;
```

5.12.API Procedures Structure

```
typedef      struct _opvp_api_procs {
        opvp_dc_t     (*opvpOpenPrinter)(opvp_int_t,const opvp_char_t*, const
opvp_int_t [2],struct _opvp_api_procs**);
        opvp_result_t (*opvpClosePrinter)(opvp_dc_t);
        opvp_result_t (*opvpStartJob)(opvp_dc_t,const opvp_char_t*);
        opvp_result_t (*opvpEndJob)(opvp_dc_t);
        opvp_result_t (*opvpAbortJob)(opvp_dc_t);
        opvp_result_t (*opvpStartDoc)(opvp_dc_t,const opvp_char_t*);
        opvp_result_t (*opvpEndDoc)(opvp_dc_t);
        opvp_result_t (*opvpStartPage)(opvp_dc_t,const opvp_char_t*);
        opvp_result_t (*opvpEndPage)(opvp_dc_t);
        opvp_result_t (*opvpQueryDeviceCapability)
(opvp_dc_t,opvp_flag_t,opvp_int_t*,opvp_char_t*);
        opvp_result_t (*opvpQueryDeviceInfo)
(opvp_dc_t,opvp_flag_t,opvp_int_t*,opvp_char_t*);
        opvp_result_t (*opvpResetCTM)(opvp_dc_t);
        opvp_result_t (*opvpSetCTM)(opvp_dc_t,const opvp_ctm_t*);
        opvp_result_t (*opvpGetCTM)(opvp_dc_t,opvp_ctm_t*);
        opvp_result_t (*opvpInitGS)(opvp_dc_t);
        opvp_result_t (*opvpSaveGS)(opvp_dc_t);
        opvp_result_t (*opvpRestoreGS)(opvp_dc_t);
        opvp_result_t (*opvpQueryColorSpace)(opvp_dc_t,opvp_int_t*,opvp_cspace_t*);
        opvp_result_t (*opvpSetColorSpace)(opvp_dc_t,opvp_cspace_t);
        opvp_result_t (*opvpGetColorSpace)(opvp_dc_t,opvp_cspace_t*);
        opvp_result_t (*opvpSetFillMode)(opvp_dc_t,opvp_fillmode_t);
        opvp_result_t (*opvpGetFillMode)(opvp_dc_t,opvp_fillmode_t*);
        opvp_result_t (*opvpSetAlphaConstant)(opvp_dc_t,opvp_float_t);
        opvp_result_t (*opvpGetAlphaConstant)(opvp_dc_t,opvp_float_t*);
        opvp_result_t (*opvpSetLineWidth)(opvp_dc_t,opvp_fix_t);
        opvp_result_t (*opvpGetLineWidth)(opvp_dc_t,opvp_fix_t*);
        opvp_result_t (*opvpSetLineDash)(opvp_dc_t,opvp_int_t,const opvp_fix_t*);
        opvp_result_t (*opvpGetLineDash)(opvp_dc_t,opvp_int_t*,opvp_fix_t*);
        opvp_result_t (*opvpSetLineDashOffset)(opvp_dc_t,opvp_fix_t);
        opvp_result_t (*opvpGetLineDashOffset)(opvp_dc_t,opvp_fix_t*);
```

```
        opvp_result_t (*opvpSetLineStyle)(opvp_dc_t,opvp_linestyle_t);
        opvp_result_t (*opvpGetLineStyle)(opvp_dc_t,opvp_linestyle_t*);
        opvp_result_t (*opvpSetLineCap)(opvp_dc_t,opvp_linecap_t);
        opvp_result_t (*opvpGetLineCap)(opvp_dc_t,opvp_linecap_t*);
        opvp_result_t (*opvpSetLineJoin)(opvp_dc_t,opvp_linejoin_t);
        opvp_result_t (*opvpGetLineJoin)(opvp_dc_t,opvp_linejoin_t*);
        opvp_result_t (*opvpSetMiterLimit)(opvp_dc_t,opvp_fix_t);
        opvp_result_t (*opvpGetMiterLimit)(opvp_dc_t,opvp_fix_t*);
        opvp_result_t (*opvpSetPaintMode)(opvp_dc_t,opvp_paintmode_t);
        opvp_result_t (*opvpGetPaintMode)(opvp_dc_t,opvp_paintmode_t*);
        opvp_result_t (*opvpSetStrokeColor)(opvp_dc_t,const opvp_brush_t*);
        opvp_result_t (*opvpSetFillColor)(opvp_dc_t,const opvp_brush_t*);
        opvp_result_t (*opvpSetBgColor)(opvp_dc_t,const opvp_brush_t*);
        opvp_result_t (*opvpNewPath)(opvp_dc_t);
        opvp_result_t (*opvpEndPath)(opvp_dc_t);
        opvp_result_t (*opvpStrokePath)(opvp_dc_t);
        opvp_result_t (*opvpFillPath)(opvp_dc_t);
        opvp_result_t (*opvpStrokeFillPath)(opvp_dc_t);
        opvp_result_t (*opvpSetClipPath)(opvp_dc_t,opvp_cliprule_t);
        opvp_result_t (*opvpResetClipPath)(opvp_dc_t);
        opvp_result_t (*opvpSetCurrentPoint)(opvp_dc_t,opvp_fix_t,opvp_fix_t);
        opvp_result_t (*opvpLinePath)(opvp_dc_t,opvp_pathmode_t,opvp_int_t,const
opvp_point_t*);
        opvp_result_t (*opvpPolygonPath)(opvp_dc_t,opvp_int_t,const opvp_int_t*,const
opvp_point_t*);
        opvp_result_t (*opvpRectanglePath)(opvp_dc_t,opvp_int_t,const
opvp_rectangle_t*);
        opvp_result_t (*opvpRoundRectanglePath)(opvp_dc_t,opvp_int_t,const
opvp_roundrectangle_t*);
        opvp_result_t (*opvpBezierPath)(opvp_dc_t,opvp_int_t,const opvp_point_t*);
