

1
2
3
4
5
6
7
8

OpenPrinting
Vector Printer Driver
Application Program Interface
(OPVP)
Specification
Version-1.0 RC3
(2007-5-20)

9

1 **OpenPrinting Vector Printer Driver Application Interface Specification**

2

3 Copyright © 2007 OpenPrinting Work Group

4 Permission is hereby granted, free of charge, to any person obtaining a copy of this documentation files, to deal in the
5 documentation without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute,
6 sublicense, and/or sell copies of the documentation, and to permit persons to whom the Software is furnished to do so, subject
7 to the following conditions:

8 The above copyright notice and this permission notice shall be included in all copies or substantial portions of the
9 documentation.

10 THE DOCUMENTATION IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
11 INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
12 PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
13 LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT
14 OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE DOCUMENTATION OR THE USE OR
15 OTHER DEALINGS IN THE DOCUMENTATION.

16 UNIX is a registered trademark of the Open Group in the United States and other countries.

17 Linux is a trademark of Linus Torvalds.

18 The X Window System is a trademark of X Consortium, Inc.

19 OpenGL is a registered trademark of Silicon Graphics, Inc.

20 PostScript is a registered trademark of Adobe Systems Inc.

Table of Contents

1. Notation and Terminology.....	6
1.1. Notation Conventions.....	6
1.2. Conformance Terminology.....	6
2. Introduction.....	7
2.1. Driver and Caller.....	7
2.2. Loading and Calling Printer Driver.....	7
2.3. Printer Driver Database.....	7
2.4. Notes about Function Parameters.....	7
3. Graphic Model.....	8
3.1. Coordinate System.....	8
3.2. Objects.....	8
3.3. Graphics State Object.....	9
3.4. CTM.....	9
3.5. Color.....	9
3.6. Scan Rule.....	10
4. Operations.....	11
4.1. Creating and Managing Print Contexts.....	11
4.1.1. opvpOpenPrinter.....	11
4.1.2. opvpClosePrinter.....	11
4.2. Job, Document and Page Operations.....	13
4.2.1. opvpStartJob.....	13
4.2.2. opvpEndJob.....	13
4.2.3. opvpAbortJob.....	14
4.2.4. opvpStartDoc.....	14
4.2.5. opvpEndDoc.....	15
4.2.6. opvpStartPage.....	15
4.2.7. opvpEndPage.....	15
4.3. Query Operations.....	17
4.3.1. opvpQueryDeviceCapability.....	17
4.3.2. opvpQueryDeviceInfo.....	17
4.4. Attribute of Job, Document and Page Operations.....	19
4.5. Graphics State Object Operations.....	21
4.5.1. opvpResetCTM.....	21
4.5.2. opvpSetCTM.....	21
4.5.3. opvpGetCTM.....	22
4.5.4. opvpInitGS.....	22
4.5.5. opvpSaveGS.....	23
4.5.6. opvpRestoreGS.....	23
4.5.7. opvpQueryColorSpace.....	23
4.5.8. opvpSetColorSpace.....	24
4.5.9. opvpGetColorSpace.....	24
4.5.10. opvpQueryROP.....	25
4.5.11. opvpSetROP.....	25
4.5.12. opvpGetROP.....	26
4.5.13. opvpSetFillMode.....	26
4.5.14. opvpGetFillMode.....	26
4.5.15. opvpSetAlphaConstant.....	27
4.5.16. opvpGetAlphaConstant.....	27
4.5.17. opvpSetLineWidth.....	27

4.5.18.opvpGetLineWidth.....	28
4.5.19.opvpSetLineDash.....	28
4.5.20.opvpGetLineDash.....	29
4.5.21.opvpSetLineDashOffset.....	29
4.5.22.opvpGetLineDashOffset.....	30
4.5.23.opvpSetLineStyle.....	30
4.5.24.opvpGetLineStyle.....	31
4.5.25.opvpSetLineCap.....	31
4.5.26.opvpGetLineCap.....	32
4.5.27.opvpSetLineJoin.....	32
4.5.28.opvpGetLineJoin.....	33
4.5.29.opvpSetMiterLimit.....	33
4.5.30.opvpGetMiterLimit.....	33
4.5.31.opvpSetPaintMode.....	34
4.5.32.opvpGetPaintMode.....	34
4.5.33.opvpSetStrokeColor.....	35
4.5.34.opvpSetFillColor.....	35
4.5.35.opvpSetBgColor.....	36
4.6.Path Operations.....	37
4.6.1.opvpNewPath.....	37
4.6.2.opvpEndPath.....	37
4.6.3.opvpStrokePath.....	37
4.6.4.opvpFillPath.....	38
4.6.5.opvpStrokeFillPath.....	38
4.6.6.opvpSetClipPath.....	39
4.6.7.opvpResetClipPath.....	39
4.6.8.opvpSetCurrentPoint.....	39
4.6.9.opvpLinePath.....	40
4.6.10.opvpPolygonPath.....	40
4.6.11.opvpRectanglePath.....	41
4.6.12.opvpRoundRectanglePath.....	41
4.6.13.opvpBezierPath.....	42
4.6.14.opvpArcPath.....	42
4.7.Bitmap Image Operations.....	44
4.7.1.opvpDrawImage.....	44
4.7.2.opvpStartDrawImage.....	45
4.7.3.opvpTransferDrawImage.....	46
4.7.4.opvpEndDrawImage.....	46
4.8.Scan Line Operations.....	47
4.8.1.opvpStartScanline.....	47
4.8.2.opvpScanline.....	47
4.8.3.opvpEndScanline.....	48
4.9.Raster Image Operations.....	49
4.9.1.opvpStartRaster.....	49
4.9.2.opvpTransferRasterData.....	49
4.9.3.opvpSkipRaster.....	50
4.9.4.opvpEndRaster.....	50
4.10.Stream Data Operations.....	51
4.10.1.opvpStartStream.....	51
4.10.2.opvpTransferStreamData.....	51
4.10.3.opvpEndStream.....	51

5.Graphic Operation Fallback.....	53
6.Macros, Types, Enums and Structures.....	54
6.1.Return Values.....	54
6.2.Error Codes.....	54
6.3.Basic Types.....	54
6.4.Image Formats.....	54
6.5.Color Presentation.....	54
6.6.Fill, Paint, Clip.....	55
6.7.Line.....	55
6.8.Brush.....	55
6.9.Misc. Flags.....	55
6.10.CTM.....	56
6.11.Device Information and Capabilities.....	56
6.12.Sample Header File.....	56
7.Authors and Contributors.....	61
7.1.Editors.....	61
7.2.Authors.....	61
7.3.Contributers.....	61
8.History.....	62

1. Notation and Terminology

1.1. Notation Conventions

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

Fond and Style	Description	Examples
Courier	Definition of functions, structures, enums and constants.	<pre>opvp_result_t opvpClosePrinter(opvp_dc_t printerContext); typedef struct _opvp_point { opvp_fix_t x, y; } opvp_point_t; #define OPVP_OK 0</pre>
	Function parameters	<code>printerContext</code>
	Source code examples.	<pre>#ifndef _OPVP_H_ #define _OPVP_H_</pre>
<i>Italic</i>	Coordinate values (x, y)	<i>(x0, y0)</i>

1.2. Conformance Terminology

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].

2.1.Driver and Caller

This specification defines Application Program Interface (API) which is used in printing environment offered by OpenPrinting. In this document, the word “printer driver” or “driver” refers a software library. Driver translates document data generated by application programs into printer command data stream which is consumed by a printer.

This specification aims to make an abstraction interface for graphics drawing functions which are supported by printer languages and offers them as API to make available to use them without knowledges of each printer model drawing functions and languages.

This specification covers ink jet printers which handle only raster data as well as high-end laser printers which have high-level graphics functions. Especially, to improve printing performance of high-end laser printers, this specification covers generic API for high level-graphics drawing functions. This specification covers black and white as well as full color printers.

The word “caller” is a program which calls a driver via the API defined in this specification. For example, open source renderers Ghostscript, Xpdf, and X Print Server can be a caller if it calls a driver via the API. However, the API are independent of particular renderers, printer driver based on this specification does not depend on any specific type of renderers.

2.2.Loading and Calling Printer Driver

This document specifies several APIs for printer drivers that are provided as static or dynamic libraries. However linking method to the library is prepared by each operating system, therefore, this document does not specify the driver linking method.

Printer driver which is loaded and linked to caller has the same memory space of the caller. On the other hand, printer driver which is loaded into a separated memory space from caller and executed as a different server process from the caller, the caller communicates with the driver via RPC call. In this case, the server process may link the printer driver library and call it via APIs specified in this document, and the caller does not link the driver, but may communicate with the server process via RPC call. This document does not specify the RPC specification between the caller and the server process, and RPC specification should be defined in another document.

Printing data stream which is generated by printer driver during each drawing API calls is written into the file descriptor given by `opvpOpenPrinter()` function call. Printer driver does not need to generate each printing data stream during each API call, rather than driver may generate printing data stream during particular API call, or may generate whole printing data stream at `opvpEndPage()` function call. Caller MUST receive whole the printing data stream properly from driver at any time between `opvpOpenPrinter()` and `opvpClosePrinter()` function calls.

2.3.Printer Driver Database

Printer driver name and its model name are managed by driver data base. The driver data base should be supplied in UPDF and/or PPD format.

2.4.Notes about Function Parameters

Data area allocated by caller and given by a pointer argument for each API is possibly referred by printer driver at any time during the process of each page. Therefore, caller MUST keep the data area during each page processing and release the data area after `opvpEndPage()` function call.

1

3.1. Coordinate System

1

2 The origin of the coordinate system which is defined and used in this
3 document is physical upper left corner of the media handled by each
4 device. The unit of the coordinate system is based on each device resolution.
5 Positive direction on x axis is from the origin to the right direction on the
6 media, and positive direction on y axis is from the origin to the lower
7 direction on the media.

8 Type of the coordinate value is signed fixed point 32 bits value, where
9 integer takes 24 bits and decimal takes 8 bits (type name is `opvp_fix_t`).

10 Coordinate value x and y define each horizontal and vertical distance from
11 the origin to the point on the media. The point which has the integer
12 coordinate value x and y is on the intersection of the coordinate system grid,
13 and physical device pixels are assumed to be printed on the intersections of
14 the grid (Grid Intersection Model). The relation between coordinate grids
15 and device pixels are shown in the right down figure.

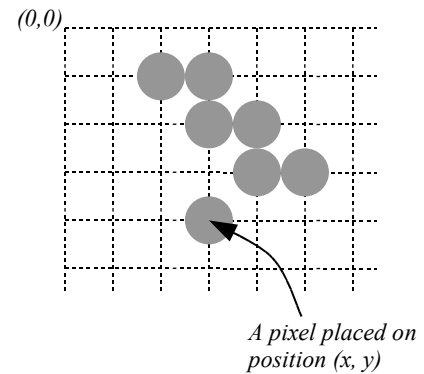
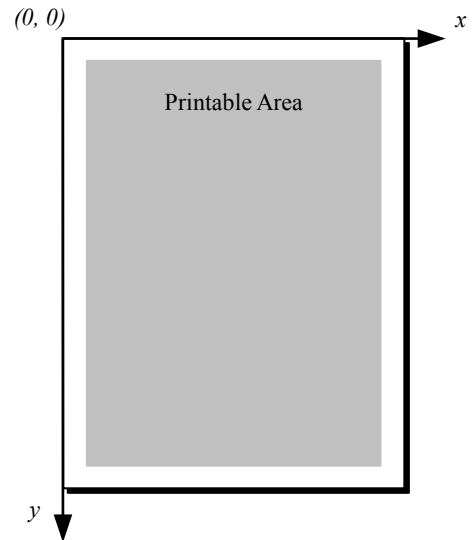
3.2. Objects

1

2 Following kinds of graphics objects are handled in this document.

- 3 • Path
- 4 • Bitmap Image
- 5 • Scanline
- 6 • Raster Image

7



3.3. Graphics State Object

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

Graphics State Object MUST keep the following properties and drawing attributes. For each properties and drawing attributes, refer the description of related functions for more detail.

Properties	Related functions
CTM	<code>opvpResetCTM()</code> , <code>opvpSetCTM()</code> , <code>opvpGetCTM()</code>
Path	<code>opvpNewPath()</code> , <code>opvpEndPath()</code> , <code>opvpStrokePath()</code> , <code>opvpFillPath()</code> , <code>opvpStrokeFillPath()</code> , <code>opvpSetClipPath()</code> , <code>opvpResetClipPath()</code> , <code>opvpSetCurrentPoint()</code> , <code>opvpLinePath()</code> , <code>opvpPolygonPath()</code> , <code>opvpRectanglePath()</code> , <code>opvpRoundRectanglePath()</code> , <code>opvpBezierPath()</code> , <code>opvpArcPath()</code>
Clipping region	<code>opvpSetClipPath()</code> , <code>opvpResetClipPath()</code>

Drawing Attributes	Related functions
ColorSpace	<code>opvpQueryColorSpace()</code> , <code>opvpSetColorSpace()</code> , <code>opvpGetColorSpace</code>
Raster operation code	<code>opvpQueryROP()</code> , <code>opvpSetROP()</code> , <code>opvpGetROP()</code> ★削除?
Filling mode	<code>opvpSetFillMode()</code> , <code>opvpGetFillMode()</code>
Alpha blending constant	<code>opvpSetAlphaConstant()</code> , <code>opvpGetAlphaConstant()</code>
Stroke line width	<code>opvpSetLineWidth()</code> , <code>opvpGetLineWidth()</code>
Line dash pattern	<code>opvpSetLineDash()</code> , <code>opvpGetLineDash()</code>
Line dash pattern offset	<code>opvpSetLineDashOffset()</code> , <code>opvpGetLineDashOffset()</code>
Line style	<code>opvpSetLineStyle()</code> , <code>opvpGetLineStyle()</code>
Line cap style	<code>opvpSetLineCap()</code> , <code>opvpGetLineCap()</code>
Line join style	<code>opvpSetLineJoin()</code> , <code>opvpGetLineJoin()</code>
Miter limit value	<code>opvpSetMiterLimit()</code> , <code>opvpGetMiterLimit()</code>
Painting mode	<code>opvpSetPaintMode()</code> , <code>opvpGetPaintMode()</code>
Stroke line color	<code>opvpSetStrokeColor()</code>
Fill color	<code>opvpSetFillColor()</code>
Background color	<code>opvpSetBgColor()</code>

3.4. CTM

Printer driver MUST maintain Coordinate Transformation Matrix (CTM) in each Graphics State Object. CTM is used for transformation from caller (renderer) coordinate system to printer (device) coordinate system. CTM is presented as following matrix form.

