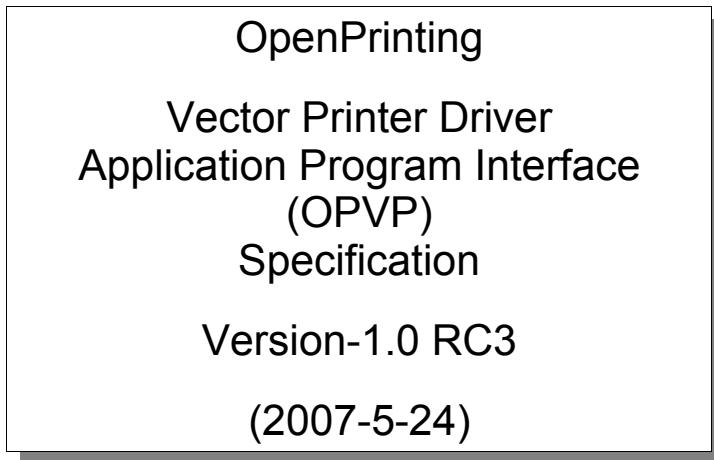


1 |  
2 |  
3  
4  
5  
6  
7  
8



9

# 1 | OpenPrinting Vector Printer Driver Application **Program** Interface Specification

2

3 | Copyright © 2007 OpenPrinting Work Group★Copyrightは何処のものになるのか？この通り。

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.Notational 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.....	10
3.6.Scan Rule.....	10
4.Operations.....	11
4.1.Creating and Managing Printer Contexts.....	11
4.1.1.opvpOpenPrinter.....	11
4.1.2.opvpClosePrinter.....	12
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.....	16
4.3.Query Operations.....	17
4.3.1.opvpQueryDeviceCapability.....	17
4.3.2.opvpQueryDeviceInfo.....	18
4.4.Attributes for Job, Document and Page Operations.....	20
4.5.Graphics State Object Operations.....	22
4.5.1.opvpResetCTM.....	22
4.5.2.opvpSetCTM.....	22
4.5.3.opvpGetCTM.....	23
4.5.4.opvpInitGS.....	23
4.5.5.opvpSaveGS.....	24
4.5.6.opvpRestoreGS.....	24
4.5.7.opvpQueryColorSpace.....	25
4.5.8.opvpSetColorSpace.....	25
4.5.9.opvpGetColorSpace.....	26
4.5.10.opvpQueryROP.....	26
4.5.11.opvpSetROP.....	27
4.5.12.opvpGetROP.....	27
4.5.13.opvpSetFillMode.....	28
4.5.14.opvpGetFillMode.....	28
4.5.15.opvpSetAlphaConstant.....	29
4.5.16.opvpGetAlphaConstant.....	29
4.5.17.opvpSetLineWidth.....	29

4.5.18.opvpGetLineWidth.....	30
4.5.19.opvpSetLineDash.....	30
4.5.20.opvpGetLineDash.....	31
4.5.21.opvpSetLineDashOffset.....	32
4.5.22.opvpGetLineDashOffset.....	32
4.5.23.opvpSetLineStyle.....	33
4.5.24.opvpGetLineStyle.....	33
4.5.25.opvpSetLineCap.....	33
4.5.26.opvpGetLineCap.....	34
4.5.27.opvpSetLineJoin.....	35
4.5.28.opvpGetLineJoin.....	35
4.5.29.opvpSetMiterLimit.....	36
4.5.30.opvpGetMiterLimit.....	36
4.5.31.opvpSetPaintMode.....	37
4.5.32.opvpGetPaintMode.....	37
4.5.33.opvpSetStrokeColor.....	37
4.5.34.opvpSetFillColor.....	38
4.5.35.opvpSetBgColor.....	39
4.6.Path Operations.....	40
4.6.1.opvpNewPath.....	40
4.6.2.opvpEndPath.....	40
4.6.3.opvpStrokePath.....	41
4.6.4.opvpFillPath.....	41
4.6.5.opvpStrokeFillPath.....	41
4.6.6.opvpSetClipPath.....	42
4.6.7.opvpResetClipPath.....	42
4.6.8.opvpSetCurrentPoint.....	43
4.6.9.opvpLinePath.....	43
4.6.10.opvpPolygonPath.....	44
4.6.11.opvpRectanglePath.....	44
4.6.12.opvpRoundRectanglePath.....	45
4.6.13.opvpBezierPath.....	46
4.6.14.opvpArcPath.....	46
4.7.Bitmap Image Operations.....	48
4.7.1.opvpDrawImage.....	48
4.7.2.opvpStartDrawImage.....	49
4.7.3.opvpTransferDrawImage.....	50
4.7.4.opvpEndDrawImage.....	51
4.8.Scan Line Operations.....	52
4.8.1.opvpStartScanline.....	52
4.8.2.opvpScanline.....	52
4.8.3.opvpEndScanline.....	53
4.9.Raster Image Operations.....	54
4.9.1.opvpStartRaster.....	54
4.9.2.opvpTransferRasterData.....	55
4.9.3.opvpSkipRaster.....	55
4.9.4.opvpEndRaster.....	56
4.10.Stream Data Operations.....	57
4.10.1.opvpStartStream.....	57
4.10.2.opvpTransferStreamData.....	57
4.10.3.opvpEndStream.....	58

5. Graphic Operation Fallback.....	59
6. Macros, Types, Enumerations and Structures.....	60
6.1.Return Values.....	60
6.2.Error Codes.....	60
6.3.Basic Types.....	60
6.4.Image Formats.....	60
6.5.Color Presentation.....	61
6.6.Fill, Paint, Clip.....	61
6.7.Line.....	61
6.8.Brush.....	62
6.9.Miscellaneous Flags.....	62
6.10.CTM.....	62
6.11.Device Information and Capabilities.....	63
6.12.Sample Header File.....	63
7. Authors and Contributors.....	69
7.1.Editors.....	69
7.2 AUTHORS.....	69
7.3 CONTRIBUTORS.....	69
8. History.....	70

# 1. Notation and Terminology

---

## 1.1. Notational Conventions

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

Font <sup>1</sup> and Style	Description	Examples
Courier	Definition of functions, structures, enumerations and constants.	<pre>opvp_result_t opvpClosePrinter(     opvp_dc_t printerContext);</pre> <pre>typedef struct _opvp_point {     opvp_fix_t x, y; } opvp_point_t;</pre> <pre>#define OPVP_OK      0</pre>
	Function parameters <sup>2</sup>	printerContext
	Source code examples.	<pre>#ifndef _OPVP_H_ #define _OPVP_H_</pre>
<i>Italic</i>	Coordinate values (x, y)	(x0, y0)

3

## 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].<sup>3</sup>

4

### 2.1. Driver and Caller

This specification defines an Application Program Interface (API) which is used in the printing environment offered by OpenPrinting. In this document, the word “printer driver” or “driver” refers to a software library. The driver translates document data generated by application programs into a 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 renderer 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 renderer, printer driver based on this specification does not depend on any specific type of renderer.

14

### 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.

15

### 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.

4

### 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.

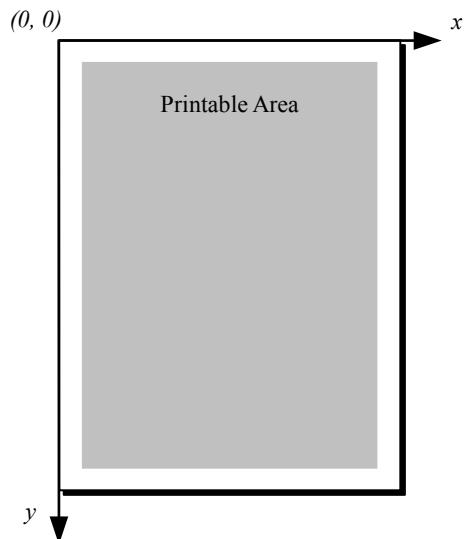
5

## 3.1. Coordinate System

The origin of the coordinate system ~~as which is~~ defined and used in this ~~documentspecification~~ is ~~the~~ physical upper left corner of ~~the~~ media handled by ~~each~~ device. ~~The~~ Units of the coordinate system ~~isare~~ based on ~~each~~ the device resolution. Positive direction on ~~the~~ x-axis is from the origin ~~towards~~ the right ~~direction outside of~~ the media, and positive direction on ~~the~~ y-axis is from the origin ~~towards~~ the ~~lower direction on bottom~~ of the media.

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

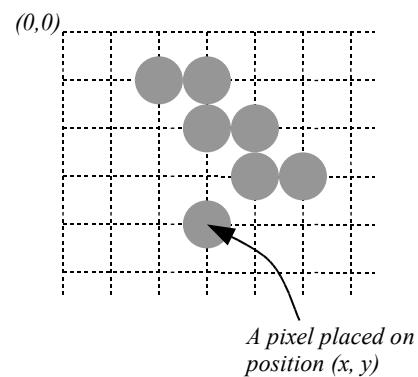
Coordinate values ~~s~~ x and ~~y~~ define ~~each~~ the horizontal and vertical distance from the origin to ~~the~~ a point on the media. ~~The~~ A point ~~which has the with~~ integer coordinate values ~~s~~ x and ~~y~~ is on the intersection of the coordinate system grid ~~, and p.~~ Physical device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



## 3.2. Objects

~~F~~The following kinds of graphics objects are handled in this document:

- Paths
- Bitmap Images
- Scan Lines
- Raster Image
- ★Text 削除。削除する。



### 1 3.3.Graphics State Object

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

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

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

9

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

10

### 1 3.4.CTM

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

$$5 \begin{bmatrix} x_{dev} & y_{dev} & 1 \end{bmatrix} = \begin{bmatrix} x_{ren} & y_{ren} & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

6 To set or get, set or reset a CTM from the Graphics State Object, refer the description of use the opvpResetGetCTM(),  
7 opvpSetCTM(), opvpGetResetCTM() functions.

### 1 3.5.Color

2 The following color spaces are defined in this document.

Color Space Macro	Color Space
OPVP_CSPACE_BW	Black and White
OPVP_CSPACE_DEVICEGRAY	Grayscale
OPVP_CSPACE_DEVICECMY	CMY
OPVP_CSPACE_DEVICECMYK	CMY and Black
OPVP_CSPACE_DEVICERGB	Device RGB
OPVP_CSPACE_DEVICEKRGB	Device KRGB★必要?必要である。
OPVP_CSPACE_STANDARDRGB	sRGB
OPVP_CSPACE_STANDARDRGB64	scRGB

1  
2 | Only OPVP\_CSPACE\_BW, OPVP\_CSPACE\_DEVICEGRAY and OPVP\_CSPACE\_STANDARDRGB should need to be supported  
3 | by printer driver which implements 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**  
3 | **method, “right-bottom exclusive method” can be used in a driver, however, the painting method implementation of each**  
4 | **printer driver depends on each page description language of each printer. Therefore, this document does not specify the**  
5 | **painting method implementation.**

## 1 | 4.1.Creating and Managing Printer Contexts

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

### 3 | 4.1.1.opvpOpenPrinter

4 | **Name**

5 | opvpOpenPrinter – Creates a printer context.

6 | **Synopsis**

```
7 |     opvp_dc_t opvpOpenPrinter(
8 |         opvp_int_t outputFD,
9 |         opvp_char_t *printerModel,
10 |         opvp_int_t apiVersion[2],
11 |         opvp_int_t *nApiEntry, ★この引数は必要なのか？必要。caller側がチェックする MUST。
12 |         struct _opvp_api_procs **apiEntry); ★この書き方であつてはいるか？何故2重ポインタ？
```

13 | **Arguments**

14 | outputFD – File descriptor to write the printing data stream to.

15 | printerModel – Printer Model Name in (UTF-8 encoded).

16 | ~~apiVersion – Vector of printer driver specification version number which supported by the driver supports.~~  
17 | ~~A apiVersion[0] is the major and apiVersion[1] is the minor value of the version respectively.~~  
18 | ~~nApiEntry – Number of apiEntry array elements.~~

19 |  
20 | apiEntry – Pointer to a structure which stores all API entries of the driver.

21 | **Description**

22 | This function initializes the driver. The caller MUST specify the file descriptor for writing the printing data stream, and the  
23 | driver ~~MUST write generates~~ the printing data stream and ~~MUST write~~ it ~~generates~~ into the file descriptor. The caller may  
24 | specify the a UTF-8 encoded printer model name by via printerModel – in UFT-8. If the caller ~~setspasses~~ NULL in for  
25 | printerModel, the driver SHOULD use the default printer model of the driver. PThe printer model name should be defined  
26 | in the printer model data base.

27 | The driver may write its debugging messages into stderr. Therefore stderr MUST not be given for passed as  
28 | outputFD by the caller.

29 | The driver MUST allocate the struct \_opvp\_api\_procs buffer and store the address of each driver API entry address  
30 | of the driver into each to corresponding member of apiEntry. If the driver does not prepare support some API entries, the  
31 | driver MUST store NULL into the corresponding apiEntry members. ドライバ側がエリアを確保して返す。

32 |  
33 | Driver MUST give the number of the apiEntry members in nApiEntry. ★この引数は必要？必要。ドライバ側でサイズ  
34 | チェックする。MUST。何故ポインタなのか？ドライバが必要なサイズを返す？ Only this function MUST be exported by the  
35 | driver library to a caller which links with the driver library. The caller MUST refer and call each API entry via the  
36 | addresses stored in apiEntry.

37 | The printer driver MUST return a printer context as the return value of this function. The printer context MUST be a unique  
38 | number which is managed by the driver. The caller setspasses the printer context a to the first parameter argument to when  
39 | calling other API entries.

40 | **Return Value**

41 | Printer context value (positive value) or -1 in case of when error. In the later case, the and driver MUST store at the detailed error  
42 | code in error = opvpErrorNo.

## 1    4.1.2.opvpClosePrinter

### 2    **Name**

3 |    opvpClosePrinter – Deletes sa printer context.

### 4    **Synopsis**

5 |    opvp\_result\_t opvpClosePrinter(  
6 |                opvp\_dc\_t printerContext);

### 7    **Arguments**

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

### 9    **Description**

10 |    This function terminates the printing process and deletes the printer context kept in from the printer driver. The caller MUST  
11 |    close the file descriptor that the caller gave as the was passed as outputFD to for the opvpOpenPrinter() function.

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

### 16    **Return Value**

17 |    OPVP\_OK or -1 in case of when error. In the latter case, -and the driver MUST store the a detailed error code in  
18 |    errorneopvpErrorNo.

## 1 4.2.Job, Document and Page Operations

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

### 9 4.2.1.opvpStartJob

#### 10 Name

11 | opvpStartJob – Declare to the start of a printing job.

#### 12 Synopsis

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

#### 16 Arguments

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

18 | jobInfo – Printing job property string.

19

#### 20 Description

21 | This function declares to the start of a printing job.

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

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

28 | If the caller passeses NULL for the jobInfo, the printer driver SHOULD use its default printing job properties. The  
29 | dData format of -jobInfo is described later in the section "Attribute of Job, Document and Page Operations" in this  
30 | document.

31 | It depends on printer and driver capabilities whether nesting printing jobs are supported. A nested printing job is one where  
32 | the (opvpStartJob() and opvpEndJob() functions can be called between an enclosing pair of the  
33 | opvpStartJob() and opvpEndJob() functions are called) can be handled. If a printer or printer driver does not  
34 | allow support nesting printing jobs, the driver MUST return an error and set the detailed error code to OPVP\_BADREQUEST  
35 | in opvpErrorNo.BADREQUESTにセット? errno.★何を

#### 36 Return Value

37 | OPVP\_OK or -1 in case ofwhen error. In the latter case, and the driver MUST store the detailed error code in  
38 | errorneopvpErrorNo.

