OpenPrinting

Vector Printer Driver Application Program Interface (OPVP) Specification

Version-1.0 RC3

(2007-5-20)

OpenPrinting Vector Printer Driver Application Interface Specification

Table of Contents

1

1.Notation and Terminology

1.1.Notation Conventions

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

Fond and Style	Description	Examples

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

	Function parameters	`printerContext`

	Source code examples.	`#ifndef _OPVP_H_`
		`#define _OPVP_H_`

Italic	Coordinate values (x, y)	*(x0, y0)*

1.2.Conformance Terminology

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

2.Introduction

2.1.Driver and Caller

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

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

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

The word “caller” is a program which calls a driver via the API defined in this specification. For example, open source 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.

2.2.Loading and Calling Printer Driver

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

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

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

2.3.Printer Driver Database

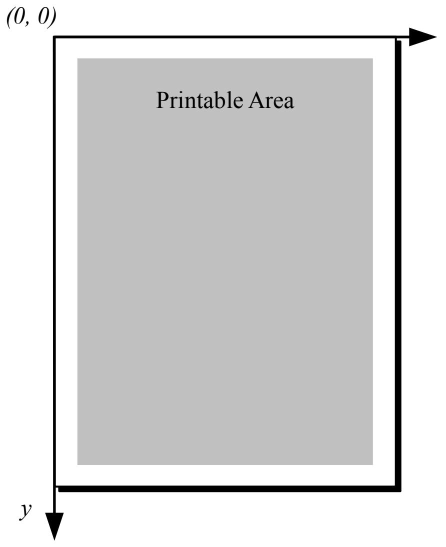
Printer driver name and its model name are managed by driver data base. The driver data base should be supplied in UPDF and/or PPD format.

2.4.Notes about Function Parameters

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

3.Graphic Model

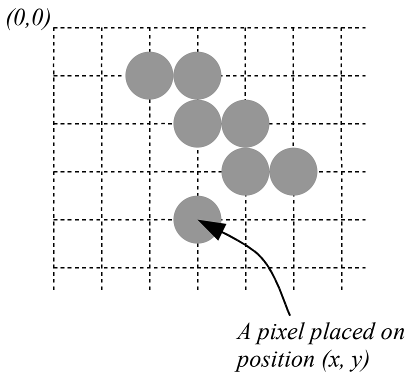
3.1.Coordinate System



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

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

Coordinate value x and y define each horizontal and vertical distance from the origin to the point on the media. The point which has the integer coordinate value x and y is on the intersection of the coordinate system grid, and 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 shown in the right down figure.



3.2.Objects

Following kinds of graphics objects are handled in this document.

- Path
- Bitmap Image
- Scanline
- Raster Image

3.3.Graphics State Object

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

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

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

Drawing Attributes	Related functions
ColorSpace	`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 pattern	`opvpSetLineDash(), opvpGetLineDash()`
Line dash pattern offset	`opvpSetLineDashOffset(), opvpGetLineDashOffset()`
Line style	`opvpSetLineStyle(), opvpGetLineStyle()`
Line cap style	`opvpSetLineCap(), opvpGetLineCap()`
Line join style	`opvpSetLineJoin(), opvpGetLineJoin()`
Miter limit value	`opvpSetMiterLimit(), opvpGetMiterLimit()`
Painting mode	`opvpSetPaintMode(), opvpGetPaintMode()`
Stroke line color	`opvpSetStrokeColor()`
Fill color	`opvpSetFillColor()`
Background color	`opvpSetBgColor()`

3.4.CTM

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

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

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

3.5.Color

Following color spaces are defined in this document.

Color Space Macro	Color Space
`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
`OPVP_CSPACE_STANDARDRGB`	sRGB
`OPVP_CSPACE_STANDARDRGB64`	scRGB

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

3.6.Scan Rule

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

4.Operations

4.1.Creating and Managing Print Contexts

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

4.1.1.opvpOpenPrinter

Name

opvpOpenPrinter – Create printer context.

Synopsis

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

Arguments

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

`printerModel` – Printer Model Name in UTF-8.