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

To set or get CTM from the Graphics State Object, refer the description of `opvpResetCTM()`, `opvpSetCTM()`, `opvpGetCTM()` functions.

3.5. Color

Following color spaces are defined in this document.

Color Space Macro	Color Space
<code>OPVP_CSPACE_BW</code>	Black and White
<code>OPVP_CSPACE_DEVICEGRAY</code>	Grayscale
<code>OPVP_CSPACE_DEVICECMY</code>	CMY
<code>OPVP_CSPACE_DEVICECMYK</code>	CMY and Black
<code>OPVP_CSPACE_DEVICEKRGB</code>	Device RGB
<code>OPVP_CSPACE_DEVICEKRGB</code>	Device KRGB
<code>OPVP_CSPACE_STANDARDRGB</code>	sRGB
<code>OPVP_CSPACE_STANDARDRGB64</code>	scRGB

1
2 Only OPVP_CSPACE_BW, OPVP_CSPACE_DEVICEGRAY and OPVP_CSPACE_STANDARDRGB should be supported by
3 printer driver which is based on this document. Other color spaces are reserved for future use.

1 **3.6.Scan Rule**

2 Pixel drawing on two or more neighbor regions should be done without any contradiction. For example of the painting method,
3 “right-bottom exclusive method” can be used in a driver, however, the painting method implementation of each printer driver
4 depends on each page description language of each printer. Therefore, this document does not specify the painting method
5 implementation.

4.1.Creating and Managing Print Contexts

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

4.1.1.opvpOpenPrinter

Name

opvpOpenPrinter – Create printer context.

Synopsis

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

Arguments

outputFD – File descriptor to write printing data stream.

printerModel – Printer Model Name in UTF-8.

apiVersion – Vector Printer Driver Specification Version number which the driver supports. ApiVersion[0] is the major and apiVersion[1] is the minor value of the version respectively.

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

Description

This function initializes driver. Caller MUST specify the file descriptor for writing the printing data stream, and the driver generates the printing data stream and MUST write it into the file descriptor. The caller may specify the printer model name by printerModel in UFT-8. If caller sets NULL in printerModel, driver SHOULD use the default printer model of the driver. Printer model name should be defined in the printer model data base.

Driver may write its debug messages into stderr. Therefor stderr MUST not be given for outputFD by caller.

Driver MUST allocate the struct _opvp_api_procs buffer and store each API entry address of the driver into each corresponding member of apiEntry. If driver does not prepare some APIs, the driver MUST store NULL into the corresponding apiEntry members.

Only this function MUST be exported by driver library to caller which links the driver library. Caller MUST refer and call each API by the addresses stored in apiEntry.

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. Caller sets the printer context to the first parameter to call other APIs.

Return Value

Printer context value (positive value) or -1 when error , and driver MUST store the detailed error code in opvpErrorNo.

4.1.2.opvpClosePrinter

Name

opvpClosePrinter – Delete printer context.

Synopsis

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

Arguments

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

1 **Description**

2 This function terminates the printing process and deletes the printer context kept in the printer driver. Caller MUST close the
3 file descriptor that the caller gave as the `outputFD` for the `opvpOpenPrinter()` function.

4 Caller should call the `opvpEndJob()` function before calling this function to declare the end of printing job. If caller calls this
5 function before a printing job is not completed, in other words, if caller calls this function before calling the `opvpEndJob()`
6 function, driver should discard all the printing data stream for the printing job, however driver may not guarantee to cancel the
7 printing job normally.

8 **Return Value**

9 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

1 4.2. Job, Document and Page Operations

2 Functions to operate job controls. These functions MUST be supported by all drivers. One printing job consists of one or more
3 documents. One document consists of one or more pages. Caller MUST call the `opvpStartJob()` to declare to start a
4 printing job, and call the `opvpEndJob()` function to declare the end of the printing job for each job, and MUST call the
5 `opvpStartDoc()` and `opvpEndDoc()` functions for starting and ending of every documents, and also MUST call the
6 `opvpStartPage()` and `opvpEndPage()` functions for starting and ending of every pages. However, if a printing job
7 consists of one document, caller can omit calling the `opvpStartDoc()` and `opvpEndDoc()` functions.

8 4.2.1. `opvpStartJob`

9 **Name**

10 `opvpStartJob` – Declare to start a printing job.

11 **Synopsis**

```
12 opvp_result_t opvpStartJob(  
13     opvp_dc_t printerContext,  
14     opvp_char_t *jobInfo);
```

15 **Arguments**

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

17 `jobInfo` – Printing job property string.

18

19 **Description**

20 This function declares to start a printing job.

21 Caller MUST call this function before calling the `opvpStartDoc()` or `opvpStartPage()` functions or other drawing
22 APIs.

23 Caller can set printing job properties into `jobInfo`. Printer driver MUST keep the given printing job properties until the
24 `opvpEndJob()` function is called. When caller calls the `opvpStartJob()` function again after calling the
25 `opvpEndJob()` function, driver MUST override the former `jobInfo` by the new `jobInfo` given by the latest
26 `opvpStartJob()` function call.

27 If caller sets `NULL` to `jobInfo`, printer driver SHOULD use its default printing job properties. Data format of `jobInfo` is
28 described in the section "Attribute of Job, Document and Page Operations" in this document.

29 It depends on printer and driver capabilities whether nesting printing job (`opvpStartJob()` and `opvpEndJob()` functions
30 can be called between the `opvpStartJob()` and `opvpEndJob()` functions are called) can be handled. If a printer or
31 printer driver does not allow nesting printing jobs, the driver MUST return error and set the error code `OPVP_BADREQUEST` in
32 `opvpErrorNo`.

33 **Return Value**

34 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

35 4.2.2. `opvpEndJob`

36 **Name**

37 `opvpEndJob` – Declare to terminate a printing job.

38 **Synopsis**

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

41 **Arguments**

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

43 **Description**

44 This function declares to terminate the printing job.

45 Caller MUST call this function after finishing each printing job processing.

1 **Return Value**
2 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in `opvpErrorNo`.

3 4.2.3.opvpAbortJob

4 **4.2.3.1.Name**
5 `fsgpdAbortJob` – Declare to abort the printing job.

6 **Synopsis**
7 `opvp_result_t fsgpdAbortJob(
8 opvp_dc_t printerContext);`

9 **Arguments**
10 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.

11 **Description**
12 This function cleans up the printing operations to abort the print job. Driver MAY create a print data to clean up the printer
13 printing status and MAY send the data to the printer. However, because whether driver sends the printing job data to the printer
14 before sending the aborting data depends on the timing when caller calls this function, so calling this function does not
15 guarantee that no papers are wasted by the printer. However, after caller calls this function, driver and printer MUST become in
16 the initial state and next printing job MUST be accepted normally. ★文章を変更、これでよいか？
17 ★EndJobは呼ぶ必要は無いのか？明記する必要あり

18 **Return Value**
19 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in `opvpErrorNo`.

20 4.2.4.opvpStartDoc

21 **Name**
22 `opvpStartDoc` – Declare to start a printing document.

23 **Synopsis**
24 `opvp_result_t opvpStartDoc(
25 opvp_dc_t printerContext,
26 opvp_char_t *docInfo);`

27 **Arguments**
28 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
29 `docInfo` – Printing document property string.

30 **Description**
31 This function declares to start a printing document.
32 Caller should call this function after calling the `opvpStartJob()` function, and before calling the `opvpStartPage()`
33 function and other drawing APIs. However, if a printing job consists of one document, caller can omit calling the
34 `opvpStartDoc()` and `opvpEndDoc()` functions.
35 Caller can set document properties into `jobInfo`. Printer driver MUST keep the given document properties until the
36 `opvpEndDoc()` function is called. When caller calls the `opvpStartDoc()` function again after calling the
37 `opvpEndDoc()` function, driver MUST override the former `docInfo` by the new `docInfo` given by the latest
38 `opvpStartDoc()` function call.
39 If caller sets NULL to `docInfo`, printer driver SHOULD use its default printing document properties. Data format of
40 `docInfo` is described in the section "Attribute of Job, Document and Page Operations" in this document.
41 It depends on printer and driver capabilities whether nesting document (`opvpStartDoc()` and `opvpEndDoc()` functions
42 can be called between the `opvpStartDoc()` and `opvpEndDoc()` functions are called) can be handled. If a printer or
43 printer driver does not allow nesting document, the driver MUST return error and set the error code `OPVP_BADREQUEST` in
44 `opvpErrorNo`.

45

1 **Return Value**
2 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

3 4.2.5.opvpEndDoc

4 **Name**
5 opvpEndDoc – Declare to terminate the printing document

6 **Synopsis**
7 opvp_result_t opvpEndDoc(
8 opvp_dc_t printerContext);

9 **Arguments**
10 printerContext – Printer context value returned by opvpOpenPrinter () function.

11 **Description**
12 This function declares to terminate the printing document.
13 If caller called the opvpStartDoc () function, the caller MUST call this function after finishing each document processing.
14 If a printing job consists of one document, caller can omit calling the opvpStartDoc () and opvpEndDoc () functions.

15 **Return Value**
16 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

17 4.2.6.opvpStartPage

18 **Name**
19 opvpStartPage – Declare to start a printing page.

20 **Synopsis**
21 opvp_result_t opvpStartPage(
22 opvp_dc_t printerContext,
23 opvp_char_t *pageInfo);

24 **Arguments**
25 printerContext – Printer context value returned by opvpOpenPrinter () function.
26 pageInfo – Printing page property string.

27 **Description**
28 This function declares to start a printing page.
29 Caller MUST call this function after calling the opvpStartJob () and opvpStartDoc () functions, and before calling
30 other drawing APIs.
31 Caller can set page properties into pageInfo. Driver MUST keep the page properties until the opvpEndPage () is called.
32 When the caller calls the opvpStartPage () function again after calling the opvpEndPage (), the driver MUST override
33 the former pageInfo by the new pageInfo given by the latest opvpStartPage () function call.
34 If caller sets NULL to pageInfo, printer driver SHOULD use its default page properties. Data format of pageInfo is
35 described in the section "Attribute of Job, Document and Page Operations" in this document.

36 **Return Value**
37 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

38 4.2.7.opvpEndPage

39 **Name**
40 opvpEndPage – Declare to terminate the printing page.

1 **Synopsis**

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

4 **Arguments**

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

6 **Description**

7 This function declares to terminate the printing page.

8 Caller MUST call this function after finishing each page processing.

9 **Return Value**

10 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

11

1 4.3. Query Operations

2 Functions to query device capabilities and information. These functions are OPTIONAL.

3 4.3.1. opvpQueryDeviceCapability

4 **Name**

5 opvpQueryDeviceCapability – Query device capabilities.

6 **Synopsis**

```
7 opvp_result_t opvpQueryDeviceCapability(  
8     opvp_dc_t printerContext,  
9     opvp_queryinfoflags_t queryflag, ✘  
10    opvp_int_t *buflen, ✘  
11    opvp_char_t *infoBuf); ✘
```

12 **Arguments**

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

14 queryflag – Flag which specifies the device capability to query.

15 buflen – Number of bytes of the buffer specified by *infoBuf.

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

17 **Description**

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

19

20 Caller MUST set one or more bit flags into queryflag to query capabilities. Following enums should be supported by driver;

```
21 typedef enum _opvp_queryinfoflags {  
22     OPVP_QF_DEVICERESOLUTION      = 0x00000001,  
23     OPVP_QF_MEDIASIZE              = 0x00000002,  
24     OPVP_QF_PAGEROTATION          = 0x00000004,  
25     OPVP_QF_MEDIANUP              = 0x00000008,  
26     OPVP_QF_MEDIADUPLEX           = 0x00000010,  
27     OPVP_QF_MEDIASOURCE           = 0x00000020,  
28     OPVP_QF_MEDIADESTINATION      = 0x00000040,  
29     OPVP_QF_MEDIATYPE              = 0x00000080,  
30     OPVP_QF_MEDIACOPY              = 0x0000100, /* Maximum copy number supported */  
31     OPVP_QF_PRINTREGION            = 0x00010000 /* only for opvpQueryDeviceInfo use */  
32 } opvp_queryinfoflags_t;  
33
```

34 Driver MUST return queried capabilities into the infoBuf buffer in ASCII text format, and also return the number of bytes of
35 the capabilities text into *buflen. If the buffer does not have enough length to store all the capabilities queried, driver MUST
36 return the necessary number of bytes to retrieve all the capabilities in *buflen, and return error and set the error code
37 OPVP_PARAMERROR into opvpErrorNo. If caller sets NULL in infoBuf, driver MUST return the number of the bytes of
38 all the capabilities in *buflen.

39 The name and value pair format of each capability stored in the infoBuf buffer has the same format of jobInfo which is
40 used for the opvpStartJob() function. For example, if the device resolution is queried, resolution list is stored in the
41 following multiple name and value pairs format. The first name and value pair in the list is the default capability.

42 updf:DeviceResolution=deviceResolution_600x600,deviceResolution_1200x1200

43 **Return Value**

44 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

45 4.3.2. opvpQueryDeviceInfo

46 **Name**

47 opvpQueryDeviceInfo – Query device information.

1 **Synopsis**
 2 `opvp_result_t opvpQueryDeviceInfo(
 3 opvp_dc_t printerContext,
 4 opvp_queryinfoflags_t queryflag,※
 5 opvp_int_t *buflen,※
 6 opvp_char_t *infoBuf);※`

7 **Arguments**
 8 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
 9 `queryflag` – Flag which specifies the device information to query.
 10 `buflen` – Number of bytes of the buffer specified by `*infoBuf`.
 11 `infoBuf` – Pointer to the buffer to store device information.

12 **Description**
 13 This function queries the current setting of printer or driver information.