### 39 4.2.2.opvpEndJob

#### 40 Name

41 | opvpEndJob – Declare to terminates the end of a printing job.

#### 42 Synopsis

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

#### 45 Arguments

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

1 | **Description**  
 2 | This function declares to terminate the end of a the printing job.  
 3 | The caller MUST call this function after it finishes processing eacha printing job processing.  
  
 4 | **Return Value**  
 5 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
 6 | errorneopvpErrorNo.  
  
 7 | **4.2.3.opvpAbortJob**

8 | **4.2.3.1.Name**  
 9 | fsgpdAbortJob – Declare to abort the printing job.  
  
 10 | **Synopsis**  
 11 | opvp\_result\_t fsgpdAbortJob(  
 12 |     opvp\_dc\_t printerContext);  
  
 13 | **Arguments**  
 14 | printerContext – Printer context value returned by opvpOpenPrinter() function.  
  
 15 | **Description**  
 16 | This function cleans up the printing operations to abort the print job. Driver MAY create a print data to clean up the printer  
 17 | printing status and MAY send the data to the printer. However, because whether driver sends the printing job data to the printer  
 18 | before sending the aborting data depends on the timing when caller calls this function, so calling this function does not  
 19 | guarantee that no papers are wasted by the printer. However, after caller calls this function, driver and printer MUST become in  
 20 | the initial state and next printing job MUST be accepted normally.★文章を変更、これでよいか?  
 21 | ★EndJobは呼ぶ必要は無いのか?明記する必要あり  
  
 22 | **Return Value**  
 23 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in opvpErrorNo.  
  
 24 | **4.2.4.opvpStartDoc**

25 | **Name**  
 26 | opvpStartDoc – Declares to the start of a printing document.  
  
 27 | **Synopsis**  
 28 | opvp\_result\_t opvpStartDoc(  
 29 |     opvp\_dc\_t printerContext,  
 30 |     opvp\_char\_t \*docInfo);  
  
 31 | **Arguments**  
 32 | printerContext – Printer context value returned by the opvpOpenPrinter() function.  
 33 | docInfo – Printing document property string.  
  
 34 | **Description**  
 35 | This function declares to the start of a printing document.  
 36 | The caller should call this function after calling the opvpStartJob() function, and before calling the  
 37 | opvpStartPage() function or any and other drawing API entries. However, if a printing job consists of only one document,  
 38 | the caller can omit calling the opvpStartDoc() and opvpEndDoc() functions.  
 39 | The caller can set document properties into via jobInfo. P The printer driver MUST keep the given document properties  
 40 | until the opvpEndDoc() function is called. When the caller calls the opvpStartDoc() function again after calling the  
 41 | opvpEndDoc() function, the driver MUST override the former docInfo by with the new docInfo given specified by the  
 42 | latest opvpStartDoc() function call.  
 43 | If the caller sets passes NULL to for the docInfo, the printer driver SHOULD use its default printing document properties. The  
 44 | dData format of -docInfo is described later in the section "Attributes of Job, Document and Page Operations" in this  
 45 | document.

1 It depends on printer and driver capabilities whether nested documents are supported. A nested document is one where the  
2 `opvpStartDoc()` and `opvpEndDoc()` functions can be called between the enclosing pair of `opvpStartDoc()`  
3 and `opvpEndDoc()` functions can be handled. If a printer or printer driver does not allow support for nested  
4 documents, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST` in  
5 `opvpErrorNo.BADREQUEST`に何をセット? errno:★

7 **Return Value**

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

## 10 4.2.5.`opvpEndDoc`

11 **Name**

12 `opvpEndDoc` – Declares the end of a to terminate the printing document

13 **Synopsis**

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

16 **Arguments**

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

18 **Description**

19 This function declares the end of a to terminate the printing document.

20 If the caller called the `opvpStartDoc()` function, the caller MUST call this function after it finishes processing each  
21 document-processing.

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

23 **Return Value**

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

## 26 4.2.6.`opvpStartPage`

27 **Name**

28 `opvpStartPage` – Declares to the start of a printing page.

29 **Synopsis**

```
30     opvp_result_t opvpStartPage(  
31         opvp_dc_t printerContext,  
32         opvp_char_t *pageInfo);
```

33 **Arguments**

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

35 `pageInfo` – Printing page property string.

36 **Description**

37 This function declares to the start of a printing page.

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

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

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

1    **Return Value**  
2 |    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
3 |    errorneopvpErrorNo.

## 4    **4.2.7.opvpEndPage**

5    **Name**  
6 |    opvpEndPage – Declares to terminate the end of a printing page.

7    **Synopsis**  
8 |    opvp\_result\_t opvpEndPage(  
9 |         opvp\_dc\_t printerContext);

10   **Arguments**  
11 |    printerContext – Printer context value returned by the opvpOpenPrinter() function.

12   **Description**  
13 |    This function declares to terminate the end of a printing page.  
14 |    The caller MUST call this function after it finishes processing each page processing.

15   **Return Value**  
16 |    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
17 |    errorneopvpErrorNo.

18

## 1 4.3.Query Operations

2 | This section defines functions to query device capabilities and information. Driver support for these functions are optional.

### 4 4.3.1.opvpQueryDeviceCapability

#### 5 Name

6 | opvpQueryDeviceCapability – Queries for device capabilities.

#### 7 Synopsis

```
8 opvp_result_t opvpQueryDeviceCapability(
9     opvp_dc_t printerContext,
10    opvp_queryinfoflags_t queryflag,※夫谷★ヘッダでは opvp_flag_t になっているが?
11    opvp_int_t *buflen,※ GETROP の仕様にあわせる ポイントに変更 夫谷さんに確認
12    opvp_char_t *infoBuf);※夫谷さん★ヘッダでは opvp_byte_t* になっているが?
```

#### 13 Arguments

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

15 | queryflag – Flag which specifies which device capability to query.

16 | buflen – Number of bytes of the buffer specified pointed to by \*infoBuf.

17 | infoBuf – Pointer to the buffer to store the device capability.

#### 18 Description

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

20 |  
21 | ★どこに長さを返すのか? If caller sets NULL to infoBuf, printer driver MUST return the number of bytes to store all the capabilities in integer format. In this case, the caller also MUST set the queryflag and buflen. The caller MUST set one or more bit flags of info queryflag to query capabilities. The values in the following enumerations should be supported by the driver:  
22 |  
23 |  
24 |  
25 |

```
25 typedef enum _opvp_queryinfoflags {
26     OPVP_QF_DEVICERESOLUTION      = 0x00000001,
27     OPVP_QF_MEDIASIZE            = 0x00000002,
28     OPVP_QF_PAGEROTATION         = 0x00000004,
29     OPVP_QF_MEDIANUP             = 0x00000008,
30     OPVP_QF_MEDIADUPLEX          = 0x00000010,
31     OPVP_QF_MEDIASOURCE          = 0x00000020,
32     OPVP_QF_MEDIDESTINATION     = 0x00000040,
33     OPVP_QF_MEDIATYPE            = 0x00000080,
34     OPVP_QF_MEDIACOPY             = 0x000100000010, /* Maximum copy number supported */
35     OPVP_QF_PRINTREGION          = 0x00021000 /* only for opvpQueryDeviceInfo use */
36 } opvp_queryinfoflags_t;
37 値を返す Max の MediaCopy ★ OPVP_QF_MEDIASIZE と opvpQueryDeviceInfo のみだったか? (ヘッダではそう書いてあるが日本語版は。。。) 夫谷さんに確認。MediaCopy では? 値も違う。
38 |
```

39 | The driver MUST return the queried capabilities in ASCII text format via the infoBuf buffer in ASCII text format, and also return the number of bytes of the capabilities in text via \*buflen. If the buffer does not have enough length this too small to store all the capabilities queried the query result, ★エラーを返す the stored capabilities MUST be truncated by driver. In this case, driver MUST return without error the driver MUST return the necessary number of bytes to retrieve all the capabilities the query result in \*buflen, and return an error and set the detailed error code to OPVP\_PARAMERROR into opvpErrorNo. If the caller sets passes NULL in as the infoBuf buffer, the driver MUST return the number of the bytes of all the capabilities required to store the query result via in \*buflen.

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

50 | updf:DeviceResolution=deviceResolution\_600x600,deviceResolution\_1200x1200

1   **Return Value**  
2   OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
3   errorneopvpErrorNo.

#### 4   **4.3.2.opvpQueryDeviceInfo**

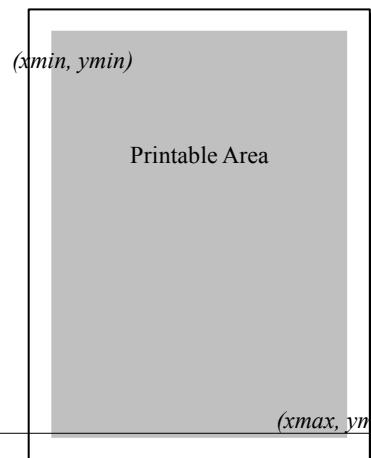
5   **Name**  
6   opvpQueryDeviceInfo – Queries for device information.

7   **Synopsis**  
8   opvp\_result\_t opvpQueryDeviceInfo(  
9       opvp\_dc\_t printerContext,  
10      opvp\_queryinfoflags\_t queryflag, **※★ヘッダでは opvp\_flag\_t になっているが?**  
11      opvp\_int\_t \*buflen, **※を返す バッファサイズ**  
12      opvp\_char\_t \*infoBuf); **※★opvpQueryDeviceCapability は opvp\_byte\_t\* になっているが?**

13   **Arguments**  
14    printerContext – Printer context value returned by the opvpOpenPrinter() function.  
15    queryflag – Flag which specifies which the device information to query.  
16    buflen – Number of bytes of the buffer specified pointed to by \*infoBuf.  
17    infoBuf – Pointer to the buffer to store the device information.

18   **Description**  
19    This function queries for information about the current settings of the printer or driver information.  
20    一追記ポインタに返す。★どこに長さを返すのか? If caller sets NULL to infoBuf, printer driver MUST return the number of  
21    bytes to store all the information in integer format. In this case, the caller also MUST set the queryflag and buflen.  
22    The caller MUST set one or more bit flags in queryflag to query for information. Values of the same enumerations  
23    which are used for the opvpQueryDeviceCapability() function MUST be used.  
24    エラーを返す★The driver MUST return queried information into the infoBuf buffer in ASCII text format. If the buffer does  
25    not have enough length to store all the information queried, the stored information MUST be truncated by driver. In this case,  
26    driver MUST return without error.  
27    The driver MUST return the queried information in ASCII text format into the infoBuf buffer in ASCII text format and  
28    also return the number of bytes of the information's text into via \*buflen. If the buffer does not have enough length this too  
29    small to store all the information queried query result, the driver MUST return the necessary number of bytes to retrieve all the  
30    information the query result in \*buflen, and return an error and set the detailed error code OPVP\_PARAMERROR into  
31    opvpErrorNo. If the caller sets passes NULL in as the infoBuf pointer, the driver MUST return the number of the bytes  
32    required to store the query result via of all the information in \*buflen.  
33    The format of the information name-and-value pairs format of each information stored in the infoBuf buffer has the same  
34    format as that of the jobInfo which is used for with the opvpStartJob() function.  
35    When the caller sets the OPVP\_OF\_PRINTREGION int bit of the queryflag, the driver MUST respond provide the  
36    printable area for its query in using the current resolution setting. The printable area values format MUST be as given  
37    below following. Each The values represents correspond to the x and y coordinates of the left top and right bottom corners,  
38    respectively, of the current printable area setting as shown in the figure. These values depend on the current media orientation  
39    setting.

40       PrintRegion=xmin,ymin,xmax,ymax  
41  
42  
43  
44  
45  
46  
47  
48



1  
2  
3

4     **Return Value**  
5     OPVP\_OK or -1 whenin case of error. In the latter case, and the driver MUST store the detailed error code in  
6     errornoopvpErrorNo.

## 4.4.Attributes for Job, Document and Page Operations

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

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

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

    • <scheme>: Name space for the following <key> and <value>.   

    • <key>: Name of the attribute.   

    • <value>: Value for given <key>.  

Multiple <value>s for a <key> MUST be separated by a semicolon given with the separator ";" (comma) for each <key>. If When multiple values are given for one single key, the printer driver should search the list of values from the beginning of starting with the first value in the multiple values for each key, and take use the first possible value that the driver can use under with the current settings.  

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

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

Conforming drivers which conforms to this specification MUST ignore unknown properties to keep them maintain compatibility with future vendor or standard extensions.  

Major UPDF attributes are shown in the table below:
```

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

Attribute	Name	Value	Effective in		
			Job	Doc	Page
		high normal			

23

24 | ~~document attributes bthe The page attributes always override Between the opvpStartPage () and opvpEndPage ()~~  
 25 | ~~functions calls, page attributes take precedence over document attributes, and the document attributes always override the job-~~  
 26 | ~~attributes Similarly, between the opvpStartDoc () and opvpEndDoc () functions calls, document attributes take~~  
 27 | ~~precedence over job attributes.~~ For example, ~~if a three page job which contains three pages and has the with a landscape~~  
 28 | ~~orientation attribute passed to given by the opvpStartJob () function; and if the second page has the a portrait orientation~~  
 29 | ~~attribute passed to given by the opvpStartPage () function for its second page, will use a landscape orientation for the first~~  
 30 | ~~page and third page have the landscape attribute but the second page has the will use portrait.~~

31 | ★propertyをattributeに全面変更。問題ないか？(章タイトルがAttributeだったため)

## 1 4.5.Graphics State Object Operations

These [section defines](#) functions [that operate on](#) the properties or drawing attributes of [the Graphics State Object](#). [Driver support for any](#)[All](#) of these functions [are](#)[is](#) OPTIONAL.

## 4.5.1.opvpResetCTM

5 Name

6 | opvpResetCTM – Initializes the 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 [the opvpOpenPrinter\(\)](#) function.

## 12 Description

13 | This function initializes the CTM of a Graphics State Object. The initial setting value of a CTM is as follow:

## 14 | -the identity matrix:

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

16 Return Value

17 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
18 | errorneOpvpErrorNo.

## 19 4.5.2.opvpSetCTM

20 Name

21 | opvpSetCTM – Sets the CTM to the of a Graphics State Object.

## 22 Synopsis

```
23     opvp_result_t opvpSetCTM(
24         opvp_dc_t printerContext,
25         opvp_ctm_t *pCTM);
```

26 Arguments

27 | printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.

28 | pCTM – Pointer to thean opvp\_ctm\_t structure holding the six CTM elements. It contains 6 opvp\_float\_t elements – a,  
29 | b, c, d, e and f.

## 30 Description

31 | This function sets thea CTM to theofa Graphics State Object.

32 The pPrinter (device) coordinate system valuecoordinates  $[x_{dev}, y_{dev}]$  is represented by CTM and the caller (renderer) coordinate  
33 system valuecoordinates  
34  $[x_{ren}, y_{ren}]$  are related via the CTM as shown in the following equation:-

$$35 \quad [x_{dev} \quad y_{dev} \quad 1] = [x_{ren} \quad y_{ren} \quad 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

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

```
5 Return Value
6     OPVP_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in
7     errno to opvpErrorNo.
```

## 8 4.5.3.opvpGetCTM

```
9 Name
10 | opvpGetCTM – Gets the CTM from the of a Graphics State Object.
```

### 11 **Synopsis**

```
12     opvp_result_t opvpGetCTM(
13         opvp_dc_t printerContext;
14         opvp_ctm_t *pCTM);
```

### 15 **Arguments**