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

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

Description

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

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

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

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

Printer driver MUST return a printer context as the return value of this function. The printer context MUST be a unique number which is managed by the driver. Caller sets the printer context to the first parameter to call other APIs.

Return Value

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

4.1.2.opvpClosePrinter

Name

opvpClosePrinter – Delete printer context.

Synopsis

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

Arguments

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

Description

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

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

Return Value

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

4.2.Job, Document and Page Operations

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

4.2.1.opvpStartJob

Name

opvpStartJob – Declare to start a printing job.

Synopsis

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

Arguments

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

`jobInfo` – Printing job property string.

Description

This function declares to start a printing job.

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

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

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

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

Return Value

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

4.2.2.opvpEndJob

Name

opvpEndJob –Declare to terminate a printing job.

Synopsis

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

Arguments

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

Description

This function declares to terminate the printing job.

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

Return Value

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

4.2.3.opvpAbortJob

4.2.3.1.Name

fsgpdAbortJob – Declare to abort the printing job.

Synopsis

```
opvp_result_t fsgpdAbortJob(
        opvp_dc_t printerContext);
```

Arguments

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

Description

This function cleans up the printing operations to abort the print job. Driver MAY create a print data to clean up the printer printing status and MAY send the data to the printer. However, because whether driver sends the printing job data to the printer before sending the aborting data depends on the timing when caller calls this function, so calling this function does not guarantee that no papers are wasted by the printer. However, after caller calls this function, driver and printer MUST become in the initial state and next printing job MUST be accepted normally.★文章を変更、これでよいか？

★EndJobは呼ぶ必要は無いのか？明記する必要あり

Return Value

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

4.2.4.opvpStartDoc

Name

opvpStartDoc – Declare to start a printing document.

Synopsis

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

Arguments

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

`docInfo` – Printing document property string.

Description

This function declares to start a printing document.

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

Caller can set document properties into `jobInfo`. Printer driver MUST keep the given document properties until the `opvpEndDoc()` function is called. When caller calls the `opvpStartDoc()` function again after calling the `opvpEndDoc()` function, driver MUST override the former `docInfo` by the new `docInfo` given by the latest `opvpStartDoc()` function call.

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

It depends on printer and driver capabilities whether nesting document (`opvpStartDoc()` and `opvpEndDoc()` functions can be called between the `opvpStartDoc()` and `opvpEndDoc()` functions are called) can be handled. If a printer or printer driver does not allow nesting document, the driver MUST return error and set the error code `OPVP_BADREQUEST` in `opvpErrorNo`.

Return Value

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

4.2.5.opvpEndDoc

Name

opvpEndDoc – Declare to terminate the printing document

Synopsis

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

Arguments

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

Description

This function declares to terminate the printing document.

If caller called the `opvpStartDoc()` function, the caller MUST call this function after finishing each document processing.

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

Return Value

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

4.2.6.opvpStartPage

Name

opvpStartPage – Declare to start a printing page.

Synopsis

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

Arguments

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

`pageInfo` – Printing page property string.

Description

This function declares to start a printing page.

Caller MUST call this function after calling the `opvpStartJob()` and `opvpStartDoc()` functions, and before calling other drawing APIs.

Caller can set page properties into `pageInfo`. Driver MUST keep the page properties until the `opvpEndPage()` is called. When the caller calls the `opvpStartPage()` function again after calling the `opvpEndPage()`, the driver MUST override the former `pageInfo` by the new `pageInfo` given by the latest `opvpStartPage()` function call.

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

Return Value

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

4.2.7.opvpEndPage

Name

opvpEndPage – Declare to terminate the printing page.

Synopsis

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

Arguments

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

Description

This function declares to terminate the printing page.

Caller MUST call this function after finishing each page processing.

Return Value

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

4.3.Query Operations

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

4.3.1.opvpQueryDeviceCapability

Name

opvpQueryDeviceCapability – Query device capabilities.

Synopsis

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

Arguments

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

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

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

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

Description

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

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

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

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

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

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

Return Value

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

4.3.2.opvpQueryDeviceInfo

Name

opvpQueryDeviceInfo – Query device information.