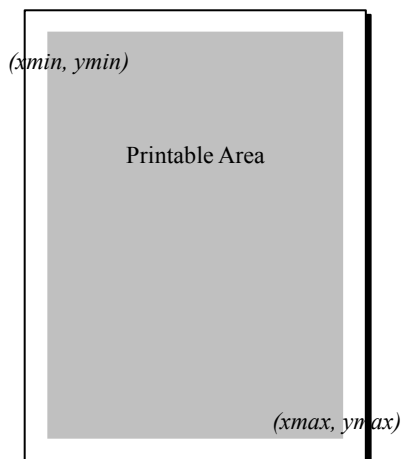
14
 15 Caller MUST set one or more bit flags in `queryflag` to query information. Same enums which are used for
 16 `opvpQueryDeviceCapability()` function MUST be used.

17
 18 Driver MUST return queried information into the `infoBuf` buffer in ASCII text format, and also return the number of bytes of
 19 the information text into `*buflen`. If the buffer does not have enough length to store all the information queried, driver
 20 MUST return the necessary number of bytes to retrieve all the information in `*buflen`, and return error and set the error code
 21 `OPVP_PARAMERROR` into `opvpErrorNo`. If caller sets NULL in `infoBuf`, driver MUST return the number of the bytes of
 22 all the information in `*buflen`.

23 The name and value pair format of each information stored in the `infoBuf` buffer has the same format of `jobInfo` which is
 24 used for the `opvpStartJob()` function.

25 When caller sets the `OPVP_QF_PRINTREGION` into `queryflag`, driver MUST respond the printable area for its query in
 26 the current resolution setting. The printable area values format MUST be as following. Each value represents the left top and
 27 right bottom corner of the current printable area setting as shown in the figure. These values depend on the current media
 28 orientation setting.

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



41 **Return Value**
 42 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

1 4.4.Attribute of Job, Document and Page Operations

2 Attributes for printing jobs, documents and pages are given as parameters for `opvpStartJob()`, `opvpStartDoc()`,
 3 `opvpStartPage()`. Supported attributes are provided by printer driver database file distributed with drivers.

4 The attribute names and values MUST be given in ASCII strings in the following format;

5
 6 `<scheme>:<key>=<value>{,<value>}*{;<key>=<value>{,<value>}}*`

- 7 • `<scheme>`: Name space for the following `<key>` and `<value>`.
- 8 • `<key>`: Name of the attribute.
- 9 • `<value>`: Value for given `<key>`.

10
 11 Multiple `<value>` can be given with the separator “,” (comma) for each `<key>`. If multiple values are given for one key, printer
 12 driver should search the values from the beginning of the first value in the multiple values for each key, and take the first
 13 possible value that the driver can use under the current setting.

14 Several pairs - `<key>=<value>{,<value>}*` can be given with the separator “;” (semi-colon).

15 Driver which conforms to this specification MUST support the "updf" for `<scheme>` as defined in IEEE-ISTO PWG 5101.4
 16 "Universal Printer Definition Format" (May 2004) developed by the Printer Working Group.

17 Driver which conforms to this specification MUST ignore unknown properties to keep the compatibility with future vendor or
 18 standard extensions.

19 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	✓	✓	✓

20

1 The page attributes always override the document attributes between the `opvpStartPage()` and `opvpEndPage()`
2 functions call, and the document attributes always override the job attributes between the `opvpStartDoc()` and
3 `opvpEndDoc()` functions call. For example, if a job which contains three pages and has the landscape attribute given by the
4 `opvpStartJob()` function, and if the second page has the portrait attribute given by the `opvpStartPage()` function, the
5 first page and third page have the landscape attribute but second page has the portrait.

6

1 4.5. Graphics State Object Operations

2 These functions operate the properties or drawing attributes of the Graphics State Object. All of these functions are
3 OPTIONAL.

4 4.5.1. opvpResetCTM

5 **Name**

6 opvpResetCTM – Initialize CTM of the Graphics State Object.

7 **Synopsis**

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

10 **Arguments**

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

12 **Description**

13 This function initializes CTM of Graphics State Object. The initial setting of CTM is as follow.

$$14 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

15 **Return Value**

16 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

17 4.5.2. opvpSetCTM

18 **Name**

19 opvpSetCTM – Set CTM to the Graphics State Object.

20 **Synopsis**

```
21 opvp_result_t opvpSetCTM(  
22     opvp_dc_t printerContext,  
23     opvp_ctm_t *pCTM);
```

24 **Arguments**

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

26 pCTM – Pointer to the opvp_ctm_t structure. It contains 6 opvp_float_t elements - a, b, c, d, e and f.

27 **Description**

28 This function sets a CTM to the Graphics State Object.

29 Printer(device) coordinate system value $[x_{dev}, y_{dev}]$ is represented by CTM and caller(renderer) coordinate system value
30 $[x_{ren}, y_{ren}]$ in the following equation.

$$31 \begin{bmatrix} x_{dev} & y_{dev} & 1 \end{bmatrix} = \begin{bmatrix} x_{ren} & y_{ren} & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

32 **Structures**

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

1 **Return Value**
2 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

3 **4.5.3.opvpGetCTM**

4 **Name**
5 opvpGetCTM – Get CTM from the Graphics State Object.

6 **Synopsis**
7 opvp_result_t opvpGetCTM(
8 opvp_dc_t printerContext;
9 opvp_ctm_t *pCTM);

10 **Arguments**
11 printerContext – Printer context value returned by opvpOpenPrinter () function.
12 pCTM – Pointer to the opvp_ctm_t structure. It contains 6 opvp_float_t elements - a, b, c, d, e and f.

13 **Description**
14 This function gets CTM from the Graphics State Object.

15 **Structures**
16 typedef struct _opvp_ctm {
17 opvp_float_t a, b, c, d, e, f;
18 } opvp_ctm_t;

19 **Return Value**
20 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

21 **4.5.4.opvplnitGS**

22 **Name**
23 opvplnitGS – Initialize parameters in the Graphics State Object.

24 **Synopsis**
25 opvp_result_t opvplnitGS(
26 opvp_dc_t printerContext);

27 **Arguments**
28 printerContext – Printer context value returned by opvpOpenPrinter () function.

29 **Description**
30 This function initializes parameters in the Graphics State Object. Driver MUST initialize parameters in the Graphics State
31 Object when caller calls this function. Graphics State is a set of values which is managed by printer driver and contains
32 parameters for drawing color, drawing mode and other settings for drawing. Driver MUST keep the current parameters in the
33 Graphics State Object unless opvplnitGS () or other parameter setting functions are called. And also driver MUST save all
34 parameters in the Graphics State Object into the driver's stack when caller calls the opvpSaveGS () function, and restore all
35 parameters from the stack to the Graphics State Object when caller calls the opvpRestoreGS () function. These two
36 functions are described later in this document. These operations are used to change the parameters in the Graphics State Object
37 temporarily and restore them after drawing operations.

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

41 **Return Value**
42 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

1 4.5.5.opvpSaveGS

2 **Name**

3 opvpSaveGS – Save the Graphics State Object parameters.

4 **Synopsis**

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

7 **Arguments**

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

9 **Description**

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

11

12 If the opvpSaveGS() function can be called N (>0) times, the opvpRestoreGS() function also MUST be able to be called
13 N times. In this case, if the caller calls the opvpRestoreGS() function more than N times, this function MUST return error
14 and set the error code OPVP_BADREQUEST in opvpErrorNo.

15 **Return Value**

16 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

17 4.5.6.opvpRestoreGS

18 **Name**

19 opvpRestoreGS – Restore the Graphics State Object parameters.

20 **Synopsis**

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

23 **Arguments**

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

25 **Description**

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

28 If the opvpSaveGS() function can be called N (>0) times, the opvpRestoreGS() function also MUST be able to be called
29 N times. In this case, if the caller calls the opvpRestoreGS() function more than N times, this function MUST return error
30 and set the error code OPVP_BADREQUEST in opvpErrorNo.

31 **Return Value**

32 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

33 4.5.7.opvpQueryColorSpace

34 **Name**

35 opvpQueryColorSpace – Query color spaces that the driver can support.

36 **Synopsis**

```
37 opvp_result_t opvpQueryColorSpace(  
38     opvp_dc_t printerContext,  
39     opvp_int_t *pnnum,  
40     opvp_cspace_t *pcspace); ★順番入れ替えて良いか？
```

42 **Arguments**

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

1 `pnum` – Pointer to the buffer to store the number of the color space enum array elements.

2 `pcspace` – Pointer to the color space enum. array.

3 **Description**

4 This function queries the list of color space that the driver can support from the Graphics State Object.

5

6 To query the color space enum in `pcspace` array, caller MUST set the number of `pcspace` elements into `*pnum`. In this

7 case, driver MUST return the color space enum in `pcspace` array and also return the number of the color space enum retrieved

8 in `*pnum`.

9 If caller sets NULL in `*pcspace`, driver MUST return only the number of the color space enum that the driver can support in

10 `*pnum`.

11 If the number of the color space enum exceeds the number of `pcspace` array elements prepared by the caller, driver MUST

12 return the necessary number to retrieve all the color space enum in `*pnum`, and return error and set the error code

13 `OPVP_PARAMERROR` into `opvpErrorNo`.

14 Driver returns the color space enum in the `pcspace` array by its preferable order. The first color space enum SHOULD specify

15 that it is the most preferable color space for the driver.

16 **Return Value**

17 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

18 4.5.8.opvpSetColorSpace

19 **Name**

20 `opvpSetColorSpace` – Set current color space into the Graphics State Object.

21 **Synopsis**

```
22 opvp_result_t opvpSetColorSpace(
23     opvp_dc_t printerContext,
24     opvp_cspace_t cspace);
```

25 **Arguments**

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

27 `cspace` – Color space enum.

28 **Description**

29 This function sets the color space into the Graphics State Object.

30 Color space given by caller in `cspace` MUST be one of the color spaces retrieved by the `opvpQueryColorSpace()`

31 function.

32 **Return Value**

33 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

34 4.5.9.opvpGetColorSpace

35 **Name**

36 `opvpGetColorSpace` – Get the current color space from the Graphics State Object.

37 **Synopsis**

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

41 **Arguments**

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

43 `pcspace` – Pointer to the buffer to store the color space enum.

1 **Description**

2 This function gets the color space which is currently set in the Graphics State Object.

3
4 Initial color space in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each driver.

5 **Return Value**

6 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

7 **4.5.10.opvpQueryROP**

8 **Name**

9 `opvpQueryROP` – Query ROPs that the driver can support.

10 **Synopsis**

```
11 opvp_result_t opvpQueryROP(  
12     opvp_dc_t printerContext,  
13     opvp_int_t *pnum,  
14     opvp_rop_t *prop);
```

15 **Arguments**

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

17 `pnum` – Pointer to the buffer to store the number of the ROP array elements.

18 `prop` – Pointer to the ROP array.

19 **Description**

20 This function queries the list of ROPs (Raster Operations) that the driver can support.

21 To query ROPs in `prop` array, caller MUST set the number of `prop` elements into `*pnum`. In this case, driver MUST return ROPs in `prop` array and also return the number of the ROPs retrieved in `*pnum`.

23 If caller sets `NULL` in `*prop`, driver MUST return only the number of the ROPs that the driver can support in `*pnum`.

24 If the number of ROPs exceeds the number of `prop` array elements prepared by the caller, driver MUST return the necessary number to retrieve all ROPs in `*pnum`, and return error and set the error code `OPVP_PARAMERROR` into `opvpErrorNo`.

26 **Return Value**

27 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

28 **4.5.11.opvpSetROP**

29 **Name**

30 `opvpSetROP` – Set ROP mode into the Graphics State Object.

31 **Synopsis**

```
32 opvp_result_t opvpSetROP(  
33     opvp_dc_t printerContext,  
34     opvp_rop_t rop);
```

35 **Arguments**

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

37 `rop` – ROP (ROP3 code) to be set into the Graphics State Object.

38 **Description**

39 This function sets ROP into the Graphics State Object.

40 ROP given by caller in `rop` MUST be one of the ROPs retrieved by the `opvpQueryROP()` function.

41 **Return Value**

42 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

1 4.5.12.opvpGetROP

2 **Name**

3 opvpGetROP – Get the ROP mode from the Graphics State Object.

4 **Synopsis**

```
5 opvp_result_t opvpGetROP(  
6     opvp_dc_t printerContext,  
7     opvp_rop_t *prop);
```

8 **Arguments**

9 printerContext – Printer context value returned by opvpOpenPrinter() function.
10 prop – Pointer to the buffer to store ROP.

11 **Description**

12 This function gets ROP which is currently set in the Graphics State Object.

13 Initial ROP mode in the Graphics State Object returned by the opvpOpenPrinter() function depends on each driver.

15 **Return Value**

16 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

17 4.5.13.opvpSetFillMode

18 **Name**

19 opvpSetFillMode – Set filling mode into the Graphics State Object.

20 **Synopsis**

```
21 opvp_result_t opvpSetFillMode(  
22     opvp_dc_t printerContext,  
23     opvp_fillmode_t fillmode);
```

24 **Arguments**

25 printerContext – Printer context value returned by opvpOpenPrinter() function.
26 fillmode – Filling mode enum. OPVP_FILLMODE_EVENODD(Even-odd rule), OPVP_FILLMODE_WINDING(Nonzero
27 winding number rule) can be set.

28 **Description**

29 This function sets filling mode into the Graphics State Object.

30 **Return Value**

31 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

32 4.5.14.opvpGetFillMode

33 **Name**

34 opvpGetFillMode – Get the filling mode from the Graphics State Object.

35 **Synopsis**

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

39 **Arguments**

40 printerContext – Printer context value returned by opvpOpenPrinter() function.
41 pfillmode – Pointer to the buffer to store the fill mode enum.

1 **Description**

2 This function gets the filling mode which is currently set in the Graphics State Object.

3 Initial filling mode in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each driver.

4

5 **Return Value**

6 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

7 **4.5.15.opvpSetAlphaConstant**

8 **Name**

9 `opvpSetAlphaConstant` – Set alpha blending constant into the Graphics State Object.

10 **Synopsis**

```
11 opvp_result_t opvpSetAlphaConstant(  
12     opvp_dc_t printerContext,  
13     opvp_float_t alpha);
```

14 **Arguments**

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

16 `alpha` – Alpha blending constant. It MUST be between 0.0 and 1.0.

17 **Description**

18 This function sets the alpha blending constant, which is transparent ratio, into the Graphics State Object. The value `alpha`
19 MUST be between 0.0 and 1.0. If the value given by caller extends the range, driver SHOULD truncate it between 0.0 and 1.0.