```
16 | printerContext – Printer context value returned by the opvpOpenPrinter() function.
17 | pCTM – Pointer to the an opvp_ctm_t structure to receive the six CTM elements. It contains 6 opvp_float_t elements –
18 | a, b, c, d, e and f.
```

### 19 **Description**

```
20 | This function gets the CTM from the of a Graphics State Object.
```

### 21 **Structures**

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

### 25 **Return Value**

```
26 | OPVP_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in
27 | errno to opvpErrorNo.
```

## 28 4.5.4.opvpInitGS

```
29 Name
30 | opvpInitGS – Initializes the parameters in the of a Graphics State Object.
```

### 31 **Synopsis**

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

### 34 **Arguments**

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

### 36 **Description**

37 | This function initializes the parameters in the of a Graphics State Object. The driver MUST initialize the parameters in the of a  
38 | Graphics State Object when the caller calls this function. The Graphics State is a set of values which is managed by the printer  
39 | driver and contains parameters for drawing color, drawing mode and other, drawing related, settings for drawing. ★The driver  
40 | MUST keep the current parameters in the of a Graphics State Object unless opvpInitGS() or other parameter setting  
41 | functions are called. And also, the driver MUST save all parameters in the Graphics State Object onto the driver's stack when  
42 | the caller calls the opvpSaveGS() function, and restore all parameters from the stack to the Graphics State Object when the  
43 | caller calls the opvpRestoreGS() function. These two functions are described later in this document. These operations are  
44 | used to change the parameters in the Graphics State Object temporarily and restore them after drawing operations.

45 | ★The driver MUST keep the Graphics State Object parameters between opvpStartJob() and opvpEndJob(), unless  
46 | opvpInitGS() or other parameter setting functions are called. When the caller calls the opvpStartJob() function, the  
47 | driver MUST set the same parameters as the opvpInitGS() sets. ★StartJob, EndJob の間に限らないのでは?

1    **Return Value**  
2    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
3    errorneopvpErrorNo.

## 4    4.5.5.opvpSaveGS

5    **Name**  
6    opvpSaveGS – Saves the Graphics State Object parameters.

7    **Synopsis**  
8    opvp\_result\_t opvpSaveGS(  
9        opvp\_dc\_t printerContext);

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

12   **Description**  
13   This function saves the Graphics State Object parameters into the driver's stack.  
14   **Driver MUST be able to save at least one set of the Graphics State Object parameters into its stack. If caller calls this function more than once without calling opvpRestoreGS() function, this function may return error.** ★何のエラーを返す  
15   If the opvpSaveGS() function can be called N (>0) times, the opvpRestoreGS() function also MUST be able to be called N times. In this case, if the caller calls the opvpRestoreGS() function more than N times, this function MUST return error and set the error code OPVP\_BADREQUEST in errorneopvpErrorNo.

19   **Return Value**  
20   OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
21   errorneopvpErrorNo.

## 22   4.5.6.opvpRestoreGS

23   **Name**  
24   opvpRestoreGS – Restores the Graphics State Object parameters.

25   **Synopsis**  
26   opvp\_result\_t opvpRestoreGS(  
27        opvp\_dc\_t printerContext);

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

30   **Description**  
31   This function retrieves the Graphics State Object parameters from the driver's stack and restores them into the current Graphics State Object.  
32   If the opvpSaveGS() function can be called N (>0) times, the opvpRestoreGS() function also MUST be able to be called N times. In this case, if the caller calls the opvpRestoreGS() function more than N times, this function MUST return error and set the error code OPVP\_BADREQUEST in errorneopvpErrorNo.

36   **Return Value**  
37   OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
38   errorneopvpErrorNo.

## 39   4.5.7.opvpQueryColorSpace

40   **Name**  
41   opvpQueryColorSpace – Query color spaces that the driver can support.

42   **Synopsis**  
43   opvp\_result\_t opvpQueryColorSpace(

```

1     opvp_dc_t printerContext,
2     opvp_int_t *pnum,
3     opvp_cspace_t *pcspace);★順番入れ替えて良いか?
4     opvp_int_t *pnum

5 Arguments
6     printerContext – Printer context value returned by opvpOpenPrinter() function.
7     pnum – Pointer to the buffer to store the number of the color space enum array elements.
8     pcspce – Pointer to the color space enum. array.
9     pnum – Pointer to the buffer which stores the number of the color space enum array elements.

10 Description
11 This function queries the list of color space that the driver can support from the Graphics State Object.
12 -Driver MUST return the number of the color space enum retrieved in *pnum. If the number of enums exceeds the number of
13 elements prepared by the caller, driver MUST return the necessary number to retrieve all the enums in *pnum.★caller MUST
14 set the number of the pcspce elements in *pnum.★Caller MUST prepare at least 4 elements for the pcspce array to
15 store the color space enums.★4でよい? Also t
16 To query the color space enum in pcspce array, caller MUST set the number of pcspce elements into *pnum. In this
17 case, driver MUST return the color space enum in pcspce array and also return the number of the color space enum retrieved
18 in *pnum.
19 If caller sets NULL in *pcspce, driver MUST return only the number of the color space enum that the driver can support in
20 *pnum.
21 If the number of the color space enum exceeds the number of pcspce array elements prepared by the caller, driver MUST
22 return the necessary number to retrieve all the color space enum in *pnum, and return error and set the error code
23 OPVP_PARAMERROR into opvpErrorNo.
24 Driver returns the color space enum in the pcspce array by its preferable order. The first color space enum SHOULD specify
25 that it is the most preferable color space for the driver.

```

## 26 **Return Value**

27 OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in `errorneopvpErrorNo`.

## 28 **4.5.8.opvpSetColorSpace**

### 29 **Name**

30 opvpSetColorSpace – Sets **the current** color space **into the of a** Graphics State Object.

### 31 **Synopsis**

```

32     opvp_result_t opvpSetColorSpace(
33         opvp_dc_t printerContext,
34         opvp_cspace_t cspace);

```

### 35 **Arguments**

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

37 cspace – Color space **valueenum**.

### 38 **Description**

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

40 The **c**olor space **given passed** by **the** caller **in via** cspace MUST be one of the color spaces **retrieved returned** by the
41 opvpQueryColorSpace() function.

### 42 **Return Value**

43 OPVP\_OK or -1 **when in case of** error. **In the latter case, and** the driver MUST store the detailed error code in
44 `errorneopvpErrorNo`.

## 1    4.5.9.opvpGetColorSpace

### 2    **Name**

3 | opvpGetColorSpace – Gets the current color space from **thea** Graphics State Object.

### 4    **Synopsis**

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

### 8    **Arguments**

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

10 | pcspace – Pointer to the **buffer to store the** color space **enumvalue to be returned**.

### 11    **Description**

12 | This function gets the color space **whichthat** is currently set in **thea** Graphics State Object.

13 | **する**OpvpSetColorSpace()関数を呼ぶないで返る値はドライバに依存★デフォルト値は？

14 | **The initial color space in the of a Graphics State Object returned by the opvpOpenPrinter() function depends on eachis driver dependent.**

### 16    **Return Value**

17 | OPVP\_OK or -1 **when in case of error**. **In the latter case, and** the driver MUST store the detailed error code in **errorneopvpErrorNo**.

## 19 | 4.5.10.opvpQueryROP

### 20    **Name**

21 | opvpQueryROP – Query ROPs that the driver can support.

### 22    **Synopsis**

```
23    opvp_result_t opvpQueryROP(  
24        opvp_dc_t printerContext,  
25        opvp_int_t *pnum,  
26        opvp_rop_t *prop);
```

### 27    **Arguments**

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

29 | pnum – Pointer to the buffer to store the number of the ROP array elements.

30 | prop – Pointer to the ROP array.

### 31    **Description**

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

33 | **To query ROPs in prop array, caller MUST set the number of prop elements into \*pnum. In this case, Ddriver MUST return ROPs in prop array and also return the number of the ROPs retrieved in \*pnum. If the number of ROPs exceeds the number of elements prepared by the caller, driver MUST return the necessary number to retrieve all the ROPs in \*pnum.**

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

37 | 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 **errorneopvpErrorNo**. Driver returns ROPs in the array by its preferable order.★QueryColorSpaceと違う？

### 40    **Return Value**

41 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in **errorneopvpErrorNo**.

## 1 | 4.5.11.opvpSetROP

### 2 | **Name**

3 | opvpSetROP – Set ROP mode into the Graphics State Object.

### 4 | **Synopsis**

```
5 |     opvp_result_t opvpSetROP(  
6 |         opvp_dc_t printerContext,  
7 |         opvp_rop_t rop);
```

### 8 | **Arguments**

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

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

### 11 | **Description**

12 | This function sets ROP into the Graphics State Object.

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

### 14 | **Return Value**

15 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in [erroneopvpErrorNo](#).

## 16 | 4.5.12.opvpGetROP

### 17 | **Name**

18 | opvpGetROP – Get the ROP mode from the Graphics State Object.

### 19 | **Synopsis**

```
20 |     opvp_result_t opvpGetROP(  
21 |         opvp_dc_t printerContext,  
22 |         opvp_rop_t *prop);
```

### 23 | **Arguments**

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

25 | prop – Pointer to the buffer to store ROP.

### 26 | **Description**

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

28 | [Initial ROP mode in the Graphics State Object returned by the opvpOpenPrinter\(\) function depends on each driver.](#)  
29 | ドライバ依存★デフォルト値は?

### 30 | **Return Value**

31 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in [erroneopvpErrorNo](#).

## 32 | 4.5.13.opvpSetFillMode

### 33 | **Name**

34 | opvpSetFillMode – Sets [the](#) filling mode [into the](#) [of](#) [a](#) Graphics State Object.

### 35 | **Synopsis**

```
36 |     opvp_result_t opvpSetFillMode(  
37 |         opvp_dc_t printerContext,  
38 |         opvp_fillmode_t fillmode);
```

### 39 | **Arguments**

40 | printerContext – Printer context value returned by [the](#) opvpOpenPrinter() function.

41 | fillmode – Filling mode enum[eration value](#). OPVP\_FILLMODE\_EVENODD\_([Even-odd rule](#)), [and](#)  
42 | OPVP\_FILLMODE\_WINDING\_([Non-zero winding number rule](#)) can be set.

1    **Description**

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

3    **Return Value**

4 |    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
5 |    errorneopvpErrorNo.

6    **4.5.14.opvpGetFillMode**

7    **Name**

8 |    opvpGetFillMode – Gets the filling mode from thea Graphics State Object.

9    **Synopsis**

10 |    opvp\_result\_t opvpGetFillMode(  
11 |         opvp\_dc\_t printerContext,  
12 |         opvp\_fillmode\_t \*pfillmode);

13    **Arguments**

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

15 |    pfillmode – Pointer to the buffer to store the fill mode enumeration value to be returned.

16    **Description**

17 |    This function gets the filling mode which that is currently set in thea Graphics State Object.

18 |    The initial filling mode in the of a Graphics State Object returned by the opvpOpenPrinter() function depends on each his  
19 |    driver dependent.

20 |    ドライバ依存★デフォルト値は？

21    **Return Value**

22 |    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
23 |    errorneopvpErrorNo.

24    **4.5.15.opvpSetAlphaConstant**

25    **Name**

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

27    **Synopsis**

28 |    opvp\_result\_t opvpSetAlphaConstant(  
29 |         opvp\_dc\_t printerContext,  
30 |         opvp\_float\_t alpha);

31    **Arguments**

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

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

34    **Description**

35 |    This function sets the alpha blending constant, which is transparent ratio, into the of a Graphics State Object. The value of  
36 |    alpha MUST be between 0.0 and 1.0. If the value given specified by the caller extends is outside of this the range, the driver  
37 |    SHOULD truncate it between 0.0 and 1.0 use the nearest value that is within range.

38    **Return Value**

39 |    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
40 |    errorneopvpErrorNo.

## 1    4.5.16.opvpGetAlphaConstant

### 2    **Name**

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

### 4    **Synopsis**

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

### 8    **Arguments**

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

10 | palpha – Pointer to the buffer to store the alpha constant value to be returned.

### 11    **Description**

12 | This function gets the alpha constant value whichthat is currently set in the Graphics State Object.

13 | The initial alpha blending constant in the of a Graphics State Object returned by the opvpOpenPrinter() function depends on each is driver dependent.

15 | ドライバ依存★デフォルト値は？

### 16    **Return Value**

17 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in errorneopvpErrorNo.

## 19    4.5.17.opvpSetLineWidth

### 20    **Name**

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

### 22    **Synopsis**

```
23    opvp_result_t opvpSetLineWidth(  
24        opvp_dc_t printerContext,  
25        opvp_fix_t width);
```

### 26    **Arguments**

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

28 | width – Line width value.

### 29    **Description**

30 | This function sets line width for stroke operations to the by a Graphics State Object. The line width MUST be set in the device coordinate system units.

32 | The treatment of line widths less than one depends on the device or driver implementation. Similarly, and the maximum line width that can be set also depends on the device or driver implementation.

### 34    **Return Value**

35 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in errorneopvpErrorNo.

## 37    4.5.18.opvpGetLineWidth

### 38    **Name**

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

### 40    **Synopsis**

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

1    **Arguments**

2 |    printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
3 |    pwidth – Pointer to [the buffer to store](#) the line width value [to be returned](#).

4    **Description**

5 |    This function gets the -line width for stroke operations from [thea](#) Graphics State Object. The line width value MUST be in [the](#)-  
6 |    device coordinate system units.

7 |    [The initial line width in theof a Graphics State Object returned by the opvpOpenPrinter\(\) function depends on each his](#)  
8 |    [driver dependent](#).

9 |    ドライバ依存★デフォルト値は?

10    **Return Value**

11 |    OPVP\_OK or -1 [whenin case of](#) error. [In the latter case, and](#) the driver MUST store the detailed error code in  
12 |    [erroneopvpErrorNo](#).

13    **4.5.19.opvpSetLineDash**

14    **Name**

15 |    opvpSetLineDash – Sets the line dash pattern [of a Graphics State Object](#).

16    **Synopsis**

17    opvp\_result\_t opvpSetLineDash(  
18         opvp\_dc\_t printerContext,  
19         opvp\_fix\_t \*pdash,  
20         opvp\_int\_t num);

21    **Arguments**

22 |    printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
23 |    pdash – Pointer to the line dash pattern array.  
24 |    num – Number of [the line dash pattern array](#) elements [in pdash](#).

25    **Description**

26 |    This function sets a [stroke](#) line dash pattern [tofor use by a the](#) Graphics State Object.

27 |    When the painting mode is OPVP\_PAINTMODE\_OPAQUE\_([oO](#)paque mode), the odd numbered (1<sup>st</sup>, 3<sup>rd</sup>, 5<sup>th</sup>, 7<sup>th</sup> ...) elements  
28 |    [inof](#) the pdash array [isindicate](#) the length for [dashes in the](#) foreground color; and the even numbered (2<sup>nd</sup>, 4<sup>th</sup>, 6<sup>th</sup>, 8<sup>th</sup>...)  
29 |    elements [isindicate the](#) a length for background color [dashes](#).

30 |    When the painting mode is OPVP\_PAINTMODE\_TRANSPARENT\_([tT](#)ransparent mode), the odd numbered (1<sup>st</sup>, 3<sup>rd</sup>, 5<sup>th</sup>, 7<sup>th</sup> ...) elements  
31 |    [inof](#) the pdash array [isindicate](#) the length for [dashes in the](#) foreground color; and the even numbered (2<sup>nd</sup>, 4<sup>th</sup>, 6<sup>th</sup>,  
32 |    8<sup>th</sup>...) elements [is=indicate the](#) length of a-line segments [which is=that](#) are not painted.

33 |    The lengths set in the pdash array MUST be in [the](#)device coordinate system units.

34 |    If the number of elements of the pdash array is odd, the dash pattern is created as if the number of elements is double the  
35 |    number of elements of the array. In this case, the first element of the pdash array in the second cycle is treated as the length  
36 |    for background color [dashes](#) in case of [OPVP\\_PAINTMODE\\_OPAQUE](#) opaque mode and for non-painted [length](#)line segments in  
37 |    case of [OPVP\\_PAINTMODE\\_TRANSPARENT](#) transparent mode.

38 |    The maximum number of elements [that can be set](#) depends on the device or driver implementation.

39 |    If [the caller setpasses](#) zero [infor num](#), [the](#) driver MUST draw solid lines.

40    **Return Value**

41 |    OPVP\_OK or -1 [whenin case of](#) error. [In the latter case, and](#) the driver MUST store the detailed error code in  
42 |    [erroneopvpErrorNo](#).

43    **4.5.20.opvpGetLineDash**

44    **Name**

45 |    opvpGetLineDash – Gets the line dash pattern [of a Graphics State Object](#).

1    **Synopsis**