Synopsis

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

Arguments

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

`queryflag` – Flag which specifies the device information to query.

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

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

Description

This function queries the current setting of printer or driver information.

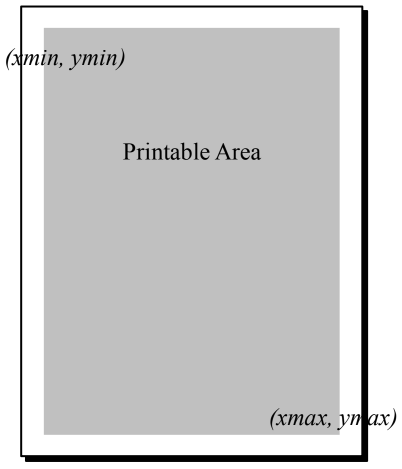
Caller MUST set one or more bit flags in `queryflag` to query information. Same enums which are used for `opvpQueryDeviceCapability()` function MUST be used.

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

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

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

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



Return Value

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

4.4.Attribute of Job, Document and Page Operations

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

The attribute names and values MUST be given in 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>* can be given with the separator ”,” (comma) for each <key>. If multiple values are given for one key, printer driver should search the values from the beginning of the first value in the multiple values for each key, and take the first possible value that the driver can use under the current setting.

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

Driver 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.

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

Major UPDF attributes are shown in the table below;

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

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

4.5.Graphics State Object Operations

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

4.5.1.opvpResetCTM

Name

opvpResetCTM – Initialize CTM of the Graphics State Object.

Synopsis

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

Arguments

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

Description

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

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

Return Value

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

4.5.2.opvpSetCTM

Name

opvpSetCTM – Set CTM to the Graphics State Object.

Synopsis

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

Arguments

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

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

Description

This function sets a CTM to the Graphics State Object.

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

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

Structures

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

Return Value

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

4.5.3.opvpGetCTM

Name

opvpGetCTM – Get CTM from the Graphics State Object.

Synopsis

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

Arguments

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

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

Description

This function gets CTM from the Graphics State Object.

Structures

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

Return Value

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

4.5.4.opvpInitGS

Name

opvpInitGS – Initialize parameters in the Graphics State Object.

Synopsis

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

Arguments

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

Description

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

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

Return Value

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

4.5.5.opvpSaveGS

Name

opvpSaveGS – Save the Graphics State Object parameters.

Synopsis

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

Arguments

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

Description

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

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 `opvpErrorNo`.

Return Value

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

4.5.6.opvpRestoreGS

Name

opvpRestoreGS – Restore the Graphics State Object parameters.

Synopsis

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

Arguments

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

Description

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

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 `opvpErrorNo`.

Return Value

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

4.5.7.opvpQueryColorSpace

Name

opvpQueryColorSpace – Query color spaces that the driver can support.

Synopsis

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

Arguments

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

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

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

Description

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

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

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

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

Driver returns the color space enum in the `pcspace` array by its preferable order. The first color space enum SHOULD specify that it is the most preferable color space for the driver.

Return Value

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

4.5.8.opvpSetColorSpace

Name

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

Synopsis

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

Arguments

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

`cspace` – Color space enum.

Description

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

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

Return Value

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

4.5.9.opvpGetColorSpace

Name

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

Synopsis

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

Arguments

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

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

Description

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

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

Return Value

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

4.5.10.opvpQueryROP

Name

opvpQueryROP – Query ROPs that the driver can support.

Synopsis

```
opvp_result_t opvpQueryROP(
        opvp_dc_t printerContext,
        opvp_int_t *pnum,
        opvp_rop_t *prop);
```

Arguments

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

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

`prop` – Pointer to the ROP array.

Description

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

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

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

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

Return Value

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

4.5.11.opvpSetROP

Name

opvpSetROP – Set ROP mode into the Graphics State Object.

Synopsis

```
opvp_result_t opvpSetROP(
        opvp_dc_t printerContext,
        opvp_rop_t rop);
```

Arguments

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

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

Description

This function sets ROP into the Graphics State Object.