20 **Return Value**

21 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

22 **4.5.16.opvpGetAlphaConstant**

23 **Name**

24 `opvpGetAlphaConstant` – Get the alpha blending constant from the Graphics State Object.

25 **Synopsis**

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

29 **Arguments**

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

31 `palpha` – Pointer to the buffer to store the alpha constant value.

32 **Description**

33 This function gets the alpha constant value which is currently set in the Graphics State Object.

34 Initial alpha blending constant in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each
35 driver.

36

37 **Return Value**

38 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

39 **4.5.17.opvpSetLineWidth**

40 **Name**

41 `opvpSetLineWidth` – Set the line width.

1 **Synopsis**
2 `opvp_result_t opvpSetLineWidth(
3 opvp_dc_t printerContext,
4 opvp_fix_t width);`

5 **Arguments**
6 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
7 `width` – Line width value.

8 **Description**
9 This function sets line width for stroke operations to the Graphics State Object. The line width MUST be set in the device
10 coordinate system unit.
11 The treatment of line width less than one depends on the device or driver implementation, and the maximum line width also
12 depends on the device or driver implementation.

13 **Return Value**
14 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

15 4.5.18.opvpGetLineWidth

16 **Name**
17 `opvpGetLineWidth` – Get the line width.

18 **Synopsis**
19 `opvp_result_t opvpGetLineWidth(
20 opvp_dc_t printerContext,
21 opvp_fix_t *pwidth);`

22 **Arguments**
23 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
24 `pwidth` – Pointer to the buffer to store the line width value.

25 **Description**
26 This function gets the line width for stroke operations from the Graphics State Object. The line width value MUST be in the
27 device coordinate system unit.
28 Initial line width in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each driver.

29
30 **Return Value**
31 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

32 4.5.19.opvpSetLineDash

33 **Name**
34 `opvpSetLineDash` – Set the line dash pattern.

35 **Synopsis**
36 `opvp_result_t opvpSetLineDash(
37 opvp_dc_t printerContext,
38 opvp_fix_t *pdash,
39 opvp_int_t num);`

40 **Arguments**
41 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
42 `pdash` – Pointer to the line dash pattern array.
43 `num` – Number of the line dash pattern array elements.

1 **Description**

2 This function sets a stroke dash pattern to the Graphics State Object.

3 When the painting mode is `OPVP_PAINTMODE_OPAQUE`(Opaque mode), the odd number (1st, 3rd, 5th, 7th ...) element in the
4 `pdash` array is the length for foreground color, and the even number (2nd, 4th, 6th, 8th ...) element is a length for background
5 color.

6 When the painting mode is `OPVP_PAINTMODE_TRANSPARENT`(Transparent mode), the odd number (1st, 3rd, 5th, 7th ...) element in the `pdash` array is the length for foreground color, and the even number (2nd, 4th, 6th, 8th ...) element is a length of a
7 line segment which is not painted.
8

9 The length set in the `pdash` array MUST be in the device coordinate system unit.

10 If the number of elements of the `pdash` array is odd, the dash pattern is created as if the number of element is double the
11 number of element of the array. In this case, the first element of the `pdash` array in the second cycle is treated as the length for
12 background color in case of `OPVP_PAINTMODE_OPAQUE` mode and for not-painted length in case of
13 `OPVP_PAINTMODE_TRANSPARENT` mode.

14 The maximum number of elements depend on the device or driver implementation.

15 If caller set zero in `num`, driver MUST draw solid lines.

16 **Return Value**

17 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

18 **4.5.20.opvpGetLineDash**

19 **Name**

20 `opvpGetLineDash` – Get the line dash pattern.

21 **Synopsis**

```
22 opvp_result_t opvpGetLineDash(  
23     opvp_dc_t printerContext,  
24     opvp_fix_t *pdash,  
25     opvp_int_t *pnum);
```

26 **Arguments**

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

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

29 `pnum` – Pointer to the buffer to store the number of the line dash pattern array elements.

30 **Description**

31 This function gets the stroke dash pattern from the Graphics State Object.

32 The caller MUST allocate at least 1 elements for `pdash` array. Also, the caller MUST set the number of element of `pdash`
33 allocated to `*pnum`. Driver MUST return the number of dash patterns retrieved in `*pnum`. If the number of dash patterns
34 exceeds the number of elements prepared by the caller, driver MUST return the necessary number to retrieve all the dash
35 patterns in `*pnum`.

36 If caller sets `NULL` in `pdash`, driver MUST return the number of the dash patterns which can be used by the driver in `*pnum`.

37 Initial line dash pattern in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each driver.

38

39 **Return Value**

40 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

41 **4.5.21.opvpSetLineDashOffset**

42 **Name**

43 `opvpSetLineDashOffset` – Set the line dash pattern offset.

44 **Synopsis**

```
45 opvp_result_t opvpSetLineDashOffset(  
46     opvp_dc_t printerContext,  
47     opvp_fix_t *pdash,  
48     opvp_int_t *pnum,  
49     opvp_int_t *poffset);
```

```
1         opvp_dc_t printerContext,  
2         opvp_fix_t offset);
```

3 **Arguments**

4 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
5 `offset` – Offset value for applying the line dash pattern.

6 **Description**

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

8 **Return Value**

9 `OPVP_OK` or `-1` when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

10 **4.5.22.opvpGetLineDashOffset**

11 **Name**

12 `opvpGetLineDashOffset` – Get the line dash pattern offset.

13 **Synopsis**

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

17 **Arguments**

18 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
19 `poffset` – Pointer to the buffer to store the offset value of the line dash pattern.

20 **Description**

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

22 Initial line dash pattern offset in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each
23 driver.

24

25 **Return Value**

26 `OPVP_OK` or `-1` when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

27 **4.5.23.opvpSetLineStyle**

28 **Name**

29 `opvpSetLineStyle` – Set line style.

30 **Synopsis**

```
31     opvp_result_t opvpSetLineStyle(  
32         opvp_dc_t printerContext,  
33         opvp_linestyle_t linestyle);
```

34 **Arguments**

35 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
36 `linestyle` – Line style enum. `OPVP_LINESTYLE_SOLID`(Solid line) or `OPVP_LINESTYLE_DASH`(Dashed line) can be
37 set.

38 **Description**

39 This function sets the line style to the Graphics State Object.

40 **Return Value**

41 `OPVP_OK` or `-1` when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

4.5.24.opvpGetLineStyle

Name

opvpGetLineStyle – Get the line style.

Synopsis

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

Arguments

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

plinestyle – Pointer to the buffer to store the line style enum.

Description

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

Initial line style in the Graphics State Object returned by the opvpOpenPrinter() function depends on each driver.

Return Value

OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

4.5.25.opvpSetLineCap

Name

opvpSetLineCap – Set line cap style.

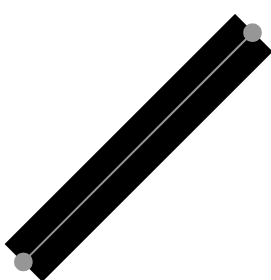
Synopsis

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

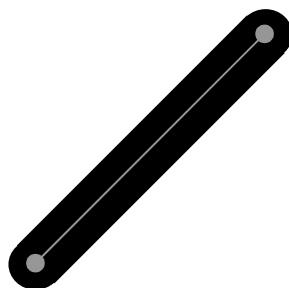
Arguments

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

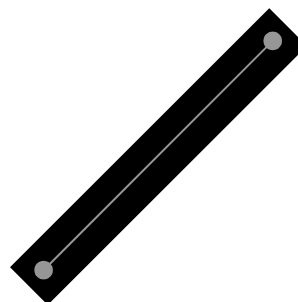
linecap – Line cap style enum. OPVP_LINECAP_BUTT(Butt cap), OPVP_LINECAP_ROUND(Round cap) or OPVP_LINECAP_SQUARE(Square projection cap) can be set.



Butt cap



Round cap



Square projection cap

Description

This function sets the line cap style to the Graphics State Object.

Return Value

OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

4.5.26.opvpGetLineCap

Name

opvpGetLineCap – Get the line cap style.

Synopsis

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

Arguments

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

plinecap – Pointer to the buffer to store the line cap style enum.

Description

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

Initial line cap style in the Graphics State Object returned by the opvpOpenPrinter() function depends on each driver.

Return Value

OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

4.5.27.opvpSetLineJoin

Name

opvpSetLineJoin – Set line join style.

Synopsis

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

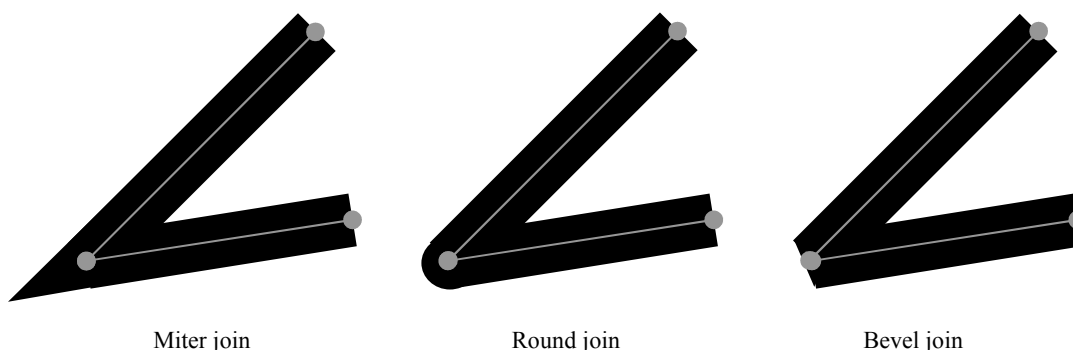
Arguments

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

linejoin – Line join style enum. OPVP_LINEJOIN_MITER(Miter join), OPVP_LINEJOIN_ROUND(Round join) or OPVP_LINEJOIN_BEVEL(Bevel join) can be set.

Description

This function sets the line join style to the Graphics State Object.



1 **Return Value**
2 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

3 4.5.28.opvpGetLineJoin

4 **Name**
5 opvpGetLineJoin – Get the line join style.

6 **Synopsis**
7 opvp_result_t opvpGetLineJoin(
8 opvp_dc_t printerContext,
9 opvp_linejoin_t *plinejoin);

10 **Arguments**
11 printerContext – Printer context value returned by opvpOpenPrinter() function.
12 plinejoin – Pointer to the buffer to store the line join style enum.

13 **Description**
14 This function gets the line join style from the Graphics State Object.
15 Initial line join style in the Graphics State Object returned by the opvpOpenPrinter() function depends on each driver.
16

17 **Return Value**
18 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

19 4.5.29.opvpSetMiterLimit

20 **Name**
21 opvpSetMiterLimit – Set miter limit value.

22 **Synopsis**
23 opvp_result_t opvpSetMiterLimit(
24 opvp_dc_t printerContext,
25 opvp_fix_t miterlimit);

26 **Arguments**
27 printerContext – Printer context value returned by
28 opvpOpenPrinter() function.
29 miterlimit – Maximum miter length.

30 **Description**
31 This function sets the maximum length of miter to the Graphics State Object. The miter limit is effective only when the line join
32 style is OPVP_LINEJOIN_MITER. The length MUST be set in the device coordinate system unit.

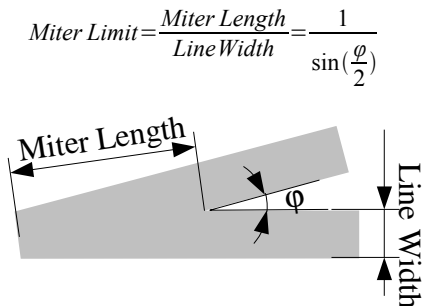
33
34

35 **Return Value**
36 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

37 4.5.30.opvpGetMiterLimit

38 **Name**
39 opvpGetMiterLimit – Get the miter limit value.

40 **Synopsis**
41 opvp_result_t opvpGetMiterLimit(
42 opvp_dc_t printerContext,
43 opvp_fix_t *miterlimit);



```
1         opvp_dc_t printerContext,  
2         opvp_fix_t *pmiterlimit);
```

3 **Arguments**

4 printerContext – Printer context value returned by opvpOpenPrinter() function.
5 pmiterlimit – Pointer to the buffer to store the maximum miter length.

6 **Description**

7 This function gets the maximum length of miter from the Graphics State Object.
8 Initial miter limit value in the Graphics State Object returned by the opvpOpenPrinter() function depends on each driver.
9

10 **Return Value**

11 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

12 **4.5.31.opvpSetPaintMode**

13 **Name**

14 opvpSetPaintMode – Set the background color painting mode.

15 **Synopsis**

```
16 opvp_result_t opvpSetPaintMode(  
17     opvp_dc_t printerContext,  
18     opvp_paintmode_t paintmode);
```

19 **Arguments**

20 printerContext – Printer context value returned by opvpOpenPrinter() function.
21 paintmode – Painting mode enum. OPVP_PAINTMODE_OPAQUE(Opaque mode) or OPVP_PAINTMODE_TRANSPARENT
22 (Transparent mode) can be set.

23 **Description**

24 This function sets the background color painting mode to the Graphics State Object.
25

26 **Return Value**

27 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

28 **4.5.32.opvpGetPaintMode**

29 **Name**

30 opvpGetPaintMode – Get the background color painting mode.

31 **Synopsis**

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

35 **Arguments**

36 printerContext – Printer context value returned by opvpOpenPrinter() function.
37 ppaintmode – Pointer to the buffer to store the painting mode enum.

38 **Description**

39 This function gets the background color painting mode from the Graphics State Object.
40 Initial painting mode in the Graphics State Object returned by the opvpOpenPrinter() function depends on each driver.
41

1 **Return Value**
2 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

3 4.5.33.opvpSetStrokeColor

4 **Name**
5 opvpSetStrokeColor – Define the stroke color and pattern for opvpStrokePath() and opvpStrokeFillPath()
6 operations.

7 **Synopsis**
8 opvp_result_t opvpSetStrokeColor(
9 opvp_dc_t printerContext,
10 opvp_brush_t *brush);

11 **Arguments**
12 printerContext – Printer context value returned by opvpOpenPrinter() function.
13 brush – Pointer to the opvp_brush_t structure data.