```
2    opvp_result_t opvpGetLineDash(
3         opvp_dc_t printerContext,
4         opvp_fix_t *pdash,
5         opvp_int_t *pnum);
```

6    **Arguments**

7 |    printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
8 |    pdash – Pointer to the buffer to store the line dash pattern array.  
9 |    pnum – Pointer to [the buffer to store](#) the number of [the line dash pattern array](#) elements [in pdash](#).

10 |    **Description**

11 |    This function gets the [stroke](#) line dash pattern from [the](#) Graphics State Object.  
12 |    The caller MUST allocate at least [one](#) elements for [the](#) pdash array. Also, the caller MUST set the number of elements of  
13 |    pdash allocated to \*pnum. [The dDriver](#) MUST return the number of dash patterns retrieved in \*pnum. If the number of dash  
14 |    patterns exceeds the number of elements [preparedallocated](#) by the caller, [the](#) driver MUST return the necessary number to  
15 |    retrieve all the dash patterns in \*pnum.  
16 |    If caller [sets](#) [NULL](#) [in](#) [for](#) pdash, [the](#) driver MUST return the number of [the](#) dash patterns which can be used by the driver  
17 |    in \*pnum.  
18 |    [The initial line dash pattern in the of a Graphics State Object returned by the opvpOpenPrinter\(\) function depends on](#)  
19 |    [each](#) [is](#) [driver](#) [dependent](#).  
20 |    ドライバ依存★デフォルト値は？

21 |    **Return Value**

22 |    OPVP\_OK or -1 [when](#) [in](#) [case](#) [of](#) [error](#). [In](#) [the](#) [latter](#) [case](#), [and](#) the driver MUST store the detailed error code in  
23 |    [error](#) [the](#) [opvpErrorNo](#).

24 |    **4.5.21.opvpSetLineDashOffset**

25 |    **Name**

26 |    opvpSetLineDashOffset – Sets the line dash pattern offset [of a Graphics State Object](#).

27 |    **Synopsis**

```
28    opvp_result_t opvpSetLineDashOffset(
29         opvp_dc_t printerContext,
30         opvp_fix_t offset);
```

31 |    **Arguments**

32 |    printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
33 |    offset – Offset value [for applying](#) [of](#) the line dash pattern.

34 |    **Description**

35 |    This function sets the offset value of the line dash pattern for stroke operations in [the](#) device coordinate system units.

36 |    **Return Value**

37 |    OPVP\_OK or -1 [when](#) [in](#) [case](#) [of](#) [error](#). [In](#) [the](#) [latter](#) [case](#), [and](#) the driver MUST store the detailed error code in  
38 |    [error](#) [the](#) [opvpErrorNo](#).

39 |    **4.5.22.opvpGetLineDashOffset**

40 |    **Name**

41 |    opvpGetLineDashOffset – Gets the line dash pattern offset [of a Graphics State Object](#).

42 |    **Synopsis**

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

1    **Arguments**  
2 |    printerContext – Printer context value returned by the `opvpOpenPrinter()` function.  
3 |    poffset – Pointer to the buffer to store the offset value of the line dash pattern to be returned.  
  
4    **Description**  
5 |    This function gets the offset value of the line dash pattern from the Graphics State Object.  
6 |    The initial line dash pattern offset in the of a Graphics State Object returned by the opvpOpenPrinter() function depends on each is driver dependent.  
7 |  
8 |    ドライバ依存★デフォルト値は？  
  
9    **Return Value**  
10 |    `OPVP_OK` or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
11 |    errorneopvpErrorNo.

## 12    4.5.23.opvpSetLineStyle

13    **Name**  
14 |    `opvpSetLineStyle` – Sets the line style of a Graphics State Object.  
  
15    **Synopsis**  
16 |    `opvp_result_t opvpSetLineStyle(`  
17 |        `opvp_dc_t printerContext,`  
18 |        `opvp_linestyle_t linestyle);`  
  
19    **Arguments**  
20 |    printerContext – Printer context value returned by the `opvpOpenPrinter()` function.  
21 |    linestyle – Line style enumeration value. `OPVP_LINESTYLE_SOLID` (solid line) and/or `OPVP_LINESTYLE_DASH`  
22 |    (dashed line) can be set.★図は不要か？  
  
23    **Description**  
24 |    This function sets the line style to the of a Graphics State Object.

25    **Return Value**  
26 |    `OPVP_OK` or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
27 |    errorneopvpErrorNo.

## 28    4.5.24.opvpGetLineStyle

29    **Name**  
30 |    `opvpGetLineStyle` – Gets the line style of a Graphics State Object.  
  
31    **Synopsis**  
32 |    `opvp_result_t opvpGetLineStyle(`  
33 |        `opvp_dc_t printerContext,`  
34 |        `opvp_linestyle_t *plinestyle);`  
  
35    **Arguments**  
36 |    printerContext – Printer context value returned by the `opvpOpenPrinter()` function.  
37 |    plinestyle – Pointer to the buffer to store the line style enumeration value to be returned.  
  
38    **Description**  
39 |    This function gets the line style from the Graphics State Object.  
40 |    The initial line style in the of a Graphics State Object returned by the opvpOpenPrinter() function depends on each is driver dependent.  
41 |  
42 |    ドライバ依存★デフォルト値は？

1    **Return Value**  
2    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
3    errorneopvpErrorNo.

## 4    4.5.25.opvpSetLineCap

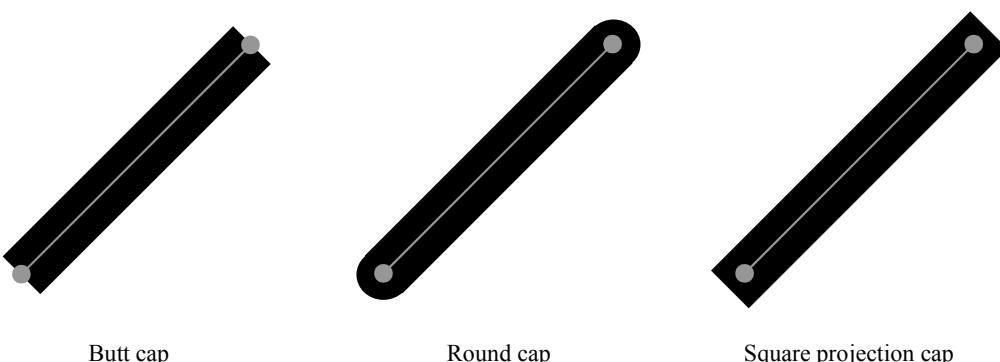
5    **Name**  
6    opvpSetLineCap – Sets the line cap style of a Graphics State Object.

7    **Synopsis**  
8    

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

### 11    **Arguments**

12    printerContext – Printer context value returned by the opvpOpenPrinter() function.  
13    linecap – Line cap style enumeration value. OPVP\_LINECAP\_BUTT\_(Butt cap), OPVP\_LINECAP\_ROUND\_(Round  
14    cap) and/or OPVP\_LINECAP\_SQUARE\_(Square projection cap) can be set.  
15



17    **Description**  
18    This function sets the line cap style to the of a Graphics State Object.

$$\text{Miter Limit} = \frac{\text{Miter Length}}{\text{LineWidth}} = \frac{1}{\sin(\frac{\varphi}{2})}$$

19    **Return Value**  
20    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
21    errorneopvpErrorNo.

## 22    4.5.26.opvpGetLineCap

23    **Name**  
24    opvpGetLineCap – Gets the line cap style of a Graphics State Object.

25    **Synopsis**  
26    

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

29    **Arguments**  
30    printerContext – Printer context value returned by the opvpOpenPrinter() function.  
31    plinecap – Pointer to the buffer to store the line cap style enumeration value to be returned.

32    **Description**  
33    This function gets the line cap style from the of a Graphics State Object.  
34    The initial line cap style in the of a Graphics State Object returned by the opvpOpenPrinter() function depends on each his  
35    driver dependent.

1 | ドライバ依存★デフォルト値は？

2 | **Return Value**

3 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
4 | errno to opvpErrorNo.

5 | **4.5.27.opvpSetLineJoin**

6 | **Name**

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

8 | **Synopsis**

```
9 |     opvp_result_t opvpSetLineJoin(  
10 |         opvp_dc_t printerContext,  
11 |         opvp_linejoin_t linejoin);
```

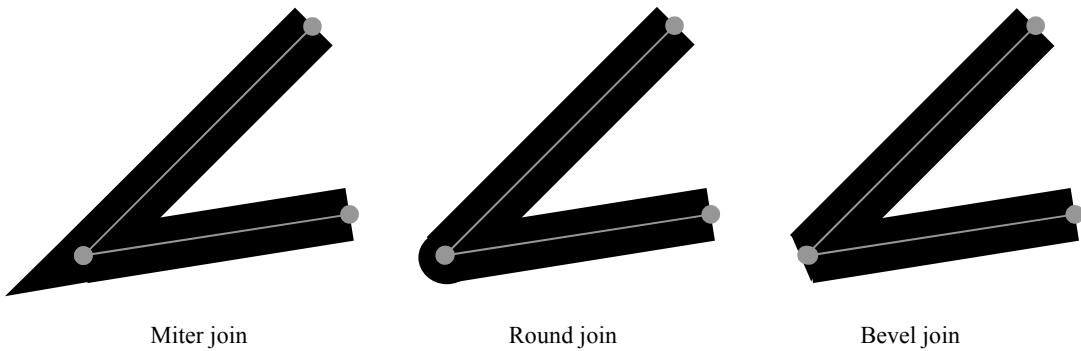
12 | **Arguments**

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

14 | linejoin – Line join style enumeration value. OPVP\_LINEJOIN\_MITER\_(mMiter join), OPVP\_LINEJOIN\_ROUND  
15 | (rRound join) or OPVP\_LINEJOIN\_BEVEL\_(bBevel join) can be set.

16 | **Description**

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



Miter join

Round join

Bevel join

20 | **Return Value**

21 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
22 | errno to opvpErrorNo.

23 | **4.5.28.opvpGetLineJoin**

24 | **Name**

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

26 | **Synopsis**

```
27 |     opvp_result_t opvpGetLineJoin(  
28 |         opvp_dc_t printerContext,  
29 |         opvp_linejoin_t *plinejoin);
```

30 | **Arguments**

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

32 | plinejoin – Pointer to the buffer to store the line join style enumeration value to be returned.

33 | **Description**

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

35 | The initial line join style in the of a Graphics State Object returned by the opvpOpenPrinter() function depends on each his  
36 | driver dependent.

1 | ドライバ依存★デフォルト値は？

## 2 | **Return Value**

3 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
4 | `errno` or `opvpErrorNo`.

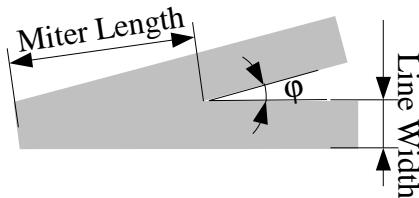
## 5 | **4.5.29.opvpSetMiterLimit**

### 6 | **Name**

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

### 8 | **Synopsis**

```
9 |     opvp_result_t opvpSetMiterLimit(  
10 |         opvp_dc_t printerContext,  
11 |         opvp_fix_t miterlimit);
```



### 12 | **Arguments**

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

15 | `miterlimit` – Maximum miter length.

### 16 | **Description**

17 | This function sets the maximum length of the miter to the created by a Graphics State Object. The miter limit is effective only  
18 | when the line join style is OPVP\_LINEJOIN\_MITER. The length MUST be set in the device coordinate system units. – 説明も  
19 | 反映三原さんの図を入れる

20 | 図は不要か？★

21 |

### 22 | **Return Value**

23 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
24 | `errno` or `opvpErrorNo`.

## 25 | **4.5.30.opvpGetMiterLimit**

### 26 | **Name**

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

### 28 | **Synopsis**

```
29 |     opvp_result_t opvpGetMiterLimit(  
30 |         opvp_dc_t printerContext,  
31 |         opvp_fix_t *pmiterlimit);
```

### 32 | **Arguments**

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

34 | `pmiterlimit` – Pointer to the buffer to store the maximum miter length to be returned.

### 35 | **Description**

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

37 | The initial miter limit value in the of a Graphics State Object returned by the `opvpOpenPrinter()` function depends on  
38 | each his driver dependent.

39 | 依存ドライバ★デフォルト値は？

### 40 | **Return Value**

41 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
42 | `errno` or `opvpErrorNo`.

## 1    4.5.31.opvpSetPaintMode

### 2    **Name**

3 | opvpSetPaintMode – Sets the background color painting mode [of a Graphics State Object](#).

### 4    **Synopsis**

```
5    opvp_result_t opvpSetPaintMode(  
6        opvp_dc_t printerContext,  
7        opvp_paintmode_t paintmode);
```

### 8    **Arguments**

9 | printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.

10 | paintmode – Painting mode [enumeration value](#). OPVP\_PAINTMODE\_OPAQUE ([Opaque mode](#)) [or and](#)  
11 | OPVP\_PAINTMODE\_TRANSPARENT ([Transparent mode](#)) can be set.

### 12 | **Description**

13 | This function sets the background color painting mode [to the of a Graphics State Object](#). ★図は不要か？

14 |

### 15 | **Return Value**

16 | OPVP\_OK or -1 [when in case of error](#). [In the latter case, and](#) the driver MUST store the detailed error code in  
17 | [errorneopvpErrorNo](#).

## 18    4.5.32.opvpGetPaintMode

### 19 | **Name**

20 | opvpGetPaintMode – Gets the background color painting mode [of a Graphics State Object](#).

### 21 | **Synopsis**

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

### 25 | **Arguments**

26 | printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.

27 | ppaintmode – Pointer to [the buffer to store](#) the painting mode [enumeration value to be returned](#).

### 28 | **Description**

29 | This function gets the background color painting mode [from the of a Graphics State Object](#).

30 | [The initial painting mode in the of a Graphics State Object returned by the opvpOpenPrinter\(\) function depends on each is](#)  
31 | [driver dependent](#) 依存ドライバ ★デフォルト値は？

32 |

### 33 | **Return Value**

34 | OPVP\_OK or -1 [when in case of error](#). [In the latter case, and](#) the driver MUST store the detailed error code in  
35 | [errorneopvpErrorNo](#).

## 36    4.5.33.opvpSetStrokeColor

### 37 | **Name**

38 | opvpSetStrokeColor – [Define Sets](#) the stroke color and pattern for [the opvpStrokePath\(\)](#) and [opvpStrokeFillPath\(\)](#)  
39 | operations [of a Graphics State Object](#).

### 40 | **Synopsis**