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

Return Value

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

4.5.12.opvpGetROP

Name

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

Synopsis

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

Arguments

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

`prop` – Pointer to the buffer to store ROP.

Description

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

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

Return Value

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

4.5.13.opvpSetFillMode

Name

opvpSetFillMode – Set filling mode into the Graphics State Object.

Synopsis

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

Arguments

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

`fillmode` – Filling mode enum. `OPVP_FILLMODE_EVENODD`(Even-odd rule), `OPVP_FILLMODE_WINDING`(Nonzero winding number rule) can be set.

Description

This function sets filling mode into the Graphics State Object.

Return Value

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

4.5.14.opvpGetFillMode

Name

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

Synopsis

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

Arguments

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

`pfillmode` – Pointer to the buffer to store the fill mode enum.

Description

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

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

Return Value

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

4.5.15.opvpSetAlphaConstant

Name

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

Synopsis

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

Arguments

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

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

Description

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

Return Value

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

4.5.16.opvpGetAlphaConstant

Name

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

Synopsis

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

Arguments

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

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

Description

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

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

Return Value

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

4.5.17.opvpSetLineWidth

Name

opvpSetLineWidth – Set the line width.

Synopsis

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

Arguments

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

`width` – Line width value.

Description

This function sets line width for stroke operations to the Graphics State Object. The line width MUST be set in the device coordinate system unit.

The treatment of line width less than one depends on the device or driver implementation, and the maximum line width also depends on the device or driver implementation.

Return Value

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

4.5.18.opvpGetLineWidth

Name

opvpGetLineWidth – Get the line width.

Synopsis

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

Arguments

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

`pwidth` – Pointer to the buffer to store the line width value.

Description

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

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

Return Value

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

4.5.19.opvpSetLineDash

Name

opvpSetLineDash – Set the line dash pattern.

Synopsis

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

Arguments

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

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

num – Number of the line dash pattern array elements.

Description

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

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

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

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

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

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

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

Return Value

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

4.5.20.opvpGetLineDash

Name

opvpGetLineDash – Get the line dash pattern.

Synopsis

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

Arguments

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

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

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

Description

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

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

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

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

Return Value

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

4.5.21.opvpSetLineDashOffset

Name

opvpSetLineDashOffset – Set the line dash pattern offset.

Synopsis

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

Arguments

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

`offset` – Offset value for applying the line dash pattern.

Description

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

Return Value

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

4.5.22.opvpGetLineDashOffset

Name

opvpGetLineDashOffset – Get the line dash pattern offset.

Synopsis

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

Arguments

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

`poffset` – Pointer to the buffer to store the offset value of the line dash pattern.

Description

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

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

Return Value

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

4.5.23.opvpSetLineStyle

Name

opvpSetLineStyle – Set line style.

Synopsis

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

Arguments

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

`linestyle` – Line style enum. `OPVP_LINESTYLE_SOLID`(Solid line) or `OPVP_LINESTYLE_DASH`(Dashed line) can be set.

Description

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

Return Value

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

4.5.24.opvpGetLineStyle

Name

opvpGetLineStyle – Get the line style.

Synopsis

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

Arguments

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

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

Description

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

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

Return Value

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

4.5.25.opvpSetLineCap

Name

opvpSetLineCap – Set line cap style.