14 **Description**
15 This function registers the color or pattern as a brush data to the Graphics State Object for drawing stroke by the
16 opvpStrokePath() and opvpStrokeFillPath() operations.
17 If the member pbrush in opvp_brush_t structure *brush is NULL, driver MUST treat it as a solid brush. In this case,
18 color value of the solid brush is specified in color[] in the opvp_brush_t structure *brush, and the color space for the
19 solid brush is specified by colorSpace.
20 If the member pbrush in opvp_brush_t structure *brush points to a opvp_brushdata_t structure data, driver MUST
21 treat it as a pattern brush. In the opvp_brushdata_t structure data, width, height are specified in pixel value, and the
22 brush pattern horizontal repetition pitch pitch is specified in the number of bytes. The actual pattern data is specified by
23 data array. The color space for the pattern is specified by colorSpace in the opvp_brush_t structure, and color[]
24 MUST be ignored.

25 **Structures**
26 typedef struct _opvp_brushdata {
27 opvp_bdtype_t type;
28 opvp_int_t width, height, pitch;
29 opvp_byte_t data[]; /* must be defined as data[1] for GCC 2.x */
30 } opvp_brushdata_t;
31
32 typedef struct _opvp_brush {
33 opvp_cspace_t colorSpace;
34 opvp_int_t color[4];
35 opvp_int_t xorg, yorg; /* brush origin ,ignored for opvpSetBgColor */
36 opvp_brushdata_t *pbrush; /* pointer to brush data */
37 } opvp_brush_t;
38
39 typedef enum _opvp_bdtype {OPVP_BDTYPE_NORMAL = 0} opvp_bdtype_t;

41 **Return Value**
42 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

43 4.5.34.opvpSetFillColor

44 **Name**
45 opvpSetFillColor – Define the fill color and pattern for opvpFillPath() and opvpStrokeFillPath() operations.

46 **Synopsis**
47 opvp_result_t opvpSetFillColor(
48 opvp_dc_t printerContext,
49 opvp_brush_t *brush);

1 **Arguments**

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

3 `brush` – Pointer to brush structure data.

4 **Description**

5 This function registers the color or pattern as a brush data to the Graphics State Object for filling by the `opvpFillPath()`
6 and `opvpStrokeFillPath()` operations..

7 The definition and operation of the `opvp_brush_t` structure `*brush` is same as the `opvpSetStrokeColor()` function
8 except filling inside the path by the color or pattern with the brush registered by this function.

9 See the `opvpSetStrokeColor()` function description.

10 Initial fill color in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each driver.

11

12 **Return Value**

13 `OPVP_OK` or -1 when error , and the driver MUST store the detailed error code in `opvpErrorNo`.

14 **4.5.35.opvpSetBgColor**

15 **Name**

16 `opvpSetBgColor` – Define the background color and pattern for `opvpStrokePath()`, `opvpFillPath()` and
17 `opvpStrokeFillPath()` operations.

18 **Synopsis**

```
19 opvp_result_t SetBgColor(  
20     opvp_dc_t printerContext,  
21     opvp_brush_t *brush);
```

22 **Arguments**

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

24 `brush` – Pointer to brush structure data.

25 **Description**

26 This function registers the background color or pattern for `opvpStrokePath()`, `opvpFillPath()` and
27 `opvpStrokeFillPath()` operations.

28 Driver MUST refer only the member `colorSpace` and `color[4]` of the `opvp_brush_t` structure, and ignore `xorg` and
29 `yorg`.

30 Initial background color in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each driver.

31

32 Caller MUST set `NULL` to the member `pbrush` in the `opvp_brush_t` structure, or this function MUST return error and set
33 the error code `OPVP_BADREQUEST` in `opvpErrorNo`.

34 **Return Value**

35 `OPVP_OK` or -1 when error , and the driver MUST store the detailed error code in `opvpErrorNo`.

1 4.6.Path Operations

2 Path is used for the drawing command, such as “stroke”, “fill” and “stroke and fill” as well as for defining the clipping area.
3 One Graphics State Object keeps only one path at once. Path is affected by CTM and registered into the Graphics State Object
4 when caller calls each path operation function.

5 All path operation functions are OPTIONAL.

6 4.6.1.opvpNewPath

7 **Name**

8 opvpNewPath – Start a new path

9 **Synopsis**

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

12 **Arguments**

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

14 **Description**

15 Delete the current path kept in the Graphics State Object and start a new path which MUST be empty.

16 Initial path in the Graphics State Object returned by the opvpOpenPrinter() function MUST be empty.

17

18 **Return Value**

19 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

20 4.6.2.opvpEndPath

21 **Name**

22 opvpEndPath – Declare the end of the current path

23 **Synopsis**

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

26 **Arguments**

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

28 **Description**

29 Declare the end of the current path

30 Graphics State Object MUST retain the path as the current path.

31 **Return Value**

32 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

33 4.6.3.opvpStrokePath

34 **Name**

35 opvpStrokePath – Draw lines along the current path

36 **Synopsis**

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

1 **Arguments**

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

3 **Description**

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

5 The current path MUST be retained in the Graphics State Object after calling this function.

6 **Return Value**

7 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

8 **4.6.4.opvpFillPath**

9 **Name**

10 `opvpFillPath` – Fill inside the current path

11 **Synopsis**

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

14 **Arguments**

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

16 **Description**

17 This function fills inside the current path according to the drawing attributes registered in the Graphics State Object.

18 When the path is not closed, the starting point and end point of the current path are connected by a straight line (but not stroked)
19 and closed.

20 The current path MUST be retained in the Graphics State Object after calling this function.

21 **Return Value**

22 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

23 **4.6.5.opvpStrokeFillPath**

24 **Name**

25 `opvpStrokeFillPath` – Draw lines along the current path and fill inside the current path

26 **Synopsis**

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

29 **Arguments**

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

31 **Description**

32 This function draws lines along the current path, and fills inside the current path according to the drawing attributes registered
33 in the Graphics State Object.

34 When the path is not closed, the starting point and end point of the current path are connected by a straight line (but not stroked)
35 and closed.

36 The current path MUST be retained in the Graphics State Object after calling this function.

37 **Return Value**

38 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

4.6.6.opvpSetClipPath

Name

opvpSetClipPath – Set the current path as a clipping region

Synopsis

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

Arguments

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

clipRule – Clipping rule enum. OPVP_CLIPRULE_EVENODD(Even-odd rule) or OPVP_CLIPRULE_WINDING(Nonzero winding number rule) can be set.

Description

This function sets the current path as a clipping region in 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 Graphics State Object after calling this function.

The clipping region also MUST be retained except caller calls the opvpSetClipPath(), opvpResetClipPath(), opvpRestoreGS() or opvpStartPage() function. When caller calls opvpStartPage() function, driver MUST reset the clipping region in the Graphics State Object to cover whole the printable area of the media.

Return Value

OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

4.6.7.opvpResetClipPath

Name

opvpResetClipPath – Reset the clipping region

Synopsis

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

Arguments

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

Description

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

Return Value

OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

4.6.8.opvpSetCurrentPoint

Name

opvpSetCurrentPoint – Set the current point

Synopsis

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

1 Arguments

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

3 x – x coordinate value to set the current point.

4 y – y coordinate value to set the current point.

5 Description

6 Set the current point to (x, y).

7 Return Value

8 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

9 4.6.9.opvpLinePath

10 Name

11 opvpLinePath – Add multiple connected lines to the current path

12 Synopsis

```
13 opvp_result_t opvpLinePath(  
14     opvp_dc_t printerContext,  
15     opvp_pathmode_t flag;  
16     opvp_int_t npoints,  
17     opvp_point_t *points);
```

18 Arguments

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

20 flag – Path open/close flag. OPVP_PATHCLOSE or OPVP_PATHOPEN can be set.

21 npoints – Number of the opvp_point_t structure array elements.

22 points – Pointer to the opvp_point_t structure array.

23 Description

24 This function adds lines specified by the points array from the current point.

25 When caller sets OPVP_PATHOPEN in flag, driver MUST append lines from the current point, then set the last point of the
26 points array as the current point. When caller sets OPVP_PATHCLOSE in flag, driver MUST append lines from the current
27 point, then set the first point of the points array as the current point.

28 Structures

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

32 Return Value

33 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

34 4.6.10.opvpPolygonPath

35 Name

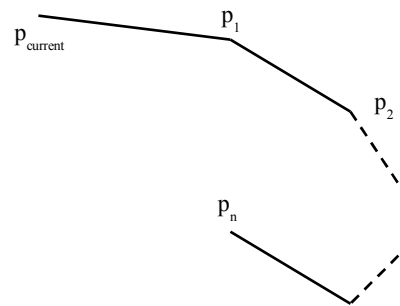
36 opvpPolygonPath – Add polygons to the current path

37 Synopsis

```
38 opvp_result_t opvpPolygonPath(  
39     opvp_dc_t printerContext,  
40     opvp_int_t npolygons,  
41     opvp_int_t *nvertexes,  
42     opvp_point_t *points);
```

43 Arguments

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



In the above case, each argument is as following:

```
npolygons=3  
*nvertexes={4, 4, 3}  
*points={p0, p1, ... p10}
```


1 npolygons – Number of polygons to add.
 2 nvertexes – Pointer to the array of the number of points of each polygon.
 3 points – Pointer to the opvp_point_t structure array. The number of points in the array MUST be equal to the total
 4 number of points in nvertexes array.

5 **Description**

6 This function adds polygons specified by the points array into the current path..
 7 After driver appends polygons, driver MUST set the last point of the points array as the current point.

8 **Return Value**

9 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

10 **4.6.11.opvpRectanglePath**

11 **Name**

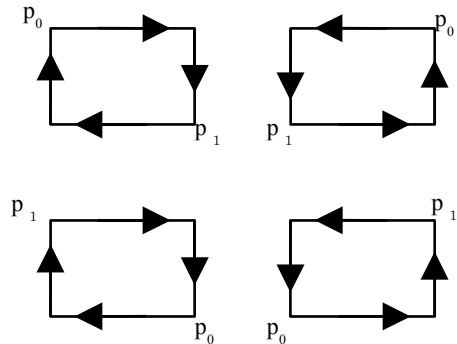
12 opvpRectanglePath – Add rectangles to the current path.

13 **Synopsis**

```
14    opvp_result_t opvpRectanglePath(  
15        opvp_dc_t printerContext,  
16        opvp_int_t nrectangles,  
17        opvp_rectangle_t *rectangles);
```

18 **Arguments**

19 printerContext – Printer context value returned by
 20 opvpOpenPrinter() function.
 21 nrectangles – Number of rectangles to add.
 22 rectangles – Pointer to the opvp_rectangle structure array.



23 **Description**

24 This function adds rectangles into the current path..
 25 After driver appends rectangles, driver MUST set the starting point of the last rectangle appended as the current point.
 26 Direction of paths of each rectangle are specified by the starting point and diagonal point as above figure. Path is appended in
 27 order of (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).

28 **Structures**

```
29    typedef struct opvp_rectangle {  
30        opvp_point_t p0;                                    /* starting point */  
31        opvp_point_t p1;                                    /* diagonal point */  
32    } opvp_rectangle_t;
```

33 **Return Value**

34 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

35 **4.6.12.opvpRoundRectanglePath**

36 **Name**

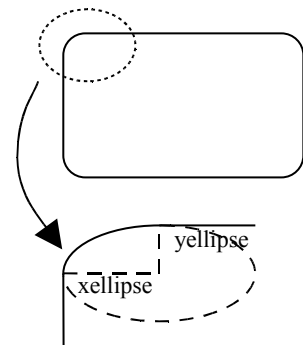
37 opvpRoundRectanglePath – Add round rectangles to the current path.

38 **Synopsis**

```
39    opvp_result_t opvpRoundRectanglePath(  
40        opvp_dc_t printerContext,  
41        opvp_int_t nrectangles,  
42        opvp_roundrectangle_t *rectangles);
```

43 **Arguments**

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



1 nrectangles – Number of round rectangles to add.
2 rectangles – Pointer to the opvp_roundrectangle_t structure array.

3 **Description**

4 This function adds round rectangles into the current path..

5 Each corner of round rectangle is connected by elliptic arc defined by xellipse and yellipse in the
6 opvp_roundrectangle_t structure. After driver appends round rectangles, driver MUST set the starting point of the last
7 round rectangle appended as the current point.

8 Direction of paths of each round rectangle MUST be treated as the opvpRectanglePath () function.

9 **Structures**

```
10 typedef struct opvp_roundrectangle {  
11     opvp_point_t p0;          /* starting point */  
12     opvp_point_t p1;          /* diagonal point */  
13     opvp_fix_t xellipse, yellipse;  
14 } opvp_roundrectangle_t;
```

15 **Return Value**

16 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

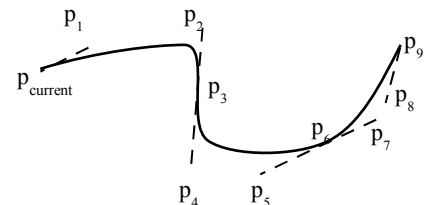
17 **4.6.13.opvpBezierPath**

18 **Name**

19 opvpBezierPath – Add 3D-bezier paths to the current path.

20 **Synopsis**

```
21 opvp_result_t opvpBezierPath(  
22     opvp_dc_t printerContext,  
23     opvp_int_t npoints,  
24     opvp_point_t *points);
```



In the above case, each arguments
are as following:

npoints = 9

*points = {p₁, p₂, ... p₉}

25 **Arguments**

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

28 npoints – Number of the points array elements. It MUST be multiple number of 3.

29 points – Pointer to the array of end points and control points of bezier curves.

30 **Description**

31 This function adds multiple bezier paths specified by the current point and end points and the control points given by the
32 points array. The first bezier path is started by the current point and the third point in the points array is the end points.

33 The first point and second point in the points array is the control points of the first bezier path. Following bezier paths are
34 started by the previous path's end point and the following two points are the control points and the next point is the end points.

35 After driver appends bezier paths from the current point, driver MUST set the last point of the points array as the current
36 point.

37 If npoints is not a multiple number of 3, this function MUST return error and set the error code OPVP_PARAMERROR in
38 opvpErrorNo.

39 **Return Value**

40 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

41 **4.6.14.opvpArcPath**

42 **Name**

43 opvpArcPath – Add arcs, chords, pies, or ellipses to the current path.