```
41    opvp_result_t opvpSetStrokeColor(  
42        opvp_dc_t printerContext,  
43        opvp_brush_t *brush);
```

## 1 | **Arguments**

2 |   printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
3 |   brush – Pointer to the `opvp_brush_t` structure data.

## 4 | **Description**

5 |   This function registers the color or pattern as a brush data to the Graphics State Object for drawing stroke by the  
6 |   `opvpStrokePath()` and `opvpStrokeFillPath()` operations.  
7 |   If the member `pbrush` in `opvp_brush_t` structure `*brush` is `NULL`, driver MUST treat it as a solid brush. In this case,  
8 |   color value of the solid brush is specified in `color[]` in the `opvp_brush_t` structure `*brush`, and the color space for the  
9 |   solid brush is specified by `colorSpace`.  
10 |   If the member `pbrush` in `opvp_brush_t` structure `*brush` points to a `opvp_brushdata_t` structure data, driver MUST  
11 |   treat it as a pattern brush. In the `opvp_brushdata_t` structure data, `width`, `height` are specified in pixel value, and the  
12 |   brush pattern horizontal repetition pitch `pitch` is specified in [the number of bytes](#).★合ってるよね?  
13 |   The actual pattern data is specified by data array★あってる?  
14 |   The color space for the pattern is specified by `colorSpace` in the `opvp_brush_t` structure  
–, and `color[1]` MUST be ignored.

## 15 | **Structures**

```
16 |   typedef struct _opvp_brushdata {  
17 |     opvp_bdtype_t type;  
18 |     opvp_int_t width, height, pitch;  
19 |     opvp_byte_t data[]; /* must be defined as data[1] for GCC 2.x */  
20 |   } opvp_brushdata_t;  
21 |  
22 |   typedef struct _opvp_brush {  
23 |     opvp_cspace_t colorSpace;  
24 |     opvp_int_t color[4];  
25 |     opvp_int_t xorg, yorg; /* brush origin, ignored for opvpSetBgColor */  
26 |     opvp_brushdata_t *pbrush; /* pointer to brush data */  
27 |   } opvp_brush_t;  
28 |  
29 |   typedef enum _opvp_bdtype {OPVP_BDTYPE_NORMAL = 0} opvp_bdtype_t;
```

## 31 | **Return Value**

32 |   `OPVP_OK` or `-1` when in case of error. In the latter case, and the driver MUST store the detailed error code in  
33 |   [errorneopvpErrorNo](#).

## 34 | **4.5.34.opvpSetFillColor**

### 35 | **Name**

36 |   `opvpSetFillColor` – [DefineSets](#) the fill color and pattern for [the opvpFillPath\(\)](#) and `opvpStrokeFillPath()`  
37 |   operations [of a Graphics State Object](#).

### 38 | **Synopsis**

```
39 |   opvp_result_t opvpSetFillColor(  
40 |     opvp_dc_t printerContext,  
41 |     opvp_brush_t *brush);
```

### 42 | **Arguments**

43 |   printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
44 |   brush – Pointer to brush structure data.

### 45 | **Description**

46 |   This function registers the color or pattern as a brush data to the Graphics State Object for filling by the `opvpFillPath()`  
47 |   and `opvpStrokeFillPath()` operations..  
48 |   The definition and operation of the `opvp_brush_t` structure `*brush` is same as the `opvpSetStrokeColor()` function  
49 |   except filling inside the path by the color or pattern with the brush registered by this function.  
50 |   See the `opvpSetStrokeColor()` function description.  
51 |   The initial fill color in the of a Graphics State Object returned by the `opvpOpenPrinter()` function depends on each is  
52 |   driver dependent.

1 | ドライバ依存デフォルト値は?★

2 | **Return Value**

3 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
4 | `errorneopvpErrorNo`.

5 | **4.5.35.opvpSetBgColor**

6 | **Name**

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

9 | **Synopsis**

10 | `opvp_result_t SetBgColor(  
11 |     opvp_dc_t printerContext,  
12 |     opvp_brush_t *brush);`

13 | **Arguments**

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

15 | `brush` – Pointer to brush structure data.

16 | **Description**

17 | This function registers the background color or pattern for `opvpStrokePath()`, `opvpFillPath()` and  
18 | `opvpStrokeFillPath()` operations.

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

21 | The initial background color in the of a Graphics State Object returned by the `opvpOpenPrinter()` function depends on each his driver dependent.

23 | ドライバ依存★デフォルト値は?

24 | Caller MUST set NULL to the member `pbrush` in the `opvp_brush_t` structure, or this function MUST return error and set  
25 | the error code `OPVP_BADREQUEST` in `errorneopvpErrorNo`.

26 | **Return Value**

27 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
28 | `errorneopvpErrorNo`.

## 1 4.6.Path Operations

2 | This sections defines functions for path operations.  
3 | A pPath is used for the-drawing commandss, such as “stroke”, “fill” and “stroke and fill”, as well as for defining the clipping  
4 | area. OneAny Graphics State Object keepsmaintains only one path at oneany one time. A pPath is affected by the CTM and is  
5 | registered intowith the Graphics State Object -when the caller calls eacha path operation function.  
6 | AllDriver support for any of the path operation functions areis OPTIONAL.

### 7 4.6.1.opvpNewPath

8 | **Name**  
9 | opvpNewPath – Starts a new path.  
  
10 | **Synopsis**  
11 | opvp\_result\_t opvpNewPath(  
12 |     opvp\_dc\_t printerContext);  
  
13 | **Arguments**  
14 | printerContext – Printer context value returned by the opvpOpenPrinter() function.  
  
15 | **Description**  
16 | This function dEletes the Graphics State Object's current path kept in the Graphics State Object and starts a new pathwhich.  
17 | The new path MUST be empty.  
18 | The initial path in the of a Graphics State Object returned by the opvpOpenPrinter () function MUST be empty. ★初期状  
19 | 態のパスは？ 空でなければならぬ。  
20 |  
  
21 | **Return Value**  
22 | OPVP\_OK or -1 whenin case of error. In the latter case, and the driver MUST store the detailed error code in  
23 | errornoopvpErrorNo.

### 24 4.6.2.opvpEndPath

25 | **Name**  
26 | opvpEndPath – Declares the end of the current path.  
  
27 | **Synopsis**  
28 | opvp\_result\_t opvpEndPath(  
29 |     opvp\_dc\_t printerContext);  
  
30 | **Arguments**  
31 | printerContext – Printer context value returned by the opvpOpenPrinter() function.  
  
32 | **Description**  
33 | This function dEclares the end of the Graphics State Object's current path.  
34 | The Graphics State Object MUST retain the path as the current path.  
  
35 | **Return Value**  
36 | OPVP\_OK or -1 whenin case of error. In the latter case, and the driver MUST store the detailed error code in  
37 | errornoopvpErrorNo.

### 38 4.6.3.opvpStrokePath

39 | **Name**  
40 | opvpStrokePath – Draws lines along the current path.

1    **Synopsis**

2    opvp\_result\_t opvpStrokePath(  
3        opvp\_dc\_t printerContext);

4    **Arguments**

5 |    printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.

6    **Description**

7 |    This function draws lines along the current path according to the drawing attributes registered [inwith](#) the Graphics State Object.

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

9    **Return Value**

10 |    OPVP\_OK or -1 [whenin case of error](#). [In the latter case, and](#) the driver MUST store the detailed error code in  
11 |    [erroneopvpErrorNo](#).

## 12    4.6.4.opvpFillPath

13    **Name**

14 |    opvpFillPath – Fills [the interior of](#) [inside](#) the current path.

15    **Synopsis**

16    opvp\_result\_t opvpFillPath(  
17        opvp\_dc\_t printerContext);

18    **Arguments**

19 |    printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.

20    **Description**

21 |    This function fills [inside](#) [the interior of](#) the current path according to the drawing attributes registered [inwith](#) the Graphics State Object.

23 |    When the path is not closed, the starting point and end point of the current path are connected by a straight line [\(but not stroked\)](#)  
24 |    [and closed](#). [This line is used solely to determine the interior of the path. It is not stroked](#).

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

26    **Return Value**

27 |    OPVP\_OK or -1 [whenin case of error](#). [In the latter case, and](#) the driver MUST store the detailed error code in  
28 |    [erroneopvpErrorNo](#).

## 29    4.6.5.opvpStrokeFillPath

30    **Name**

31 |    opvpStrokeFillPath – Draw[s](#) lines along the current path and fill[s](#) [inside](#) [the interior of](#) the current path.

32    **Synopsis**

33    opvp\_result\_t opvpStrokeFillPath(  
34        opvp\_dc\_t printerContext);

35    **Arguments**

36 |    printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.

37    **Description**

38 |    This function draws lines along the current path, and fills [inside](#) [the interior of](#) the current path according to the drawing attributes registered [inwith](#) the Graphics State Object.

40 |    When the path is not closed, the starting point and end point of the current path are connected by a straight line [\(but not stroked\)](#)  
41 |    [and closed](#). [This line is used solely to determine the interior of the path. It is not stroked](#).

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

1    **Return Value**  
2    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
3    errorneopvpErrorNo.

#### 4    4.6.6.opvpSetClipPath

5    **Name**  
6    opvpSetClipPath – Sets the current path as the clipping region.

7    **Synopsis**  
8    opvp\_result\_t opvpSetClipPath(  
9         opvp\_dc\_t printerContext,  
10      opvp\_cliprule\_t clipRule);

11   **Arguments**  
12   printerContext – Printer context value returned by the opvpOpenPrinter() function.  
13   clipRule – Clipping rule enumeration value. OPVP\_CLIPRULE\_EVENODD\_(eEven-odd rule) or and  
14   OPVP\_CLIPRULE\_WINDING\_(nNon-zero winding number rule) can be set.

15   **Description**  
16   This function sets the current path as a clipping region in for the Graphics State Object.  
17   When the path is not closed, the starting point and end point of the current path are connected by a straight line and closed.  
18   The current path MUST be retained in the Graphics State Object after calling this function.  
19   The clipping region also MUST be retained except until the caller invokes one of calls the opvpSetClipPath(),  
20   opvpResetClipPath(), opvpRestoreGS() or opvpStartPage() function. When the caller calls the  
21   opvpStartPage() function, the driver MUST reset the clipping region the printable area of the media whole in the Graphics  
22   State Object to cover as if the opvpResetClipPath() function had been called. ★表現変更、このような仕様でよいか?  
23  

24   **Return Value**  
25   OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
26   errorneopvpErrorNo.

#### 27   4.6.7.opvpResetClipPath

28   **Name**  
29   opvpResetClipPath – Resets the clipping region.

30   **Synopsis**  
31   opvp\_result\_t opvpResetClipPath(  
32         opvp\_dc\_t printerContext);

33   **Arguments**  
34   printerContext – Printer context value returned by the opvpOpenPrinter() function.  
35   **Description**  
36   This function resets the clipping region in the of a Graphics State Object to cover whole all of the printable area of the media.

37   **Return Value**  
38   OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
39   errorneopvpErrorNo.

#### 40   4.6.8.opvpSetCurrentPoint

41   **Name**  
42   opvpSetCurrentPoint – Sets the current point.

```

1 Synopsis
2     opvp_result_t opvpSetCurrentPoint(
3         opvp_dc_t printerContext,
4         opvp_fix_t x,
5         opvp_fix_t y);

```

## 6 **Arguments**

7 | printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
8 | x – [value of the x coordinate of value to set](#) the [new](#) current point.  
9 | y – [value of the y coordinate of value to set](#) the [new](#) current point.

## 10 **Description**

11 | [This function sets](#) the current point [of the Graphics State Object](#) to [\(x, y\)](#).

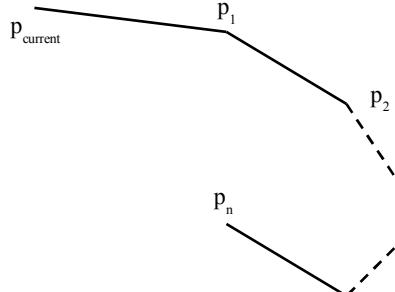
## 12 **Return Value**

13 | OPVP\_OK or -1 [when in case of error](#). [In the latter case, and](#) the driver MUST store the detailed error code in  
14 | [errorneopvpErrorNo](#).

## 15 **4.6.9.opvpLinePath**

### 16 **Name**

17 | opvpLinePath – Adds multiple connected line [segments](#) to the current path.



### 18 **Synopsis**

```

19     opvp_result_t opvpLinePath(
20         opvp_dc_t printerContext,
21         opvp_pathmode_t flag;
22         opvp_int_t npoints,
23         opvp_point_t *points);

```

### 24 **Arguments**

25 | printerContext – Printer context value returned by [the opvpOpenPrinter\(\)](#) function.  
26 | flag – Path open/close flag. [One of](#) OPVP\_PATHCLOSE [or](#) OPVP\_PATHOPEN can be set.  
27 | npoints – Number of [the opvp\\_point\\_t structure array](#) elements [in points](#).  
28 | points – Pointer to [the an](#) opvp\_point\_t structure array.

### 29 **Description**

30 | This function adds [the line segments](#) specified by the points array [from starting](#) at the current point.

31 | When [the caller sets passes](#) OPVP\_PATHOPEN [as](#) flag, [the driver](#) MUST append [the line segments](#) from the current point, [then set and make](#) the last point of the points array [as](#) the [new](#) current point. When [the caller sets passes](#) OPVP\_PATHCLOSE [as](#) flag, [the driver](#) MUST append [the line segments](#) from the current point, [then set and make](#) the first point of the points array [as](#) the [new](#) current point.

### 35 **Structures**

```

36     typedef struct _opvp_point {
37         opvp_fix_t x, y;
38     } opvp_point_t;

```

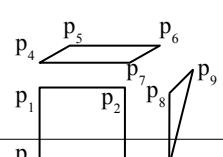
### 39 **Return Value**

40 | OPVP\_OK or -1 [when in case of error](#). [In the latter case, and](#) the driver MUST store the detailed error code in  
41 | [errorneopvpErrorNo](#).

## 42 **4.6.10.opvpPolygonPath**

### 43 **Name**

44 | opvpPolygonPath – Adds polygons to the current path.