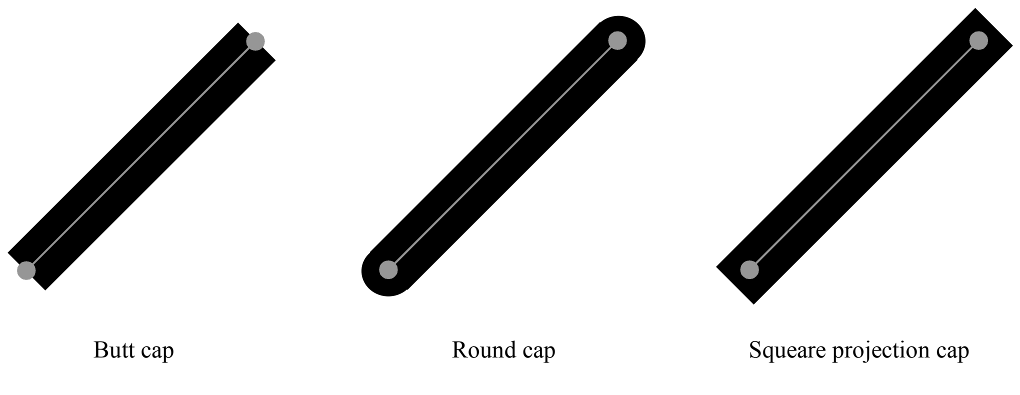
Synopsis

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

Arguments

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

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



Description

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

Return Value

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

4.5.26.opvpGetLineCap

Name

opvpGetLineCap – Get the line cap style.

Synopsis

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

Arguments

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

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

Description

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

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

Return Value

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

4.5.27.opvpSetLineJoin

Name

opvpSetLineJoin – Set line join style.

Synopsis

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

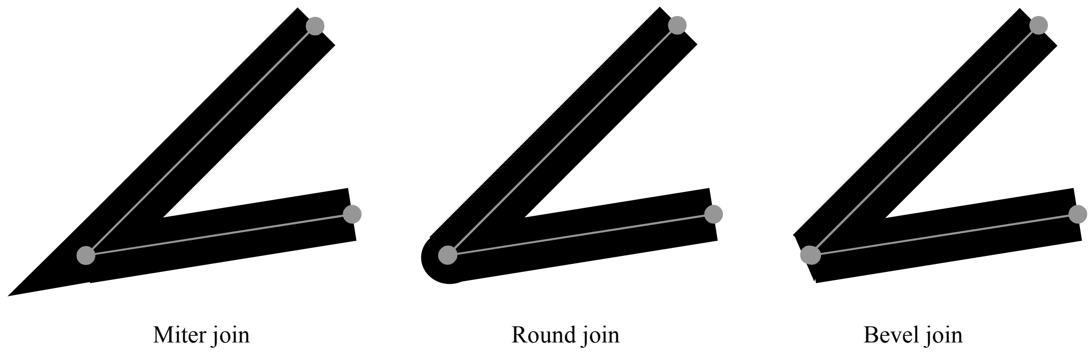
Arguments

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

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

Description

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



Return Value

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

4.5.28.opvpGetLineJoin

Name

opvpGetLineJoin – Get the line join style.

Synopsis

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

Arguments

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

`plinejoin` – Pointer to the buffer to store the line join style enum.

Description

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

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

Return Value

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

4.5.29.opvpSetMiterLimit

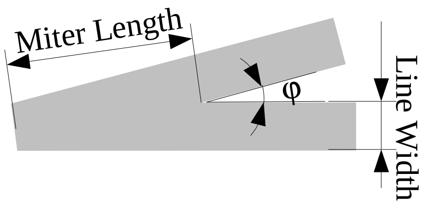
Name

opvpSetMiterLimit – Set miter limit value.

Synopsis

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

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



Arguments

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

`miterlimit` – Maximum miter length.

Description

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

Return Value

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

4.5.30.opvpGetMiterLimit

Name

opvpGetMiterLimit – Get the miter limit value.

Synopsis

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

Arguments

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

`pmiterlimit` – Pointer to the buffer to store the maximum miter length.

Description

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

Initial miter limit value in the Graphics State Object returned by the `opvpOpenPrinter()` function depends on each driver.

Return Value

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

4.5.31.opvpSetPaintMode

Name

opvpSetPaintMode – Set the background color painting mode.

Synopsis

```
opvp_result_t opvpSetPaintMode(
        opvp_dc_t printerContext,
        opvp_paintmode_t paintmode);
```

Arguments

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

`paintmode` – Painting mode enum. `OPVP_PAINTMODE_OPAQUE`(Opaque mode) or `OPVP_PAINTMODE_TRANSPARENT` (Transparent mode) can be set.

Description

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

Return Value

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

4.5.32.opvpGetPaintMode

Name

opvpGetPaintMode – Get the background color painting mode.

Synopsis

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

Arguments

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

`ppaintmode` – Pointer to the buffer to store the painting mode enum.