44 **Synopsis**

```
45 opvp_result_t opvpArchPath(  
46     opvp_dc_t printerContext,
```

```
1     opvp_arcmode_t kind,  
2     opvp_arcdir_t dir,  
3     opvp_fix_t bbx0, opvp_fix_t bby0, opvp_fix_t bbx1, opvp_fix_t bby1,  
4     opvp_fix_t x0, opvp_fix_t y0,  
5     opvp_fix_t x1, opvp_fix_t y1);
```

6 **Arguments**

7 printerContext – Printer context value returned by opvpOpenPrinter() function.
8 kind – Arc kind flag. OPVP_ARC(Arc), OPVP_CHORD(Chord), OPVP_PIE(Pie) can be set.
9 dir – Direction of the path. OPVP_CLOCKWISE(Clockwise), OPVP_COUNTERCLOCKWISE(Counter-clockwise) can be set.
10 bbx0, bby0, bbx1, bby1 – Circumscribe rectangle.
11 x0, y0 – Starting point.
12 x1, y1 – End point.

13 **Description**

14 This function adds an arc, chord or a pie into the current path. The center point of ellipse is the middle point of the circumscribe
15 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
16 start and end points, driver MUST append ellipse into the path. If the circumscribe rectangle is a square, driver MUST add
17 circle to the path.

18 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,
19 driver MUST set the left-top point of the circumscribe rectangle as the current point.

20 **Return Value**

21 OPVP_OK or -1 when error, and the driver MUST store the detailed error code in opvpErrorNo.

22

23

24

1 4.7. Bitmap Image Operations

2 Bitmap is a pixel oriented image data which is drawn in a rectangle region. Bitmap is drawn according to the drawing attributes
3 registered in the Graphics State Object.

4 Typical bitmap drawing sequence is as following.

- 5 ● `opvpStartDrawImage()` – Specify a image data.
- 6 ● `opvpTransferDrawImage()` – Transfer the actual image data (Caller calls this function one or more times)
- 7 ● `opvpEndDrawImage()` – Declare the end of transferring the image data.

8 If caller calls any functions except `opvpTransferDrawImage()` function between the `opvpStartDrawImage()` and
9 `opvpEndDrawImage()` functions, driver MUST return error and set the error code `OPVP_BADREQUEST` in
10 `opvpErrorNo`.

11 The `opvpDrawImage()` is the function that performs the `opvpStartDrawImage()`, `opvpTransferDrawImage()`
12 and `opvpEndDrawImage()` function calls by one function call.

13 All bitmap operation functions are OPTIONAL.

14 4.7.1. `opvpDrawImage`

15 **Name**

16 `opvpDrawImage` – Draw a bitmap image

17 **Synopsis**

```
18 opvp_result_t opvpDrawImage(  
19     opvp_dc_t printerContext,  
20     opvp_int_t sourceWidth,  
21     opvp_int_t sourceHeight,  
22     opvp_int_t sourcePitch,  
23     opvp_int_t colorDepth,  
24     opvp_imageformat_t imageFormat,  
  
26     opvp_int_t destinationWidth,  
27     opvp_int_t destinationHeight,  
28     void *imageData);
```

29 **Arguments**

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

31 `sourceWidth` – Width (pixels) of source image.

32 `sourceHeight` – Height (pixels) of source image.

33 `sourcePitch` – Repetition pitch (bytes) of source image. ★三原さん図を入れる

34 `colorDepth` – Number of bits per pixel of source image.

35 `imageFormat` – Image format enum to specify the source image format.

36

37 `destinationWidth` – Destination drawing width (pixels).

38 `destinationHeight` – Destination drawing height (pixels).

39 `imageData` – Pointer to actual source image data.

40 **Description**

41 This function draws a bitmap image. Destination image drawing area is specified by both the current point kept in the Graphics
42 State Object and `destinationSize`. The upper-left corner of the image MUST be drawn on the current point and the
43 bottom-right corner of the image MUST be drawn at $(x_{current} + destinationWidth - 1, y_{current} + destinationHeight - 1)$ where $x_{current}$
44 is the current point x coordinate value and $y_{current}$ is the current point y coordinate value.

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

46

47 To specify the color space of the image, caller MUST call the `opvpSetColorSpace()` function before calling this function.

- 1 Color space enum which is set to the `opvpSetColorSpace()` function MUST be one of the color space enum that the
 2 driver supports. ★これで行いか確認
- 3 Only `OPVP_IFORMAT_RAW` MUST be supported by driver, and caller MUST set `OPVP_IFORMAT_RAW` in `imageFormat`.
 4 Other image format enum (`OPVP_IFORMAT_JPEG`, `OPVP_IFORMAT_PNG`, `OPVP_IFORMAT_RLE`, etc) are reserved for
 5 future use.
- 6 After drawing image, driver MUST not change the current point.

7 **Return Value**

- 8 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

9 **4.7.2.opvpStartDrawImage**

10 **Name**

- 11 `opvpStartDrawImage` – Start to draw a bitmap image

12 **Synopsis**

```
13 opvp_result_t opvpStartDrawImage(  
14     opvp_dc_t printerContext,  
15     opvp_int_t sourceWidth,  
16     opvp_int_t sourceHeight,  
17     opvp_int_t sourcePitch,  
18     opvp_int_t colorDepth,  
19     opvp_imageformat_t imageFormat,  
  
21     opvp_int_t destinationWidth,  
22     opvp_int_t destinationHeight,  
23     );
```

24 **Arguments**

- 25 `printerContext` – Printer context value returned by `opvpOpenPrinter()` function.
- 26 `sourceWidth` – Width (pixels) of source image.
- 27 `sourceHeight` – Height (pixels) of source image.
- 28
- 29 `sourcePitch` – Repetition pitch (bytes) of source image.
- 30 `colorDepth` – Number of bits per pixel of source image.
- 31 `imageFormat` – Image format enum to specify the source image format.
- 32
- 33 `destinationWidth` – Destination drawing width (pixels).
- 34 `destinationHeight` – Destination drawing height (pixels).

35 **Description**

- 36 This function starts to draw a bitmap image. Destination image drawing area is specified by both the current point kept in the
 37 Graphics State Object and `destinationSize`. The upper-left corner of the image MUST be drawn on the current point and
 38 the bottom-right corner of the image MUST be drawn at $(x_{current} + destinationWidth - 1, y_{current} + destinationHeight - 1)$ where
 39 $x_{current}$ is the current point x coordinate value and $y_{current}$ is the current point y coordinate value.
- 40 If the current point coordinate value is not an integer, the reference point for drawing image can be rounded to an integer.
- 41 After caller calls this function, caller MUST call `opvpTransferDrawImage()` function once or more times to transfer the
 42 actual image data to driver. If caller calls any functions except the `opvpTransferDrawImage()` function between the
 43 `opvpStartDrawImage()` and `opvpEndDrawImage()` function, driver MUST return error and set the error code
 44 `OPVP_BADREQUEST` in `opvpErrorNo`.
- 45
- 46 To specify the color space of the image, caller MUST call the `opvpSetColorSpace()` function before calling this function.
 47 Color space enum which is set to the `opvpSetColorSpace()` function MUST be one of the color space enum that the
 48 driver supports. ★これで行いか確認
- 49 Only `OPVP_IFORMAT_RAW` MUST be supported by driver, and caller MUST set `OPVP_IFORMAT_RAW` in `imageFormat`.

1 Other image format enum (OPVP_IFORMAT_JPEG, OPVP_IFORMAT_PNG, OPVP_IFORMAT_RLE, etc) are reserved for
2 future use.
3 After drawing image, driver MUST not change the current point.

4 **Return Value**

5 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

6 **4.7.3.opvpTransferDrawImage**

7 **Name**

8 opvpTransferDrawImage – Transfer the actual bitmap image data

9 **Synopsis**

```
10 opvp_result_t opvpTransferDrawImage(  
11     opvp_dc_t printerContext,  
12     opvp_int_t count,  
13     void *imageData);
```

14 **Arguments**

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

16 count – Number of image bytes to transfer.

17 imageData – Pointer to actual source image data.

18 **Description**

19 This function transfers a bitmap image data. The image data MUST be declared to start by the opvpStartDrawImage ()
20 function before calling this function.

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

22 **Return Value**

23 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

24 **4.7.4.opvpEndDrawImage**

25 **Name**

26 opvpEndDrawImage – Declare the end of transferring the image data

27 **Synopsis**

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

30 **Arguments**

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

32 **Description**

33 This function declares the end of transferring the image data.

34 **Return Value**

35 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

1 4.8. Scan Line Operations

2 Scan line operations provide the support for horizontal line drawing which are defined by one or multiple pairs of begin point
3 and end point. Scan lines are drawn according to the drawing attributes registered in the Graphics State Object.

4 All scan line operations are OPTIONAL. However support for scan line operations is REQUIRED in the case that the path
5 operations are not supported by driver, and caller needs to apply different color drawing schemes for bitmap drawing and path
6 drawing.

7 If caller calls any functions except `opvpTransferScanline()` function between the `opvpStartScanline()` and
8 `opvpEndScanline()` functions, driver MUST return error and set the error code `OPVP_BADREQUEST` in `opvpErrorNo`.

9
10

11 4.8.1. `opvpStartScanline`

12 **Name**

13 `opvpStartScanline` – Start a scan line drawing

14 **Synopsis**

```
15 opvp_result_t opvpStartScanline(  
16     opvp_dc_t printerContext,  
17     opvp_int_t yposition);
```

18 **Arguments**

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

20 `yposition` – Vertical position to start a scan line.

21 **Description**

22 This function declares to start a scan line drawing. Scan lines are drawn by three functions. The `opvpStartScanline()`
23 function declares the start of scan lines drawing, the `opvpScanline()` function transfers the actual scan lines data, and
24 `opvpEndScanline()` function terminates the scan line drawing operations. Driver MUST not change the current point after
25 these operations.

26 **Return Value**

27 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

28 4.8.2. `opvpScanline`

29 **Name**

30 `opvpScanline` – Draw the scan lines

31 **Synopsis**

```
32 opvp_result_t opvpScanline(  
33     opvp_dc_t printerContext,  
34     opvp_int_t nscanpairs,  
35     opvp_int_t *scanpairs);
```

36 **Arguments**

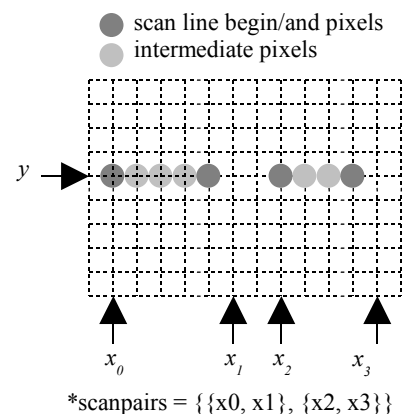
37 `printerContext` – Printer context value returned by `opvpOpenPrinter()`
38 function.

39 `nscanpairs` – Number of scan lines specified by begin and end x position pairs

40 `scanpairs` – Pointer to scan line array specified by begin and end x position pairs

41 **Description**

42 This function draws the scan lines. After caller calls this function, driver MUST increment the y coordinate value of the current
43 scan line by 1. The current scan line y coordinate value is different from the current point y coordinate value, and driver MUST
44 keep the current scan line y coordinate value temporarily between `opvpStartScanLine()` and `opvpEndScanLine()`
45 functions in the Graphics State Object. Driver MUST not change the current point after calling this function.



1

2 **Return Value**

3 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

4 **4.8.3.opvpEndScanline**

5 **Name**

6 opvpEndScanline – Terminate the scan line drawing

7 **Synopsis**

8 opvp_result_t opvpEndScanline(
9 opvp_dc_t printerContext);

10 **Arguments**

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

12 **Description**

13 This function terminates the scan line drawing.

14 **Return Value**

15 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

1 4.9.Raster Image Operations

2 Raster image operations provide the support for raster image drawing. All raster image operations are OPTIONAL. However, in
3 case of driver does not support bitmap image operations nor path operations, these operations are MUST be supported.

4 If caller calls any functions except `opvpTransferRasterData()` or `opvpSkipRaster()` functions between the
5 `opvpStartRaster()` and `opvpEndRaster()` functions, driver MUST return error and set the error code
6 `OPVP_BADREQUEST` in `opvpErrorNo`.

7
8
9

10 4.9.1.opvpStartRaster

11 **Name**

12 `opvpStartRaster` – Declare to start a raster image drawing

13 **Synopsis**

```
14 opvp_result_t opvpStartRaster(  
15     opvp_dc_t printerContext,  
16     opvp_int_t rasterWidth);
```

17 **Arguments**

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

19 `rasterWidth` – Width (pixels) of actual raster image (MUST not included padding data)

20 **Description**

21 This function declares to start a raster data drawing.

22 Driver MUST draw the raster image from the current point.

23 Driver MUST use the color space registered in the current Graphics State Object for drawing raster image.

24

25 ★三原さんの図 MSB Byte オーダ

26 Compressed raster data is not supported by this function. However, driver can perform its suitable compression inside it.

27 Raster data color spaces and format are defined as the following table.

Color Space	# of planes	Bits per Plane	Bits per Pixel	Bytes per Pixel	Note
OPVP_CSPACE_BW	1	1	1	1/8	padding bits are added to the rightmost byte if necessary
OPVP_CSPACE_DEVICEGRAY	1	8	8	1	
OPVP_CSPACE_STANDARDRGB	3	8	24	3	順番は RGB である
OPVP_CSPACE_STANDARDRGB64	3	16	48	6	
OPVP_CSPACE_DEVICECMY	3	8	24	3	
OPVP_CSPACE_DEVICECMYK	4	8	32	4	

28 **Return Value**

29 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

30 4.9.2.opvpTransferRasterData

31 **Name**

32 `opvpTransferRasterData` – Transfer raster image data

33 **Synopsis**

```
34 opvp_result_t opvpTransferRasterData (
```

```
1     opvp_dc_t printerContext,  
2     opvp_int_t count,  
3     opvp_byte_t *data);
```

4 **Arguments**

5 printerContext – Printer context value returned by opvpOpenPrinter () function.
6 count – Number of pixel data elements.
7 data – Pointer to pixel data array.

8 **Description**

9 This function transfers count pixel data elements of a single row raster data. Driver MUST ignore the given data when the
10 data pixel length exceeds the rasterWidth pixels given by opvpStartRaster () function. If caller sets the lesser
11 count than the rasterWidth width given by opvpStartRaster () function, driver MUST not draw the remaining
12 data which exceeds the given count.