```

1 Synopsis
2     opvp_result_t opvpPolygonPath(
3         opvp_dc_t printerContext,
4         opvp_int_t npolygons,
5         opvp_int_t *nvertices,
6         opvp_point_t *points);

7 Arguments
8     printerContext – Printer context value returned by the opvpOpenPrinter() function.
9     npolygons – Number of polygons to add.
10    nvertices – Pointer to the an array of with the number of points of each polygon.
11    points – Pointer to the an opvp_point_t structure array. The number of points in the this array MUST be equal to the sum total number of points the values in the nvertices array.

```

```

13 Description
14     This function adds the polygons specified by the points array in via its arguments to the current path from the current point.★カレントポイントをどこにあわせて追加する？
15
16     After driver appends the polygons have been added from the current point, the driver MUST setmake the last point of the points array as the new current point.
17

```

```

18 Return Value
19     OPVP_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in errorneopvpErrorNo.
20

```

## 4.6.11.opvpRectanglePath

```

22 Name
23     opvpRectanglePath – Adds rectangles to the current path.

```

```

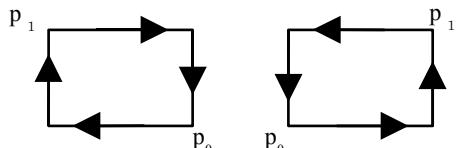
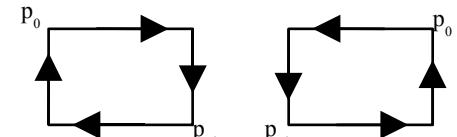
24 Synopsis
25     opvp_result_t opvpRectanglePath(
26         opvp_dc_t printerContext,
27         opvp_int_t nrectangles,
28         opvp_rectangle_t *rectangles);

```

```

29 Arguments
30     printerContext – Printer context value returned by the opvpOpenPrinter() function.
31
32     nrectangles – Number of rectangles to add.
33     rectangles – Pointer to the an opvp_rectangle structure array.

```



```

34 Description
35     This function adds rectangles into the current path from the current point.★カレントポイントをどこにあわせて追加？
36     After driver appends the rectangles have been  appended from the current point, the driver MUST setmake the starting point of the last rectangle appended as the new current point.
37
38     The direction of the paths of each rectangle are specified by the starting point and diagonal points as above illustrated in the figure above. The path is appended in order of  $(x_0, y_0)$ - $(x_1, y_0)$ - $(x_1, y_1)$ - $(x_0, y_1)$ - $(x_0, y_0)$  where the starting point  $p_0$  is  $(x_0, y_0)$  and the diagonal point  $p_1$  is  $(x_1, y_1)$ .
39
40

```

```

41 Structures
42     typedef struct _opvp_rectangle {
43         opvp_point_t p0;                      /* starting point */
44         opvp_point_t p1;                      /* diagonal point */
45     } opvp_rectangle_t;

```

```

46 Return Value
47     OPVP_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in errorneopvpErrorNo.
48

```

## 4.6.12.opvpRoundRectanglePath

### 2 Name

3 | opvpRoundRectanglePath – Adds rounded rectangles to the current path.

### 4 Synopsis

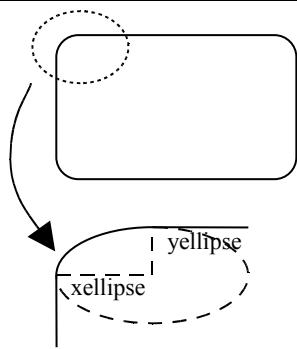
```
5 opvp_result_t opvpRoundRectanglePath(
6     opvp_dc_t printerContext,
7     opvp_int_t nrectangles,
8     opvp_roundrectangle_t *rectangles);
```

### 9 Arguments

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

11 | nrectangles – Number of round rectangles to add.★図修正 with pitch ではない

12 | rectangles – Pointer to the an opvp\_roundrectangle\_t structure array.



### 13 Description

14 | This function adds rounded rectangles into the current path.★カレントポイントを何処にあわせて追加？ from the current point

15 | Each corner of a rounded rectangle is connected by an elliptic arc defined by the values of the xellipse and yellipse  
16 | members of the opvp\_roundrectangle\_t structure. After driver appends the rounded rectangles have been appended  
17 | from the current point, the driver MUST setmake the starting point of the last rounded rectangle appended as the new current  
18 | point.★カレントポイントは更新するのか？

19 | The direction of the paths of each rounded rectangle MUST be treated as follow the conventions as specified for the  
20 | opvpRectanglePath() function.

### 21 Structures

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

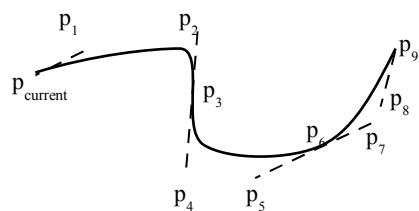
### 27 Return Value

28 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
29 | errorneopvpErrorNo.

## 30 4.6.13.opvpBezierPath

### 31 Name

32 | opvpBezierPath – Adds 3D Bezier paths to the current path.



In the above case, each arguments are as following:

npoints = 9

\*points = {p<sub>1</sub>, p<sub>2</sub>, ... p<sub>9</sub>}

### 33 Synopsis

```
34 opvp_result_t opvpBezierPath(
35     opvp_dc_t printerContext,
```

```
36     opvp_int_t npoints,
```

```
37     opvp_point_t *points);
```

### 38 Arguments

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

41 | npoints – Number of the points array elements in points. It MUST be multiple number of 3.

42 | points – Pointer to the array of end points and control points of bezierBézier curves.

### 43 Description

44 | This function adds multiple bezierBézier paths specified by the current point and end points and the control points given by the  
45 | points array. The first bezierBézier path is started by the current point and the third point in the points array is the end  
46 | points. The first point and second point in the points array is the control points of the first bezierBézier path. Following  
47 | bezierBézier paths are started by the previous path's end point and the following two points are the control points and the next  
48 | point is the end points.

1 | After driver appends bezierBézier paths from the current point, driver MUST set the last point of the `points` array as the  
2 | current point.  
3 | If `npoints` is not a multiple number of 3, this function MUST return error and set the error code `OPVP_PARAMERROR` in  
4 | erroropvpErrorNo.

5 | **Return Value**

6 | `OPVP_OK` or -1 whenin case of error. In the latter case, and the driver MUST store the detailed error code in  
7 | erroropvpErrorNo.

8 | **4.6.14.opvpArcPath**

9 | **Name**

10 | `opvpArcPath` – Adds arcs, chords, pies, or ellipses to the current path.

11 | **Synopsis**

```
12 |     opvp_result_t opvpArchPath(  
13 |         opvp_dc_t printerContext,  
14 |         opvp_arcmode_t kind,  
15 |         opvp_arcdir_t dir,  
16 |         opvp_fix_t bbx0, opvp_fix_t bby0, opvp_fix_t bbx1, opvp_fix_t bby1,  
17 |         opvp_fix_t x0, opvp_fix_t y0,  
18 |         opvp_fix_t x1, opvp_fix_t y1);
```

19 | **Arguments**

20 | `printerContext` – Printer context value returned by the `opvpOpenPrinter()` function.  
21 | `kind` – Arc kind flag. `OPVP_ARC(Arc)`, `OPVP_CHORD(Chord)`, `OPVP_PIE(Pie)` can be set.  
22 | `dir` – Direction of the path. `OPVP_CLOCKWISE(Clockwise)`, `OPVP_COUNTERCLOCKWISE(Counter-clockwise)` can be set.  
23 | `bbx0`, `bby0`, `bbx1`, `bby1` – Circumscribe rectangle.  
24 | `x0`, `y0` – Starting point.  
25 | `x1`, `y1` – End point.三原さんの図★図が欲しいが。。

26 | **Description**

27 | 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  
28 | 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  
29 | start and end points, driver MUST append ellipse into the path. If the circumscribe rectangle is a square, driver MUST adds  
30 | circle to the path.

31 | 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,  
32 | driver MUST set the left-top point of the circumscribe rectangle as the current point.

33 | **Return Value**

34 | `OPVP_OK` or -1 whenin case of error. In the latter case, and the driver MUST store the detailed error code in  
35 | erroropvpErrorNo.

36

37

38 | ★Textは削除

## 1 | 4.7.Bitmap Image Operations

2 | This section defines functions for bitmap operations.

3 | A bitmap is a pixel oriented image format for data which is drawn in a rectangle are images region. A bitmap is drawn  
4 | according to using the drawing attributes registered in the with a Graphics State Object.

5 | The typical bitmap drawing sequence is as follows:

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

9 | If caller calls any functions except opvpTransferDrawImage() function between the opvpStartDrawImage() and  
10 | opvpEndDrawImage() functions, driver MUST return error and set the error code OPVP\_BADREQUEST in  
11 | errorneopvpErrorNo.

12 | The opvpDrawImage() is the function for that performings the opvpStartDrawImage(),  
13 | opvpTransferDrawImage() and opvpEndDrawImage() function calls by one function call.

14 | All bitmap operation functions are OPTIONAL.

### 15 | 4.7.1.opvpDrawImage

#### 16 | Name

17 | opvpDrawImage – Draw a bitmap image

#### 18 | Synopsis

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

#### 30 | Arguments

31 | printerContext – Printer context value returned by opvpOpenPrinter() function.  
32 | sourceWidth – Width (pixels) of source image.  
33 | sourceHeight – Height (pixels) of source image.  
34 | sourcePitch – Number of Repetition pitch (bytes) of each raster of source image.★4の倍数の必要はあるんだつけ?★三  
35 | 原さん図を入れる  
36 | colorDepth – Number of bits per pixel of source image.  
37 | imageFormat – Image format enum to specify the source image format.  
38 | destinationSize – Destination drawing size.★サイズなのに Rectangle?  
39 | destinationWidth – Destination drawing width (pixels).  
40 | destinationHeight – Destination drawing height (pixels).  
41 | imageData – Pointer to actual source image data.

#### 42 | Description

43 | This function draws a bitmap image. Destination image drawing area is specified by both the current point kept in the Graphics  
44 | State Object and destinationSize. The upper-left corner of the image MUST be drawn on the current point and the  
45 | bottom-right corner of the image MUST be drawn at (x<sub>current</sub> + destinationWidth - 1, y<sub>current</sub> + destinationHeight - 1) where x<sub>current</sub>  
46 | is the current point x coordinate value and y<sub>current</sub> is the current point y coordinate value. width, height を足したところ★計算式  
47 | が必要はどこに?

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

1 | colorになる——Bwのイメージでビット1は foregroundとして RAWのみサポート予約★capabilityで取得できるのか？現在  
2 | は iformatRawしかないのであれば、現バージョンはそれのみで、後は予約でよいのでは？Image format can be used which  
3 | the device supports. (OPVP\_IFORMAT\_RAW, OPVP\_IFORMAT\_RLE, OPVP\_IFORMAT\_JPEG, OPVP\_IFORMAT\_PNG or  
4 | the vendor supports uniquely can be specified)

5 | To specify the color space of the image, caller MUST call the opvpSetColorSpace() function before calling this function.  
6 | Color space enum which is set to the opvpSetColorSpace() function MUST be one of the color space enum that the  
7 | driver supports.★これでよいか確認

8 | Only OPVP\_IFORMAT\_RAW MUST be supported by driver, and caller MUST set OPVP\_IFORMAT\_RAW in imageFormat.  
9 | Other image format enum (OPVP\_IFORMAT\_JPEG, OPVP\_IFORMAT\_PNG, OPVP\_IFORMAT\_RLE, etc) are reserved for  
10 | future use.

11 | After drawing image, driver MUST not change the current point.★一応確認

## 12 | **Return Value**

13 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in [errornoopvpErrorNo](#).

## 14 | **4.7.2.opvpStartDrawImage**

### 15 | **Name**

16 | opvpStartDrawImage – Start to draw a bitmap image

### 17 | **Synopsis**

```
18 |     opvp_result_t opvpStartDrawImage(  
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,  
25 |         opvp_rectangle_t destinationSize  
26 |         opvp_int_t destinationWidth,  
27 |         opvp_int_t destinationHeight,  
28 |     );
```

### 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 – Number of bytes of each line of source image.](#)★4の倍数の必要はあるんだつけ？

34 | [sourcePitch – Repetition pitch \(bytes\) of source image.](#)

35 | colorDepth – Number of bits per pixel of source image.

36 | imageFormat – Image format enum to specify the source image format.

37 | ★サイズなのに Rectangle ? DrawImage に同じ destinationSize – Destination drawing size.

38 | [destinationWidth – Destination drawing width \(pixels\).](#)

39 | [destinationHeight – Destination drawing height \(pixels\).](#)

### 40 | **Description**

41 | This function starts to draw a bitmap image. Destination image drawing area is specified by both the current point kept in the  
42 | Graphics State Object and destinationSize. The upper-left corner of the image MUST be drawn on the current point and  
43 | the bottom-right corner of the image – [DrawImage に同じ★計算式. どこに？ MUST be drawn at \(x<sub>current</sub> + destinationWidth -  
44 | 1, y<sub>current</sub> + destinationHeight - 1\) where x<sub>current</sub> is the current point x coordinate value and y<sub>current</sub> is the current point y coordinate  
45 | value.](#)

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

47 | After caller calls this function, caller MUST call opvpTransferDrawImage() function once or more times to transfer the  
48 | actual image data to driver. If caller calls any functions except the opvpTransferDrawImage() function between the  
49 | opvpStartDrawImage() and opvpEndDrawImage() function, driver MUST return error and set the error code  
50 | OPVP\_BADREQUEST in [errornoopvpErrorNo](#).

1 | Image format can be used which the device supports. (OPVP\_IFORMAT\_RAW, OPVP\_IFORMAT\_RLE,  
2 | OPVP\_IFORMAT\_JPEG, OPVP\_IFORMAT\_PNG or the vendor supports uniquely can be specified) ★capabilityで取得できる  
3 | のか? 現在はifromatRawしかないのであれば、現バージョンはそれのみで、後は予約でよいのでは?  
4 | To specify the color space of the image, caller MUST call the [opvpSetColorSpace\(\)](#) function before calling this function.  
5 | Color space enum which is set to the [opvpSetColorSpace\(\)](#) function MUST be one of the color space enum that the  
6 | driver supports. ★これでよいか確認  
7 | Only OPVP\_IFORMAT\_RAW MUST be supported by driver, and caller MUST set OPVP\_IFORMAT\_RAW in [imageFormat](#).  
8 | Other image format enum (OPVP\_IFORMAT\_JPEG, OPVP\_IFORMAT\_PNG, OPVP\_IFORMAT\_RLE, etc) are reserved for  
9 | future use.  
10 | After drawing image, driver MUST not change the current point. ★一応確認

## 11 | **Return Value**

12 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in [erroneopvpErrorNo](#).

## 13 | **4.7.3.opvpTransferDrawImage**

### 14 | **Name**

15 | opvpTransferDrawImage – Transfer the actual bitmap image data

### 16 | **Synopsis**

17 | 

```
opvp_result_t opvpTransferDrawImage (
18 |     opvp_dc_t printerContext,
19 |     opvp_int_t count,
20 |     void *imageData);
```

### 21 | **Arguments**

22 | printerContext – Printer context value returned by [opvpOpenPrinter\(\)](#) function.

23 | count – Number of image bytes to transfer.

24 | imageData – Pointer to actual source image data.

### 25 | **Description**

26 | This function transfers a bitmap image data. The image data MUST be declared to start by the [opvpStartDrawImage\(\)](#)  
27 | function before calling this function. エラーの場合はEndを呼ぶMUST

28 | When this function returns error, caller MUST call the [opvpEndDrawImage\(\)](#) function before calling other functions.

### 29 | **Return Value**

30 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in [erroneopvpErrorNo](#).

## 31 | **4.7.4.opvpEndDrawImage**

### 32 | **Name**

33 | opvpEndDrawImage – Declare the end of transferring the image data

### 34 | **Synopsis**

35 | 

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

### 37 | **Arguments**

38 | printerContext – Printer context value returned by [opvpOpenPrinter\(\)](#) function.

### 39 | **Description**

40 | This function declares the end of transferring the image data. ★DrawImage()はいらないよね?

### 41 | **Return Value**

42 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in [erroneopvpErrorNo](#).

## 1 | 4.8.Scan Line Operations

2 | This section defines functions for scan line operations.

3 | Scan line operations provide the support for drawing of horizontal lines drawing which are defined by one or multiple more pairs  
4 | of begin point and end points. Scan lines are drawn according to using the drawing attributes registered in the with a Graphics  
5 | State Object.

6 | Driver support for any of the scan line operations are OPTIONAL. However, support for scan line operations is  
7 | recommended REQUIRED- for drivers that do not support the that where in the ease path operations are not supported by driver,  
8 | and you caller want needs to apply different switch the color drawing schemes for from bitmap drawing and oriented to path  
9 | oriented drawing. 書き換え★意味が分からぬ、日本語版も不明 二の関数は今必要? 今後必要になった際に拡張するのでは?

11 | If the caller calls any functions except API entry other than opvpTransferScanline() function between the  
12 | opvpStartScanline() and opvpEndScanline() functions, the driver MUST return an error and set the detailed error  
13 | code to OPVP\_BADREQUEST in opvpErrorNo.

14 | その他の関数は呼び出し禁止★opvpStartScanline, opvpEndScanline の間で、opvpScanline 以外の関数はコールできる?

15 |

### 16 | 4.8.1.opvpStartScanline

#### 17 | **Name**

18 | opvpStartScanline – Declares the start of a scan line drawing operation.

#### 19 | **Synopsis**