Description

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

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

Return Value

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

4.5.33.opvpSetStrokeColor

Name

opvpSetStrokeColor – Define the stroke color and pattern for opvpStrokePath() and opvpStrokeFillPath() operations.

Synopsis

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

Arguments

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

brush – Pointer to the opvp_brush_t structure data.

Description

This function registers the color or pattern as a brush data to the Graphics State Object for drawing stroke by the opvpStrokePath() and opvpStrokeFillPath() operations.

If the member pbrush in opvp_brush_t structure *brush is NULL, driver MUST treat it as a solid brush. In this case, color value of the solid brush is specified in color[] in the opvp_brush_t structure *brush, and the color space for the solid brush is specified by colorSpace.

If the member pbrush in opvp_brush_t structure *brush points to a opvp_brushdata_t structure data, driver MUST treat it as a pattern brush. In the opvp_brushdata_t structure data, width, hight are specified in pixel value, and the brush pattern horizontal repetition pitch pitch is specified in the number of bytes. The actual pattern data is specified by data array. The color space for the pattern is specified by colorSpace in the opvp_brush_t structure, and color[] MUST be ignored.

Structures

```
typedef struct _opvp_brushdata {
        opvp_bdtype_t type;
        opvp_int_t width, height, pitch;
        opvp_byte_t data[];       /* must be defined as data[1] for GCC 2.x     */

} opvp_brushdata_t;

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

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

Return Value

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

4.5.34.opvpSetFillColor

Name

opvpSetFillColor – Define the fill color and pattern for opvpFillPath() and opvpStrokeFillPath() operations.

Synopsis