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

14 **Return Value**

15 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

16 **4.9.3.opvpSkipRaster**

17 **Name**

18 opvpSkipRaster – Skips lines during raster image drawing

19 **Synopsis**

```
20 opvp_result_t opvpSkipRaster(  
21     opvp_dc_t printerContext,  
22     opvp_int_t count);
```

23 **Arguments**

24 printerContext – Printer context value returned by opvpOpenPrinter () function.
25 count – Number of lines to be skipped

26 **Description**

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

29 **Return Value**

30 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

31 **4.9.4.opvpEndRaster**

32 **Name**

33 opvpEndRaster – Terminates the raster image drawing

34 **Synopsis**

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

37 **Arguments**

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

39 **Description**

40 This function terminates the raster image drawing.

41 **Return Value**

42 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

1 4.10.Stream Data Operations

2 Stream data operations provide the support that an application directly creates a PDL and sends the data to a device. All stream
3 data operations are OPTIONAL. When caller calls these functions with Path, Bitmap Image, Scan Line, or Raster Image
4 operation functions, the result of the printing is not guaranteed.

5 If caller calls any functions except `opvpTransferStreamData()` function between the `opvpStartStream()` and
6 `opvpEndStream()` functions, driver MUST return error and set the error code `OPVP_BADREQUEST` in `opvpErrorNo`.

7
8
9

10 4.10.1.opvpStartStream

11 **Name**

12 `opvpStartStream` – Start streaming data transfer

13 **Synopsis**

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

16 **Arguments**

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

18 **Description**

19 This function starts streaming data transfer. Streaming data is the data that is directly sent to the printer device without being
20 processed by the driver.

21 **Return Value**

22 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

23 4.10.2.opvpTransferStreamData

24 **Name**

25 `opvpTransferStreamData` – send stream data

26 **Synopsis**

```
27 opvp_result_t opvpTransferStreamData(  
28     opvp_dc_t printerContext,  
29     opvp_int_t count,  
30     void *data);
```

31 **Arguments**

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

33 `count` – Number of data bytes to transfer

34 `data` – Pointer to the stream data

35 **Description**

36 This function sends the streaming data directly to the printer device.

37 **Return Value**

38 `OPVP_OK` or -1 when error, and the driver MUST store the detailed error code in `opvpErrorNo`.

39 4.10.3.opvpEndStream

40 **Name**

41 `opvpEndStream` – Declare the end of streaming data transfer

1 **Synopsis**

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

4 **Arguments**

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

6 **Description**

7 This function terminates the streaming data transfer.

8 **Return Value**

9 OPVP_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.

5. Graphic Operation Fallback

1

2 Printer or printer driver may not support some operation functions, and driver must notice its supported API to caller through the
3 `opvpOpenPrinter()` function. For instance, when driver does not support Path Operation functions, driver set `NULL` into the
4 `apiEntry` members (See the definition of the `opvpOpenPrinter()` function) that reserved to store the pointer to the Path
5 Operation functions, such as rectangle function, of the driver. Then, caller checks each element if it is `NULL`, and caller knows the
6 Path Operation functions are not supported by the driver. After that, to draw a rectangle, caller may break the rectangle drawing
7 operation down into other drawing functions, for instance, drawing bitmaps that cover the rectangle. This type of implementation
8 that caller calls the alternative drawing operations is “Caller Fallback”.

9 In another case, printer device does not support some drawing operation functions but printer driver prepares the alternative
10 drawing functions in stead of it. For instance, when printer device does not support Path Operation functions, but driver may
11 prepare the rectangle drawing function that breaks its drawing operation down into other drawing functions prepared by printer
12 device. In this case, driver sets the pointer to the rectangle function into the `apiEntry` member, and caller can call the drawing
13 rectangle function. This type of implementation that prepares the alternative drawing functions is “Driver Fallback”.

14 This specification does not cover the specification of both “Caller Fallback” and “Driver Fallback”, because this specification
15 covers the API that driver prepares, and fallback is not the API. “Caller Fallback” is the specification of how caller calls the
16 alternative API when some API are not supported by driver, and the “Caller Fallback” specification depends on each caller
17 implementation. “Driver Fallback” is also not the API, and caller can not know if driver prepares drawing functions with or
18 without fallback because driver conceal the fallback process inside the driver.

19

20

6. Macros, Types, Enums and Structures

6.1. Return Values

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

6.2. 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 */
#define OPVP_JOBANCELED -5     /* error: job has been canceled by some cause */
#define OPVP_PARAMERROR  -6    /* error: invalid parameter */
```

6.3. 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 unsigned int opvp_rop_t; /* raster operation */

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;
```

6.4. Image Formats

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

6.5. Color Presentation

```
typedef enum _opvp_colormapping {
    OPVP_CMAP_DIRECT      = 0,
    OPVP_CMAP_INDEXED     = 1
} opvp_colormapping_t;

typedef enum _opvp_cspace {
    OPVP_CSPACE_BW        = 0,
    OPVP_CSPACE_DEVICEGRAY = 1,
    OPVP_CSPACE_DEVICECMY = 2,
    OPVP_CSPACE_DEVICECMYK = 3,
}
```

```

1         OPVP_CSPACE_DEVICERGB      = 4,
2         OPVP_CSPACE_DEVICEKRGB     = 5,
3         OPVP_CSPACE_STANDARDRGB    = 6,
4         OPVP_CSPACE_STANDARDRGB64  = 7
5     } opvp_cspace_t;

```

1 6.6.Fill, Paint, Clip

```

2     typedef enum _opvp_fillmode {
3         OPVP_FILLMODE_EVENODD      = 0,
4         OPVP_FILLMODE_WINDING      = 1
5     } opvp_fillmode_t;
6
7     typedef enum _opvp_paintmode {
8         OPVP_PAINTMODE_OPAQUE      = 0,
9         OPVP_PAINTMODE_TRANSPARENT = 1
10    } opvp_paintmode_t;
11
12    typedef enum _opvp_cliprule {
13        OPVP_CLIPRULE_EVENODD       = 0,
14        OPVP_CLIPRULE_WINDING       = 1
15    } opvp_cliprule_t;

```

1 6.7.Line

```

2     typedef enum _opvp_linestyle {
3         OPVP_LINESTYLE_SOLID       = 0,
4         OPVP_LINESTYLE_DASH        = 1
5     } opvp_linestyle_t;
6
7     typedef enum _opvp_linecap {
8         OPVP_LINECAP_BUTT          = 0,
9         OPVP_LINECAP_ROUND         = 1,
10        OPVP_LINECAP_SQUARE        = 2
11    } opvp_linecap_t;
12
13    typedef enum _opvp_linejoin {
14        OPVP_LINEJOIN_MITER         = 0,
15        OPVP_LINEJOIN_ROUND         = 1,
16        OPVP_LINEJOIN_BEVEL        = 2
17    } opvp_linejoin_t;

```

1 6.8.Brush

```

2     typedef struct _opvp_brushdata {
3         opvp_bdtype_t type;
4         opvp_int_t width, height, pitch;
5         opvp_byte_t data[]; /* must be defined as data[1] for GCC 2.x */
6
7     } opvp_brushdata_t;
8
9     typedef struct _opvp_brush {
10        opvp_cspace_t colorSpace;
11        opvp_int_t color[4]; /* コメント削除
12        opvp_int_t xorg, yorg; /* brush origin , ignored for opvpSetBgColor */
13        opvp_brushdata_t *pbrush; /* pointer to brush data */
14    } opvp_brush_t;
15
16    typedef enum _opvp_bdtype {OPVP_BDTYPE_NORMAL = 0} opvp_bdtype_t;

```

1 6.9.Misc. Flags

```

2     typedef enum _opvp_arcmode {
3         OPVP_ARC                    = 0,
4         OPVP_CHORD                   = 1,
5         OPVP_PIE                      = 2

```

```

1   } opvp_arcmode_t;
2
3   typedef enum _opvp_arcdir {
4       OPVP_CLOCKWISE           = 0,
5       OPVP_COUNTERCLOCKWISE   = 1
6   } opvp_arcdir_t;
7
8   typedef enum _opvp_pathmode {
9       OPVP_PATHCLOSE           = 0,
10      OPVP_PATHOPEN            = 1
11   } opvp_pathmode_t;

```

1 6.10.CTM

```

2   typedef struct _opvp_ctm {
3       opvp_float_t a, b, c, d, e, f;
4   } opvp_ctm_t;

```

1 6.11.Device Information and Capabilities

```

2   typedef enum _opvp_queryinfoflags {
3       OPVP_QF_DEVICERESOLUTION = 0x00000001,
4       OPVP_QF_MEDIASIZE        = 0x00000002,
5       OPVP_QF_PAGEROTATION     = 0x00000004,
6       OPVP_QF_MEDIANUP         = 0x00000008,
7       OPVP_QF_MEDIADUPLEX      = 0x00000010,
8       OPVP_QF_MEDIASOURCE      = 0x00000020,
9       OPVP_QF_MEDIADESTINATION = 0x00000040,
10      OPVP_QF_MEDIATYPE         = 0x00000080,
11
12      OPVP_QF_MEDIACOPY          = 0x00000100, /* Maximum copy number supported */
13      OPVP_QF_PRINTREGION       = 0x00010000 /* only for opvpQueryDeviceInfo use */
14   } opvp_queryinfoflags_t;

```

1 6.12.Sample Header File

2 Here is a sample header file which can be used by driver or render based on this specification.

```

3
4   #ifndef _OPVP_H_
5   #define _OPVP_H_
6
7   /* Return Values and Error Codes */
8   #define OPVP_OK           0       /* -1 for errors */
9   #define OPVP_FATALERROR  -1      /* error: cannot be recovered */
10  #define OPVP_BADREQUEST  -2      /* error: called where it should not be called */
11  #define OPVP_BADCONTEXT  -3      /* error: invalid printer context */
12  #define OPVP_NOTSUPPORTED -4     /* error: combination of parameters are set */
13                                     /* which cannot be handled by driver or printer */
14  #define OPVP_JOBCANCELED -5     /* error: job has been canceled by some cause */
15  #define OPVP_PARAMERROR  -6     /* error: invalid parameter */
16
17  /* Basic Types */
18  typedef int opvp_dc_t;           /* driver/device context */
19  typedef int opvp_result_t;       /* return value */
20  typedef unsigned char opvp_byte_t; /* BYTE */
21  typedef unsigned char opvp_char_t; /* character (string) */
22  typedef int opvp_int_t;          /* integer */
23  typedef int opvp_fix_t;          /* fixed integer */
24  typedef float opvp_float_t;     /* float */
25  typedef unsigned int opvp_flag_t; /* flags */
26  typedef unsigned int opvp_rop_t; /* raster operation */
27
28  /* for opvp_fix_t */
29  #define OPVP_FIX_FRACT_WIDTH 8
30  #define OPVP_FIX_FRACT_DENOM (1<<OPVP_FIX_FRACT_WIDTH)
31  #define OPVP_FIX_FLOOR_WIDTH (sizeof(int)*8-OPVP_FIX_FRACT_WIDTH)

```



```

1  /* convert macro */
2  #define      OPVP_I2FIX(i,fix)      (fix=i<<OPVP_FIX_FRACT_WIDTH)
3  #define      OPVP_F2FIX(f,fix)      (fix=((int)floor(f)<<OPVP_FIX_FRACT_WIDTH)\
4                                          |((int)((f-floor(f))*OPVP_FIX_FRACT_DENOM)\
5                                          &(OPVP_FIX_FRACT_DENOM-1)))
6
7  /* graphic elements */
8  typedef struct _opvp_point {
9      opvp_fix_t x, y;
10 } opvp_point_t;
11
12 typedef struct _opvp_rectangle {
13     opvp_point_t p0;          /* start point */
14     opvp_point_t p1;          /* diagonal point */
15 } opvp_rectangle_t;
16
17 typedef struct _opvp_roundrectangle {
18     opvp_point_t p0;          /* start point */
19     opvp_point_t p1;          /* diagonal point */
20     opvp_fix_t xellipse, yellipse;
21 } opvp_roundrectangle_t;
22
23 /* Image Formats */
24 typedef enum _opvp_imageformat {
25     OPVP_IFORMAT_RAW          = 0,
26     OPVP_IFORMAT_RLE          = 1,
27     OPVP_IFORMAT_JPEG         = 2,
28     OPVP_IFORMAT_PNG          = 3
29 } opvp_imageformat_t;
30
31 /* Color Presentation */
32 typedef enum _opvp_colormapping {
33     OPVP_CMAP_DIRECT          = 0,
34     OPVP_CMAP_INDEXED         = 1
35 } opvp_colormapping_t;
36
37 typedef enum _opvp_cspace {
38     OPVP_CSPACE_BW            = 0,
39     OPVP_CSPACE_DEVICEGRAY    = 1,
40     OPVP_CSPACE_DEVICECMY     = 2,
41     OPVP_CSPACE_DEVICECMYK    = 3,
42     OPVP_CSPACE_DEVICERGB     = 4,
43     OPVP_CSPACE_DEVICEKRGB    = 5,
44     OPVP_CSPACE_STANDARDRGB   = 6,
45     OPVP_CSPACE_STANDARDRGB64 = 7
46 } opvp_cspace_t;
47
48 /* Fill, Paint, Clip */
49 typedef enum _opvp_fillmode {
50     OPVP_FILLMODE_EVENODD     = 0,
51     OPVP_FILLMODE_WINDING     = 1
52 } opvp_fillmode_t;
53
54 typedef enum _opvp_paintmode {
55     OPVP_PAINTMODE_OPAQUE      = 0,
56     OPVP_PAINTMODE_TRANSPARENT = 1
57 } opvp_paintmode_t;
58
59 typedef enum _opvp_cliprule {
60     OPVP_CLIPRULE_EVENODD     = 0,
61     OPVP_CLIPRULE_WINDING     = 1
62 } opvp_cliprule_t;
63
64 /* Line */
65 typedef enum _opvp_linestyle {
66     OPVP_LINESTYLE_SOLID      = 0,
67     OPVP_LINESTYLE_DASH       = 1
68 } opvp_linestyle_t;
69
70 typedef enum _opvp_linecap {
71     OPVP_LINECAP_BUTT         = 0,