        opvp_result_t (*opvpArcPath)
(opvp_dc_t,opvp_arcmode_t,opvp_arcdir_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,o
pvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t);
        opvp_result_t (*opvpDrawImage)
(opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imageformat_t,opvp_int_t,opvp_int_t,
const void*);
        opvp_result_t (*opvpStartDrawImage)
(opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imageformat_t,opvp_int_t,opvp_int_t)
;
        opvp_result_t (*opvpTransferDrawImage)(opvp_dc_t,opvp_int_t,const void*);
        opvp_result_t (*opvpEndDrawImage)(opvp_dc_t);
        opvp_result_t (*opvpStartScanline)(opvp_dc_t,opvp_int_t);
        opvp_result_t (*opvpScanline)(opvp_dc_t,opvp_int_t,const opvp_int_t*);
        opvp_result_t (*opvpEndScanline)(opvp_dc_t);
        opvp_result_t (*opvpStartRaster)(opvp_dc_t,opvp_int_t);
        opvp_result_t (*opvpTransferRasterData)(opvp_dc_t,opvp_int_t,const
opvp_byte_t*);
        opvp_result_t (*opvpSkipRaster)(opvp_dc_t,opvp_int_t);
        opvp_result_t (*opvpEndRaster)(opvp_dc_t);
        opvp_result_t (*opvpStartStream)(opvp_dc_t);
        opvp_result_t (*opvpTransferStreamData)(opvp_dc_t,opvp_int_t,const void*);
        opvp_result_t (*opvpEndStream)(opvp_dc_t);
} opvp_api_procs_t;
```

6.Authors and Contributors

6.1.Editors

Osamu Mihara – Fuji Xerox Co., Ltd.

6.2.Authors

Osamu Mihara – Fuji Xerox Co., Ltd.
Yasumasa Toratani – Canon Inc.

6.3.Contributors

(Alphabetical order) Hidekazu Hagiwara (MintWave), Masaki Iwata (AXE), Hidenori Kanjo (BBR), Shimpei Kitayama (EPSON AVASYS), Kenichi Maeda (E&D), Akeo Maruyama (Ricoh), Olaf Meeuwissen (EPSON AVASYS), Hisao Nakamura (E&D), Naruhiko Ogasawara (Ricoh), Koji Otani (BBR), Kenji Wakabayashi (MintWave), Toshihiro Yamagishi (Turbolinux), Akira Yoshiyama (NEC)

7.History

Version 0.1 (Japanese) Mihara/Toratani

Version 0.2 (Japanese) Mihara

openprinting-0.1.1 implementation by opfc

Version 0.2-en 2004-3-14 Mihara/Toratani/Kitayama/Yamagishi/Kanjo

Translation to English
Feedback from opfc implementation

Version 1.0 RC1 2005-7-20 Mihara

Change copyright notice from FDL to MIT style copyright
Delete temporary font operations.

Version 1.0 RC2 2006-6-9/2006-6-17/2006-6-20/2006-6-26/2006-6-29

Add API version number to OpenPrinter() parameter
Add pitch and delete count to/from DrawImage()/StartDrawImage()
Add a figure for SetMiterLimit.
Change function names/constants/enumerations/structures names to add FSG prefixes.
Change copyright year and dates
Delete "Printer Driver Database"
Delete *cmap* from CSPASE chart.
Fix parameter colorSpace (wrong) to colorDepth (correct) for DrawImage()/StartDrawImage
Change description for fsgpdStartJob/fsgpdEndJob/fsgpdAbortJob
Add type FSGPD_CHAR for character type

Version 1.0 RC2 2006-10-20

Add FSGPD_CSPACE_DEVICEKRGB
Remove "current implementation" descriptions from Color Scheme and Color Space descriptions.
Remove "described later" descriptions
Make support of updf for scheme "MUST."

Version 1.0 RC2 2006-12-05

Format: Add line numbers and chapter numbers.
Format: Replace Microsoft Expressions object to OOo Expressions

Version 1.0 RC3 2007-5-19 Toratani

Append “opvp” or “OPVP” to function names, enumerations, symbols according to the naming rules.
Update parameters of `opvpOpenPrinter()` function and the bitmap image functions.
Rearrange the document sections, chapters and text format.
Merge the RC2 2006-6-9 – 2006-12-05 updates made by Mihara

Version 1.0 RC5 2007-9-19 Toratani – Overall parts are reviewed and revised.

Brushed-up massively English expressions and corrected typos.
Re-write “Introduction.”
Added sections “3.5” and “3.6”
Added “const” type keywords to function parameters which should not be over-written.
Added “num” parameter to opvpSetLineDash().
Added “pnum” parameter to opvpGetLineDash().
Deleted “Graphic Operation Fallback” section.

Version 1.0 RC6 2009-12-24 Mihara

Changed years for Copyright from “2007” to “2004-2009.”