```
20 |     opvp_result_t opvpStartScanline(  
21 |         opvp_dc_t printerContext,  
22 |         opvp_int_t yposition);
```

#### 23 | **Arguments**

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

25 | yposition – Vertical position to draw the start a scan line.

#### 26 | **Description**

27 | This function declares the start of a scan line drawing operation. Scan lines are drawn by using three functions. The  
28 | opvpStartScanline() function declares the start of a scan lines drawing operation, the opvpScanline() function  
29 | transfers the actual scan lines data, and the opvpEndScanline() function terminates the scan line drawing operations. The  
30 | driver MUST not change the current point after these operations.

#### 31 | **Return Value**

32 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
33 | errorne opvpErrorNo.

### 34 | 4.8.2.opvpScanline

#### 35 | **Name**

36 | opvpScanline – Draws the scan lines.

● scan line begin/and pixels  
○ intermediate pixels

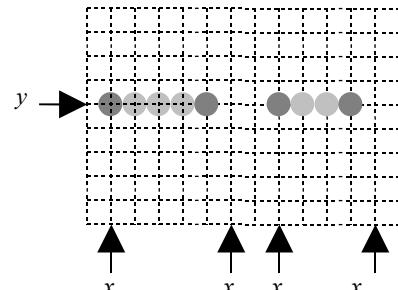
#### 37 | **Synopsis**

```
38 |     opvp_result_t opvpScanline(  
39 |         opvp_dc_t printerContext,  
40 |         opvp_int_t nscanpairs,  
41 |         opvp_int_t *scancpairs);
```

#### 42 | **Arguments**

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

44 | nscanpairs – Number of scan lines specified by begin and end position point  
45 | pairs



\*scancpairs = {{x0, x1}, {x2, x3}}

1 | scanpairs – Pointer to scan line array specified by of begin and end x-positionpoint pairs

## 2 | **Description**

3 | This function draws the scan lines. After the caller calls this function, the driver MUST increment the y coordinate value of the  
4 | current pointscan line by 1. The current scan line's y coordinate value of the current scan line is different independent from the  
5 | current point y coordinate value of the current point., and The driver MUST keep maintain the current scan line's y coordinate in  
6 | the Graphics State Object value temporarily between calls to the opvpStartScanLine () and opvpEndScanLine ()  
7 | functions in the Graphics State Object. The driver MUST not change the current point after calling when this function is  
8 | called. 前の説明(英語版)と矛盾してないか? 良い★更新しなくてよいのか?

9 | Y-coordinate value と current point の違いを追記

## 10 | **Return Value**

11 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
12 | erroropvpErrorNo.

## 13 | **4.8.3.opvpEndScanline**

### 14 | **Name**

15 | opvpEndScanline – Terminates the scan line drawingoperation.

### 16 | **Synopsis**

17 | opvp\_result\_t opvpEndScanline(  
18 |       opvp\_dc\_t printerContext);

### 19 | **Arguments**

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

### 21 | **Description**

22 | This function terminates the scan line drawingoperation.

### 23 | **Return Value**

24 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
25 | erroropvpErrorNo.

## 1 4.9.Raster Image Operations

2 | This section defines functions for raster image operations.

3 | Raster image operations provide the support for raster image drawing. AllDriver support for any of the raster image operations  
4 | are OPTIONAL. However, in case of the driver does not supports neither bitmap image operations nor path operations,  
5 | the these raster image operations are MUST be supported.

6 | If the caller calls any functions API entry except other than opvpTransferRasterData() or opvpSkipRaster()  
7 | functions between the opvpStartRaster() and opvpEndRaster() functions, the driver MUST return an error and set  
8 | the detailed error code to OPVP\_BADREQUEST in opvpErrorNo.

9 | 他の関数禁止★StartRaster()とEndRasterの間、他の関数を呼んで良いのか？

10 | ★このような書き方でよい？

11 |

### 12 | 4.9.1.opvpStartRaster

#### 13 | Name

14 | opvpStartRaster – Declare to start a raster image drawing

#### 15 | Synopsis

16 | opvp\_result\_t opvpStartRaster(  
17 |     opvp\_dc\_t printerContext,  
18 |     opvp\_int\_t rasterWidth);

#### 19 | Arguments

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

21 | rasterWidth – Width (pixels) of actual raster image (MUST not included padding data)★実装とあってる？

#### 22 | Description

23 | This function declares to start a raster data drawing.

24 | Driver MUST draw the raster image from the current point.

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

26 | Bit and/or byte order is also determined by the value set in the current Graphics State Object.★どこに定義？

27 | ★三原さんの図

28 | MSB\_Byte オーダー

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

30 | 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	bit order required★どこに定義？ 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	byte order required★どこに定義？
OPVP_CSPACE_DEVICECMY	3	8	24	3	
OPVP_CSPACE_DEVICECMYK	4	8	32	4	
emapIndexed	+	4,8	4,8	1/2,+	bit order required★ない？

#### 31 | Return Value

32 | OPVP\_OK or -1 when error , and the driver MUST store the detailed error code in errornoopvpErrorNo.

## 1    4.9.2.opvpTransferRasterData

### 2    Name

3 | opvpTransferRasterData – Transfers raster image data.

### 4    Synopsis

```
5    opvp_result_t opvpTransferRasterData(
6        opvp_dc_t printerContext,
7        opvp_int_t count,
8        opvp_byte_t *data);
```

### 9    Arguments

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

11 | count – Number of pixel data elements. ★ピクセル数？現状の実装確認

12 | data – Pointer to pixel data array.

### 13    Description

14 | This function transfers count pixel data elements of a single row of raster data. The dDriver MUST ★どうやって判断するのか？ data non-existent if as ピクセルは描画しない data 残りの treat the which exceeds the given count, not draw the remaining data, driver MUST the rasterWidth width given by opvpStartRaster() function スタートラスターで指定された幅 pixel data elements of data the number of count than sserel greater the s. If ealler set count スタートラスターで指定された幅 by opvpStartRaster() function given pixels rasterWidth the the exceeds when the data pixel length which ignore the given data only transfer the lesser of count and rasterWidth pixels. If count is less than rasterWidth, the driver MUST not transfer “filler” pixels. In case count exceeds rasterWidth, the driver MUST ignore the additional pixels.

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

### 23    Return Value

24 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in errorneopvpErrorNo.

## 26    4.9.3.opvpSkipRaster

### 27    Name

28 | opvpSkipRaster – Skips lines during raster image drawing.

### 29    Synopsis

```
30    opvp_result_t opvpSkipRaster(
31        opvp_dc_t printerContext,
32        opvp_int_t count);
```

### 33    Arguments

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

35 | count – Number of lines to be skipped.

### 36    Description

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

### 39    Return Value

40 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in errorneopvpErrorNo.

## 42    4.9.4.opvpEndRaster

### 43    Name

44 | opvpEndRaster – Terminates the raster image drawing operation.

```
1 Synopsis
2     opvp_result_t opvpEndRaster(
3         opvp_dc_t printerContext);
4 Arguments
5 |     printerContext – Printer context value returned by the opvpOpenPrinter() function.
6 Description
7 |     This function terminates the raster image drawing operation.
8 Return Value
9 |     OPVP_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in
10 |     erroneopvpErrorNo.
```

## 1 4.10.Stream Data Operations

2 | This section defines functions for stream data operations.

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

7 | If the caller calls any functions except API entries other than opvpTransferStreamData() - function between the opvpStartStream() and opvpEndStream() functions, the driver MUST return an error and set the detailed error code to OPVP\_BADREQUEST in opvpErrorNo.

10 | ?StartStream, EndStream の間では、他の関数はコールできない★本当にこの関数は必要なのか？

11 |  
12 |

### 13 4.10.1.opvpStartStream

#### 14 Name

15 | opvpStartStream – Declares the start of a streaming data transfer.

#### 16 Synopsis

17 | opvp\_result\_t opvpStartStream(  
18 |       opvp\_dc\_t printerContext);

#### 19 Arguments

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

#### 21 Description

22 | This function declares the starts of a streaming data transfer. Streaming data is the data that is directly sent to the printer device directly without being any processed by the driver.

#### 24 Return Value

25 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
26 | errorneopvpErrorNo.

### 27 4.10.2.opvpTransferStreamData

#### 28 Name

29 | opvpTransferStreamData – sends stream data.

#### 30 Synopsis

31 | opvp\_result\_t opvpTransferStreamData(  
32 |       opvp\_dc\_t printerContext,  
33 |       opvp\_int\_t count,  
34 |       void \*data);

#### 35 Arguments

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

37 | count – Number of data bytes to transfer.

38 | data – Pointer to the stream data.

#### 39 Description

40 | This function sends the streaming data directly to the printer device.

#### 41 Return Value

42 | OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
43 | errorneopvpErrorNo.

## 1    **4.10.3.opvpEndStream**

### 2    **Name**

3 |    opvpEndStream – Declares the end of a streaming data transfer.

### 4    **Synopsis**

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

### 7    **Arguments**

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

### 9    **Description**

10 |    This function terminates the streaming data transfer.

### 11   **Return Value**

12 |    OPVP\_OK or -1 when in case of error. In the latter case, and the driver MUST store the detailed error code in  
13 |    errerno opvpErrorNo.

## 5. Graphic Operation Fallback

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

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

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

**Thus, this specification defines the minimum set of APIs that driver must prepare to help caller guaranty the “Caller Fallback” process in the caller side implementation.**

**The minimum set APIs is as following.**

—

★原文の意味が良く分からないので大幅書き直し、これでよいか？どの関数を最小セットとするか？

—

# 1 | 6.Macros, Types, Enumerations and Structures

---

## 1 | 6.1.Return Values

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

## 1 | 6.2.Error Codes

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

## 1 | 6.3.Basic Types

```
2 | typedef int opvp_dc_t;           /* driver/device context */
3 | typedef int opvp_result_t;        /* return value */
4 | typedef unsigned char opvp_byte_t; /* BYTE */
5 | typedef unsigned char opvp_char_t; /* character (string) */
6 | typedef int opvp_int_t;          /* integer */
7 | typedef int opvp_fix_t;          /* fixed integer */
8 | typedef float opvp_float_t;       /* float */
9 | typedef unsigned int opvp_flag_t; /* flags */
10 | typedef unsigned int opvp_rop_t;  /* raster operation */

11 | typedef struct _opvp_point {
12 |     opvp_fix_t x, y;
13 | } opvp_point_t;

14 | typedef struct _opvp_rectangle {
15 |     opvp_point_t p0;             /* start point */
16 |     opvp_point_t p1;             /* diagonal point */
17 | } opvp_rectangle_t;

18 | typedef struct _opvp_roundrectangle {
19 |     opvp_point_t p0;            /* start point */
20 |     opvp_point_t p1;            /* diagonal point */
21 |     opvp_fix_t xellipse, yellipse;
22 | } opvp_roundrectangle_t;
```

## 1 | 6.4.Image Formats

```
2 | typedef enum _opvp_imageformat {
3 |     OPVP_FORMAT_RAW            = 0,
4 |     OPVP_FORMAT_RLE             = 1,
5 |     OPVP_FORMAT_JPEG            = 2,
6 |     OPVP_FORMAT_PNG             = 3
7 | } opvp_imageformat_t;
```

## 1 | 6.5.Color Presentation

```
2 | typedef enum _opvp_colormapping {
3 |     OPVP_CMAP_DIRECT            = 0,
4 |     OPVP_CMAP_INDEXED           = 1
5 | } opvp_colormapping_t;

6 | typedef enum _opvp_cspace {
7 |     OPVP_CSPACE_BW              = 0,
8 |     OPVP_CSPACE_DEVICEGRAY      = 1,
9 |     OPVP_CSPACE_DEVICECMY       = 2,
10 | }
```

```

1     OPVP_CSPACE_DEVICECMYK      = 3,
2     OPVP_CSPACE_DEVICERGB       = 4,
3     OPVP_CSPACE_DEVICEKRGB      = 5,
4     OPVP_CSPACE_STANDARDRGB     = 6,
5     OPVP_CSPACE_STANDARDRGB64   = 7
6 } 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.ellaneous Flags

```

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

```

```

1         OPVP_PIE          = 2
2 } opvp_arcmode_t;
3
4 typedef enum _opvp_arcdir {
5     OPVP_CLOCKWISE      = 0,
6     OPVP_COUNTERCLOCKWISE = 1
7 } opvp_arcdir_t;
8
9 typedef enum _opvp_pathmode {
10    OPVP_PATHCLOSE      = 0,
11    OPVP_PATHOPEN        = 1
12 } 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    OPVP_QF_MEDIASIZE        = 0x00010000 /* only for opvpQueryDeviceInfo */★これも？
12    OPVP_QF_PRINTREGION      = 0x00020000 /* only for opvpQueryDeviceInfo */
13    OPVP_QF_MEDIACOPY         = 0x00000100 /* Maximum copy number supported */
14    OPVP_QF_PRINTREGION      = 0x00010000 /* only for opvpQueryDeviceInfo use */
15 } opvp_queryinfoflags_t;

```

## 1 6.12.Sample Header File

2 | [HereFollowing](#) is a sample header file which can be used by driver or and renderer implementations based on this specification.

```

3 /*
4 * OpenPrinting Vector Printer Driver API Definitions [opvp.h]
5 *
6 * Copyright (c) 2006 Free Standards Group
7 * Copyright (c) 2006 Fuji Xerox Printing Systems Co., Ltd.
8 * Copyright (c) 2006 Canon Inc.
9 * Copyright (c) 2003-2006 AXE Inc.★Copyrightはつけて良いか?
10 */
11
12 /* All Rights Reserved.
13 */
14
15 /* Permission to use, copy, modify, distribute, and sell this software
16 * and its documentation for any purpose is hereby granted without
17 * fee, provided that the above copyright notice appear in all copies
18 * and that both that copyright notice and this permission notice
19 * appear in supporting documentation.
20 */
21
22
23 /* The above copyright notice and this permission notice shall be
24 * included in all copies or substantial portions of the Software.
25 */
26
27 /* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
28 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
29 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
30 * NONINFRINGEMENT. IN NO EVENT SHALL THE OPEN GROUP BE LIABLE FOR
31 * ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
32 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
33 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
34 */