```

```

1         OPVP_LINECAP_ROUND           = 1,
2         OPVP_LINECAP_SQUARE         = 2
3     } opvp_linecap_t;
4
5     typedef enum _opvp_linejoin {
6         OPVP_LINEJOIN_MITER         = 0,
7         OPVP_LINEJOIN_ROUND         = 1,
8         OPVP_LINEJOIN_BEVEL         = 2
9     } opvp_linejoin_t;
10
11     /* Brush */
12     typedef enum _opvp_bdtype {
13         OPVP_BDTYPE_NORMAL           = 0
14     } opvp_bdtype_t;
15
16     typedef struct _opvp_brushdata {
17         opvp_bdtype_t type;
18         opvp_int_t width, height, pitch;
19 #if defined(__GNUC__) && __GNUC__ <= 2
20         opvp_byte_t data[1];
21 #else
22         opvp_byte_t data[];
23 #endif
24     } opvp_brushdata_t;
25
26     typedef struct _opvp_brush {
27         opvp_cspace_t colorSpace;
28         opvp_int_t color[4]; /* コメント削除
29         opvp_int_t xorg, yorg; /* brush origin */
30                                     /* ignored for opvpSetBgColor */
31         opvp_brushdata_t *pbrush; /* pointer to brush data */
32                                     /* solid brush used, if NULL */
33     } opvp_brush_t;
34
35     /* Misc. Flags */
36     typedef enum _opvp_arcmode {
37         OPVP_ARC                       = 0,
38         OPVP_CHORD                       = 1,
39         OPVP_PIE                         = 2
40     } opvp_arcmode_t;
41
42     typedef enum _opvp_arcdir {
43         OPVP_CLOCKWISE                   = 0,
44         OPVP_COUNTERCLOCKWISE           = 1
45     } opvp_arcdir_t;
46
47     typedef enum _opvp_pathmode {
48         OPVP_PATHCLOSE                   = 0,
49         OPVP_PATHOPEN                   = 1
50     } opvp_pathmode_t;
51
52     /* CTM */
53     typedef struct _opvp_ctm {
54         opvp_float_t a, b, c, d, e, f;
55     } opvp_ctm_t;
56
57     /* Device Information and Capabilites */
58     typedef enum _opvp_queryinfoflags {
59         OPVP_QF_DEVICERESOLUTION = 0x00000001,
60         OPVP_QF_MEDIASIZE         = 0x00000002,
61         OPVP_QF_PAGEROTATION     = 0x00000004,
62         OPVP_QF_MEDIANUP         = 0x00000008,
63         OPVP_QF_MEDIADUPLEX     = 0x00000010,
64         OPVP_QF_MEDIASOURCE     = 0x00000020,
65         OPVP_QF_MEDIADESTINATION = 0x00000040,
66         OPVP_QF_MEDIATYPE       = 0x00000080,
67
68         OPVP_QF_MEDIACOPY         = 0x00000100, /* Maximum copy number supported */
69         OPVP_QF_PRINTREGION       = 0x00010000 /* only for opvpQueryDeviceInfo use */
70     } opvp_queryinfoflags_t;
71
72

```

```

1  /* API Procedure Entries */
2  typedef      struct _opvp_api_procs {
3      opvp_dc_t      (*opvpOpenPrinter) (opvp_int_t,opvp_char_t*,opvp_int_t,struct
4  _opvp_api_procs**);※引数削除
5      opvp_result_t  (*opvpClosePrinter) (opvp_dc_t);
6      opvp_result_t  (*opvpStartJob) (opvp_dc_t,opvp_char_t*);
7      opvp_result_t  (*opvpEndJob) (opvp_dc_t);
8      opvp_result_t  (*opvpAbortJob) (opvp_dc_t);
9      opvp_result_t  (*opvpStartDoc) (opvp_dc_t,opvp_char_t*);
10     opvp_result_t  (*opvpEndDoc) (opvp_dc_t);
11     opvp_result_t  (*opvpStartPage) (opvp_dc_t,opvp_char_t*);
12     opvp_result_t  (*opvpEndPage) (opvp_dc_t);
13     opvp_result_t  (*opvpQueryDeviceCapability) (opvp_dc_t,opvp_flag_t,opvp_int_t,opvp_byte_t*);
14     opvp_result_t
15     (*opvpQueryDeviceInfo) (opvp_dc_t,opvp_flag_t,opvp_int_t,opvp_char_t*);
16     opvp_result_t  (*opvpResetCTM) (opvp_dc_t);
17     opvp_result_t  (*opvpSetCTM) (opvp_dc_t,opvp_ctm_t*);
18     opvp_result_t  (*opvpGetCTM) (opvp_dc_t,opvp_ctm_t*);
19     opvp_result_t  (*opvpInitGS) (opvp_dc_t);
20     opvp_result_t  (*opvpSaveGS) (opvp_dc_t);
21     opvp_result_t  (*opvpRestoreGS) (opvp_dc_t);
22
23     opvp_result_t  (*opvpQueryColorSpace) (opvp_dc_t,opvp_int_t*, opvp_cspace_t*);※
24     引数順番変更
25     opvp_result_t  (*opvpSetColorSpace) (opvp_dc_t,opvp_cspace_t);
26     opvp_result_t  (*opvpGetColorSpace) (opvp_dc_t,opvp_cspace_t*);
27     opvp_result_t  (*opvpQueryROP) (opvp_dc_t,opvp_int_t*,opvp_rop_t*);
28     opvp_result_t  (*opvpSetROP) (opvp_dc_t,opvp_rop_t);
29     opvp_result_t  (*opvpGetROP) (opvp_dc_t,opvp_rop_t*);
30     opvp_result_t  (*opvpGetFillMode) (opvp_dc_t,opvp_fillmode_t);
31     opvp_result_t  (*opvpGetFillMode) (opvp_dc_t,opvp_fillmode_t*);
32     opvp_result_t  (*opvpSetAlphaConstant) (opvp_dc_t,opvp_float_t);
33     opvp_result_t  (*opvpGetAlphaConstant) (opvp_dc_t,opvp_float_t*);
34     opvp_result_t  (*opvpSetLineWidth) (opvp_dc_t,opvp_fix_t);
35     opvp_result_t  (*opvpGetLineWidth) (opvp_dc_t,opvp_fix_t*);
36     opvp_result_t  (*opvpSetLineDash) (opvp_dc_t,opvp_fix_t*,opvp_int_t);
37     opvp_result_t  (*opvpGetLineDash) (opvp_dc_t,opvp_fix_t*,opvp_int_t*);
38     opvp_result_t  (*opvpSetLineDashOffset) (opvp_dc_t,opvp_fix_t);
39     opvp_result_t  (*opvpGetLineDashOffset) (opvp_dc_t,opvp_fix_t*);
40     opvp_result_t  (*opvpSetLineStyle) (opvp_dc_t,opvp_linestyle_t);
41     opvp_result_t  (*opvpGetLineStyle) (opvp_dc_t,opvp_linestyle_t*);
42     opvp_result_t  (*opvpSetLineCap) (opvp_dc_t,opvp_linecap_t);
43     opvp_result_t  (*opvpGetLineCap) (opvp_dc_t,opvp_linecap_t*);
44     opvp_result_t  (*opvpSetLineJoin) (opvp_dc_t,opvp_linejoin_t);
45     opvp_result_t  (*opvpGetLineJoin) (opvp_dc_t,opvp_linejoin_t*);
46     opvp_result_t  (*opvpSetMiterLimit) (opvp_dc_t,opvp_fix_t);
47     opvp_result_t  (*opvpGetMiterLimit) (opvp_dc_t,opvp_fix_t*);
48     opvp_result_t  (*opvpSetPaintMode) (opvp_dc_t,opvp_paintmode_t);
49     opvp_result_t  (*opvpGetPaintMode) (opvp_dc_t,opvp_paintmode_t*);
50     opvp_result_t  (*opvpSetStrokeColor) (opvp_dc_t,opvp_brush_t*);
51     opvp_result_t  (*opvpSetFillColor) (opvp_dc_t,opvp_brush_t*);
52     opvp_result_t  (*opvpSetBgColor) (opvp_dc_t,opvp_brush_t*);
53     opvp_result_t  (*opvpNewPath) (opvp_dc_t);
54     opvp_result_t  (*opvpEndPath) (opvp_dc_t);
55     opvp_result_t  (*opvpStrokePath) (opvp_dc_t);
56     opvp_result_t  (*opvpFillPath) (opvp_dc_t);
57     opvp_result_t  (*opvpStrokeFillPath) (opvp_dc_t);
58     opvp_result_t  (*opvpSetClipPath) (opvp_dc_t,opvp_cliprule_t);
59     opvp_result_t  (*opvpResetClipPath) (opvp_dc_t);
60     opvp_result_t  (*opvpSetCurrentPoint) (opvp_dc_t,opvp_fix_t,opvp_fix_t);
61     opvp_result_t
62     (*opvpLinePath) (opvp_dc_t,opvp_pathmode_t,opvp_int_t,opvp_point_t*);
63     opvp_result_t
64     (*opvpPolygonPath) (opvp_dc_t,opvp_int_t,opvp_int_t*,opvp_point_t*);
65     opvp_result_t  (*opvpRectanglePath) (opvp_dc_t,opvp_int_t,opvp_rectangle_t*);
66     opvp_result_t
67     (*opvpRoundRectanglePath) (opvp_dc_t,opvp_int_t,opvp_roundrectangle_t*);
68     opvp_result_t  (*opvpBezierPath) (opvp_dc_t,opvp_int_t,opvp_point_t*);
69     opvp_result_t
70     (*opvpArcPath) (opvp_dc_t,opvp_arcmode_t,opvp_arcdir_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,
71     opvp_fix_t,opvp_fix_t,opvp_fix_t);

```

```

1  t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t);
2  opvp_result_t
3  (*opvpDrawImage)(opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imageform
4  at_t,opvp_rectangle_t,void*);
5  opvp_result_t
6  (*opvpStartDrawImage)(opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imag
7  eformat_t,opvp_rectangle_t);
8  opvp_result_t (*opvpTransferDrawImage)(opvp_dc_t,opvp_int_t,void*);
9  opvp_result_t (*opvpEndDrawImage)(opvp_dc_t);
10 opvp_result_t (*opvpStartScanline)(opvp_dc_t,opvp_int_t);
11 opvp_result_t (*opvpScanline)(opvp_dc_t,opvp_int_t,opvp_int_t*);
12 opvp_result_t (*opvpEndScanline)(opvp_dc_t);
13 opvp_result_t (*opvpStartRaster)(opvp_dc_t,opvp_int_t);
14 opvp_result_t (*opvpTransferRasterData)(opvp_dc_t,opvp_int_t,opvp_byte_t*);
15 opvp_result_t (*opvpSkipRaster)(opvp_dc_t,opvp_int_t);
16 opvp_result_t (*opvpEndRaster)(opvp_dc_t);
17 opvp_result_t (*opvpStartStream)(opvp_dc_t);
18 opvp_result_t (*opvpTransferStreamData)(opvp_dc_t,opvp_int_t,void*);
19 opvp_result_t (*opvpEndStream)(opvp_dc_t);
20 } opvp_api_procs_t;
21
22 /* Function prototype */
23 opvp_dc_t opvpOpenPrinter(
24     opvp_int_t outputFD,
25     opvp_char_t *printerModel,
26     opvp_int_t apiVersion[2],
27     opvp_int_t *nApiEntry,
28     opvp_api_procs_t **apiProcs);
29
30 /* error no */
31 extern opvp_int_t opvpErrorNo;
32
33 #endif /* _OPVP_H_ */

```

7. Authors and Contributors

1

1 7.1. Editors

2 Osamu Mihara – Fuji Xerox Co., Ltd.

1 7.2. Authors

2 Osamu Mihara – Fuji Xerox Co., Ltd.

3 Yasumasa Toratani – Canon Inc.

1 7.3. Contributors

2 (Alphabetical order) Hidekazu Hagiwara (MintWave), Masaki Iwata (AXE), Hidenori Kanjo (BBR), Shinpei Kitayama
3 (EPSON Avasys), ★オラフさんを追加しても良いかオラフさんに確認 Kenichi Maeda (E&D), Akeo Maruyama (Ricoh), ★小
4 笠原さん追加しても良いか小笠原さんに確認 Hisao Nakamura (E&D), Koji Otani (BBR), Kenji Wakabayashi (MintWave),
5 Toshihiro Yamagishi (Turbolinux), Akira Yoshiyama (NEC)

- 1
- 2 Version 0.1 (Japanese) Mihara/Toratani
- 3 Version 0.2 (Japanese) Mihara
- 4 Openprinting-0.1.1 by opfc have implemented based on this version
- 5 Version 0.2-en 2004-3-14 Mihara/Toratani/Kitayama/Yamagishi/Kanjo
- 6 Translation to English
- 7 Some feedback from opfc implementation
- 8 Version 1.0 RC1 2005-7-20 Mihara
- 9 Change copyright notice from FDL to MIT style copyright
- 10 Delete temporary font operations.
- 11 Version 1.0 RC2 2006-6-9/2006-6-17/2006-6-20/2006-6-26/2006-6-29
- 12 Add API version number to `OpenPrinter()` parameter
- 13 Add pitch and delete count to/from `DrawImage()/StartDrawImage()`
- 14 Add a figure for `SetMiterLimit`.
- 15 Change function names/constants/enum/structures names to add FSG prefixes.
- 16 Change copyright year and dates
- 17 Delete "Printer Driver Database"
- 18 Delete *cmap* from CSPASE chart.
- 19 Fix parameter `colorSpace` (wrong) to `colorDepth` (correct) for `DrawImage()/StartDrawImage`
- 20 Change description for `fsgpdStartJob/fsgpdEndJob/fsgpdAbortJob`
- 21 Add type `FSGPD_CHAR` for character type
- 22 Version 1.0 RC2 2006-10-20
- 23 Add `FSGPD_CSPACE_DEVICEKRGB`
- 24 Remove "current implementation" descriptions from Color Scheme and Color Space descriptions.
- 25 Remove "described later" descriptions
- 26 Make support of updf for scheme "MUST."
- 27 Version 1.0 RC2 2006-12-05
- 28 Format: Add line numbers and chapter numbers.
- 29 Format: Replace Microsoft Expressions object to OOo Expressions
- 30 Version 1.0 RC3 2007-5-19 Toratani
- 31 Append "opvp" or "OPVP" to function names, enums, symbols according to the naming rules.
- 32 Update parameters of `opvpOpenPrinter()` function and the bitmap image functions.
- 33 Rearrange the document sections, chapters and text formant.
- 34 Merge the RC2 2006-6-9 – 2006-12-05 updates made by Mihara
- 35
- 36