```

```

1  /*
2  */
3  2007 Modified for OPVP 1.0 by BBR Inc.
4  */
5
6 #ifndef _OPVP_H_
7 #define _OPVP_H_
8
9 /* Return Values and Error Codes */
10#define OPVP_OK 0 /* -1 for errors */
11#define OPVP_FATALERROR -1 /* error: cannot be recovered */
12#define OPVP_BADREQUEST -2 /* error: called where it should not be called */
13#define OPVP_BADCONTEXT -3 /* error: invalid printer context */
14#define OPVP_NOTSUPPORTED -4 /* error: combination of parameters are set */
15 | /* which that cannot be handled by driver or printer */
16 */
17#define OPVP_JOBCANCELED -5 /* error: job has been canceled by some cause */
18#define OPVP_PARAMERROR -6 /* error: invalid parameter */
19
20/* Basic Types */
21typedef int opvp_dc_t; /* driver/device context */
22 | typedef int opvp_result_t; /* return value */
23typedef unsigned char opvp_byte_t; /* BYTE */
24typedef unsigned char opvp_char_t; /* character (string) */
25typedef int opvp_int_t; /* integer */
26typedef int opvp_fix_t; /* fixed integer */
27typedef float opvp_float_t; /* float */
28 | typedef unsigned int opvp_flag_t; /* flags */
29typedef unsigned int opvp_rop_t; /* raster operation */
30
31/* for opvp_fix_t */
32#define OPVP_FIX_FRACT_WIDTH 8
33#define OPVP_FIX_FRACT_DENOM (1<<OPVP_FIX_FRACT_WIDTH)
34#define OPVP_FIX_FLOOR_WIDTH (sizeof(int)*8-OPVP_FIX_FRACT_WIDTH)
35
36/* convert macro */
37#define OPVP_I2FIX(i,fix) (fix=i<<OPVP_FIX_FRACT_WIDTH)
38#define OPVP_F2FIX(f,fix) (fix=((int)floor(f)<<OPVP_FIX_FRACT_WIDTH)\n
39 | | ((int)((f-floor(f))*OPVP_FIX_FRACT_DENOM)\n
40 | | &(OPVP_FIX_FRACT_DENOM-1)))
41
42/* graphic elements */
43typedef struct _opvp_point {
44    opvp_fix_t x, y;
45} opvp_point_t;
46
47typedef struct _opvp_rectangle {
48    opvp_point_t p0; /* start point */
49    opvp_point_t p1; /* diagonal point */
50} opvp_rectangle_t;
51
52typedef struct _opvp_roundrectangle {
53    opvp_point_t p0; /* start point */
54    opvp_point_t p1; /* diagonal point */
55    opvp_fix_t xellipse, yellipse;
56} opvp_roundrectangle_t;
57
58/* Image Formats */
59typedef enum _opvp_imageformat {
60    OPVP_IFORMAT_RAW = 0,
61    OPVP_IFORMAT_RLE = 1,
62    OPVP_IFORMAT_JPEG = 2,
63    OPVP_IFORMAT_PNG = 3
64} opvp_imageformat_t;
65
66/* Color Presentation */
67typedef enum _opvp_colormapping {
68    OPVP_CMAP_DIRECT = 0,
69    OPVP_CMAP_INDEXED = 1
70} opvp_colormapping_t;
71
72typedef enum _opvp_cspace {

```

```

1      OPVP_CSPACE_BW          = 0,
2      OPVP_CSPACE_DEVICEGRAY  = 1,
3      OPVP_CSPACE_DEVICECMY   = 2,
4      OPVP_CSPACE_DEVICECMYK  = 3,
5      OPVP_CSPACE_DEVICERGB   = 4,
6      OPVP_CSPACE_DEVICEKRGB  = 5,
7      OPVP_CSPACE_STANDARDRGB = 6,
8      OPVP_CSPACE_STANDARDRGB64 = 7
9 } opvp_cspace_t;

10 /* Fill, Paint, Clip */
11 typedef enum _opvp_fillmode {
12     OPVP_FILLMODE_EVENODD    = 0,
13     OPVP_FILLMODE_WINDING    = 1
14 } opvp_fillmode_t;

15 | typedef enum _opvp_paintmode {
16     OPVP_PAINTMODE_OPAQUE    = 0,
17     OPVP_PAINTMODE_TRANSPARENT = 1
18 } opvp_paintmode_t;

19 | typedef enum _opvp_clirule {
20     OPVP_CLIPRULE_EVENODD    = 0,
21     OPVP_CLIPRULE_WINDING    = 1
22 } opvp_clirule_t;

23 | /* Line */
24 | typedef enum _opvp_linestyle {
25     OPVP_LINESTYLE_SOLID      = 0,
26     OPVP_LINESTYLE_DASH       = 1
27 } opvp_linestyle_t;

28 | typedef enum _opvp_linecap {
29     OPVP_LINECAP_BUTT         = 0,
30     OPVP_LINECAP_ROUND        = 1,
31     OPVP_LINECAP_SQUARE       = 2
32 } opvp_linecap_t;

33 | typedef enum _opvp_linejoin {
34     OPVP_LINEJOIN_MITER       = 0,
35     OPVP_LINEJOIN_ROUND        = 1,
36     OPVP_LINEJOIN_BEVEL       = 2
37 } opvp_linejoin_t;

38 | /* Brush */
39 | typedef enum _opvp_bdtype {
40     OPVP_BTYPENORMAL         = 0
41 } opvp_bdtype_t;

42 | typedef struct _opvp_brushdata {
43     opvp_bdtype_t type;
44     opvp_int_t width, height, pitch;
45 #if defined(__GNUC__) && __GNUC__ <= 2
46     opvp_byte_t data[1];
47 #else
48     opvp_byte_t data[];
49 #endif
50 } opvp_brushdata_t;

51 | typedef struct _opvp_brush {
52     opvp_cspace_t colorSpace;
53     opvp_int_t color[4];※コメント削除
54     opvp_int_t xorg, yorg; /* brush origin */
55     /* ignored for opvpSetBgColor */
56     opvp_brushdata_t *pbrush; /* pointer to brush data */
57     /* solid brush used, if NULL */
58 } opvp_brush_t;

59 | /* Misc.ellaneous Flags */
60 | typedef enum _opvp_arcmode {
61     OPVP_ARC                = 0,
62
63
64
65
66
67
68
69
70
71
72

```

```

1      OPVP_CHORD          = 1,
2      OPVP_PIE           = 2
3 } opvp_arcmode_t;
4
5 typedef enum _opvp_arcdir {
6     OPVP_CLOCKWISE       = 0,
7     OPVP_COUNTERCLOCKWISE = 1
8 } opvp_arcdir_t;
9
10 typedef enum _opvp_pathmode {
11    OPVP_PATHCLOSE        = 0,
12    OPVP_PATHOPEN         = 1
13 } opvp_pathmode_t;
14
15 /* CTM */
16 typedef struct _opvp_ctm {
17     opvp_float_t a, b, c, d, e, f;
18 } opvp_ctm_t;
19
20 /* Device Information and Capabilities */
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     /* only for opvpQueryDeviceInfo */
31     OPVP_QF_PRINTREGION      = 0x00020000 /* only for opvpQueryDeviceInfo *//100-
32     0x000_SIZE_OPVP_QF_MEDIA
33     OPVP_QF_MEDIACOPY        = 0x00000100 /* Maximum copy number supported */※
34     OPVP_QF_PRINTREGION      = 0x00010000 /* only for opvpQueryDeviceInfo use */※
35 } opvp_queryinfoflags_t;
36
37 /* API Procedure Entries */
38 typedef struct _opvp_api_procs {
39     opvp_dc_t
40     (*opvpOpenPrinter)(opvp_int_t, opvp_char_t*, opvp_int_t*, opvp_int_t*, struct
41     _opvp_api_procs**);※引数削除
42     opvp_result_t (*opvpClosePrinter)(opvp_dc_t);
43     opvp_result_t (*opvpStartJob)(opvp_dc_t, opvp_char_t*);
44     opvp_result_t (*opvpEndJob)(opvp_dc_t);
45     opvp_result_t (*opvpAbortJob)(opvp_dc_t);
46     opvp_result_t (*opvpStartDoc)(opvp_dc_t, opvp_char_t*);
47     opvp_result_t (*opvpEndDoc)(opvp_dc_t);
48     opvp_result_t (*opvpStartPage)(opvp_dc_t, opvp_char_t*);
49     opvp_result_t (*opvpEndPage)(opvp_dc_t);
50     opvp_result_t
51     (*opvpQueryDeviceCapability)(opvp_dc_t, opvp_flag_t, opvp_int_t, opvp_byte_t*);
52     opvp_result_t
53     (*opvpQueryDeviceInfo)(opvp_dc_t, opvp_flag_t, opvp_int_t, opvp_char_t*);
54     opvp_result_t (*opvpResetCTM)(opvp_dc_t);
55     opvp_result_t (*opvpSetCTM)(opvp_dc_t, opvp_ctm_t*);
56     opvp_result_t (*opvpGetCTM)(opvp_dc_t, opvp_ctm_t*);
57     opvp_result_t (*opvpInitGS)(opvp_dc_t);
58     opvp_result_t (*opvpSaveGS)(opvp_dc_t);
59     opvp_result_t (*opvpRestoreGS)(opvp_dc_t);
60     opvp_result_t (*opvpQueryColorSpace)(opvp_dc_t, opvp_cspace_t*, opvp_int_t*), 引
61     opvp_result_t (*opvpQueryColorSpace)(opvp_dc_t, opvp_int_t*, opvp_cspace_t*); 引
62     数順番変更※
63     opvp_result_t (*opvpSetColorSpace)(opvp_dc_t, opvp_cspace_t);
64     opvp_result_t (*opvpGetColorSpace)(opvp_dc_t, opvp_cspace_t*);
65     opvp_result_t (*opvpQueryROP)(opvp_dc_t, opvp_int_t*, opvp_rop_t*);
66     opvp_result_t (*opvpSetROP)(opvp_dc_t, opvp_rop_t);
67     opvp_result_t (*opvpGetROP)(opvp_dc_t, opvp_rop_t*);
68     opvp_result_t (*opvpSetFillMode)(opvp_dc_t, opvp_fillmode_t);
69     opvp_result_t (*opvpGetFillMode)(opvp_dc_t, opvp_fillmode_t*);
70     opvp_result_t (*opvpSetAlphaConstant)(opvp_dc_t, opvp_float_t);
71     opvp_result_t (*opvpGetAlphaConstant)(opvp_dc_t, opvp_float_t);
72

```

```

1     opvp_result_t (*opvpSetLineWidth) (opvp_dc_t,opvp_fix_t);
2     opvp_result_t (*opvpGetLineWidth) (opvp_dc_t,opvp_fix_t*);
3     opvp_result_t (*opvpSetLineDash) (opvp_dc_t,opvp_fix_t*,opvp_int_t);
4     opvp_result_t (*opvpGetLineDash) (opvp_dc_t,opvp_fix_t*,opvp_int_t*);
5     opvp_result_t (*opvpSetLineDashOffset) (opvp_dc_t,opvp_fix_t);
6     opvp_result_t (*opvpGetLineDashOffset) (opvp_dc_t,opvp_fix_t*);
7     opvp_result_t (*opvpSetLineStyle) (opvp_dc_t,opvp_linestyle_t);
8     opvp_result_t (*opvpGetLineStyle) (opvp_dc_t,opvp_linestyle_t*);
9     opvp_result_t (*opvpSetLineCap) (opvp_dc_t,opvp_linecap_t);
10    opvp_result_t (*opvpGetLineCap) (opvp_dc_t,opvp_linecap_t*);
11    opvp_result_t (*opvpSetLineJoin) (opvp_dc_t,opvp_linejoin_t);
12    opvp_result_t (*opvpGetLineJoin) (opvp_dc_t,opvp_linejoin_t*);
13    opvp_result_t (*opvpSetMiterLimit) (opvp_dc_t,opvp_fix_t);
14    opvp_result_t (*opvpGetMiterLimit) (opvp_dc_t,opvp_fix_t*);
15    opvp_result_t (*opvpSetPaintMode) (opvp_dc_t,opvp_paintmode_t);
16    opvp_result_t (*opvpGetPaintMode) (opvp_dc_t,opvp_paintmode_t*);
17    opvp_result_t (*opvpSetStrokeColor) (opvp_dc_t,opvp_brush_t*);
18    opvp_result_t (*opvpSetFillColor) (opvp_dc_t,opvp_brush_t*);
19    opvp_result_t (*opvpSetBgColor) (opvp_dc_t,opvp_brush_t*);
20    opvp_result_t (*opvpNewPath) (opvp_dc_t);
21    opvp_result_t (*opvpEndPath) (opvp_dc_t);
22    opvp_result_t (*opvpStrokePath) (opvp_dc_t);
23    opvp_result_t (*opvpFillPath) (opvp_dc_t);
24    opvp_result_t (*opvpStrokeFillPath) (opvp_dc_t);
25    opvp_result_t (*opvpSetClipPath) (opvp_dc_t,opvp_cliprule_t);
26    opvp_result_t (*opvpResetClipPath) (opvp_dc_t);
27    opvp_result_t (*opvpSetCurrentPoint) (opvp_dc_t,opvp_fix_t,opvp_fix_t);
28    opvp_result_t
29    (*opvpLinePath) (opvp_dc_t,opvp_pathmode_t,opvp_int_t,opvp_point_t*);
30    opvp_result_t
31    (*opvpPolygonPath) (opvp_dc_t,opvp_int_t,opvp_int_t*,opvp_point_t*);
32    opvp_result_t (*opvpRectanglePath) (opvp_dc_t,opvp_int_t,opvp_rectangle_t*);
33    opvp_result_t
34    (*opvpRoundRectanglePath) (opvp_dc_t,opvp_int_t,opvp_roundrectangle_t*);
35    opvp_result_t (*opvpBezierPath) (opvp_dc_t,opvp_int_t,opvp_point_t*);
36    opvp_result_t
37    (*opvpArcPath) (opvp_dc_t,opvp_arcmode_t,opvp_arcdir_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t);
38    opvp_result_t
39    (*opvpDrawImage) (opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imageformat_t,opvp_rectangle_t,void*);
40    opvp_result_t
41    (*opvpStartDrawImage) (opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imageformat_t,opvp_rectangle_t);
42    opvp_result_t (*opvpTransferDrawImage) (opvp_dc_t,opvp_int_t,void*);
43    opvp_result_t (*opvpEndDrawImage) (opvp_dc_t);
44    opvp_result_t (*opvpStartScanline) (opvp_dc_t,opvp_int_t);
45    opvp_result_t (*opvpScanline) (opvp_dc_t,opvp_int_t,opvp_int_t*);
46    opvp_result_t (*opvpEndScanline) (opvp_dc_t);
47    opvp_result_t (*opvpStartRaster) (opvp_dc_t,opvp_int_t);
48    opvp_result_t (*opvpTransferRasterData) (opvp_dc_t,opvp_int_t,opvp_byte_t*);
49    opvp_result_t (*opvpSkipRaster) (opvp_dc_t,opvp_int_t);
50    opvp_result_t (*opvpEndRaster) (opvp_dc_t);
51    opvp_result_t (*opvpStartStream) (opvp_dc_t);
52    opvp_result_t (*opvpTransferStreamData) (opvp_dc_t,opvp_int_t,void*);
53    opvp_result_t (*opvpEndStream) (opvp_dc_t);
54
55
56
57 } opvp_api_procs_t;
58
59 /* Function prototype */
60 opvp_dc_t opvpOpenPrinter(
61     opvp_int_t outputFD,
62     opvp_char_t *printerModel,
63     opvp_int_t apiVersion[2],
64     opvp_int_t *nApiEntry,
65     opvp_api_procs_t **apiProcs);
66
67 /* error no */
68 extern opvp_int_t opvpErrorNo;
69
70 #endif /* _OPVP_H_ */

```

## 7. Authors and Contributors

### 7.1. Editors

Osamu Mihara – Fuji Xerox Co., Ltd.

### 7.2. Authors

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

### 7.3. Contributors

(~~a~~Alphabetical order) Hidekazu Hagiwara (MintWave), Masaki Iwata (AXE), Hidenori Kanjo (BBR), Shinpei Kitayama (~~Epson~~~~Kowa~~~~Avasy~~~~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), Akiko Maruyama (Ricoh), ~~Olaf Meeuwissen (EPSON AVASYS)~~, ★~~小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 (~~AXEBBR~~), Kenji Wakabayashi (MintWave), Toshihiro Yamagishi (Turbolinux), Akira Yoshiyama (NEC)

## 8.History

---

- 1  
2 Version 0.1 (Japanese) Mihara/Toratani  
3 Version 0.2 (Japanese) Mihara  
4 |     Openprinting-0.1.1 implementation 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 ~~few~~ 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/consts/enumerations/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 RC23 2007-5-919 Toratani  
31 |      Append "opvp" or "OPVP" to function names, enumerations, 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 format.  
34 |      Merge the RC2 2006-6-9 – 2006-12-05 updates made by Mihara  
35  
36