```
opvp_result_t opvpSetFillColor(
        opvp_dc_t printerContext,
        opvp_brush_t *brush);
```

Arguments

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

`brush` – Pointer to brush structure data.

Description

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

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

See the `opvpSetStrokeColor()` function description.

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

Return Value

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

4.5.35.opvpSetBgColor

Name

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

Synopsis

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

Arguments

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

`brush` – Pointer to brush structure data.

Description

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

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

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

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

Return Value

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

4.6.Path Operations

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

All path operation functions are OPTIONAL.

4.6.1.opvpNewPath

Name

opvpNewPath – Start a new path

Synopsis

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

Arguments

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

Description

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

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

Return Value

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

4.6.2.opvpEndPath

Name

opvpEndPath – Declare the end of the current path

Synopsis

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

Arguments

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

Description

Declare the end of the current path

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

Return Value

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

4.6.3.opvpStrokePath

Name

opvpStrokePath – Draw lines along the current path

Synopsis

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

Arguments

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

Description

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

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

Return Value

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

4.6.4.opvpFillPath

Name

opvpFillPath – Fill inside the current path

Synopsis

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

Arguments

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

Description

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

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

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

Return Value

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

4.6.5.opvpStrokeFillPath

Name

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

Synopsis

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

Arguments

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

Description

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

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

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

Return Value

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

4.6.6.opvpSetClipPath

Name

opvpSetClipPath – Set the current path as a clipping region

Synopsis

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

Arguments

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

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

Description

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

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

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

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

Return Value

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

4.6.7.opvpResetClipPath

Name

opvpResetClipPath – Reset the clipping region

Synopsis

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

Arguments

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

Description

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

Return Value

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

4.6.8.opvpSetCurrentPoint

Name

opvpSetCurrentPoint – Set the current point

Synopsis

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

Arguments

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

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

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

Description

Set the current point to (x, y).

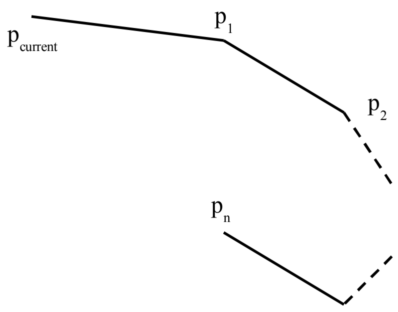
Return Value

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

4.6.9.opvpLinePath

Name

opvpLinePath – Add multiple connected lines to the current path



Synopsis

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

Arguments

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

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

`npoints` – Number of the `opvp_point_t` structure array elements.

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

Description

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

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

Structures

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

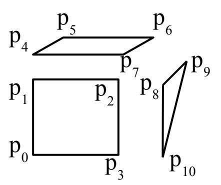
Return Value

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

4.6.10.opvpPolygonPath

Name

opvpPolygonPath – Add polygons to the current path



In the above case, each argument is as following:
npolygons=3
*nvertexes={4, 4, 3}
*points={p0, p1, … p10}

Synopsis

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

Arguments

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

npolygons – Number of polygons to add.

nvertexes – Pointer to the array of the number of points of each polygon.

points – Pointer to the opvp_point_t structure array. The number of points in the array MUST be equal to the total number of points in nvertexes array.

Description

This function adds polygons specified by the points array into the current path..

After driver appends polygons, driver MUST set the last point of the points array as the current point.

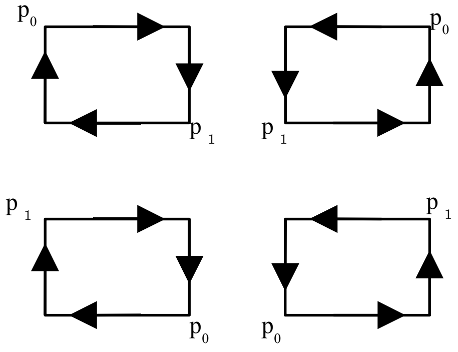
Return Value

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

4.6.11.opvpRectanglePath

Name

opvpRectanglePath – Add rectangles to the current path.



Synopsis

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

Arguments

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

nrectangles – Number of rectangles to add.

rectangles – Pointer to the opvp_rectangle structure array.

Description

This function adds rectangles into the current path..

After driver appends rectangles, driver MUST set the starting point of the last rectangle appended as the current point.

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

Structures

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

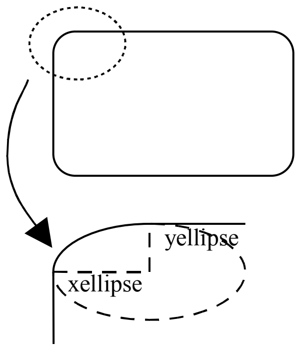
Return Value

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

4.6.12.opvpRoundRectanglePath

Name

opvpRoundRectanglePath – Add round rectangles to the current path.



Synopsis

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

Arguments

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

`nrectangles` – Number of round rectangles to add.

rectangles – Pointer to the `opvp_roundrectangle_t` structure array.

Description

This function adds round rectangles into the current path..

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

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

Structures

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

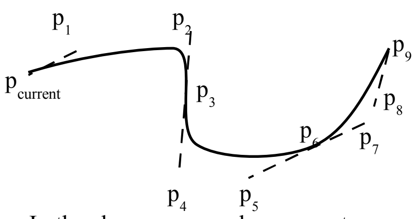
Return Value

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

4.6.13.opvpBezierPath

Name

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



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

Synopsis

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

Arguments

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

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

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

Description

This function adds multiple bezier paths specified by the current point and end points and the control points given by the `points` array. The first bezier path is started by the current point and the third point in the `points` array is the end points. The first point and second point in the `points` array is the control points of the first bezier path. Following bezier paths are started by the previous path's end point and the following two points are the control points and the next point is the end points.

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

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

Return Value

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

4.6.14.opvpArcPath

Name

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

Synopsis

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

Arguments

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

`kind` – Arc kind flag. `OPVP_ARC`(Arc), `OPVP_CHORD`(Chord), `OPVP_PIE`(Pie) can be set.

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

bbx0, bby0, bbx1, bby1 – Circumscribe rectangle.

x0, y0 – Starting point.

x1, y1 – End point.

Description

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 rectangle. The direction of the path is specified by `dir`. When `OPVP_ARC` is set into `kind` and the same point is set into both start and end points, driver MUST append ellipse into the path. If the circumscribe rectangle is a square, driver MUST adds circle to the path.

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

Return Value

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

4.7.Bitmap Image Operations

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

Typical bitmap drawing sequence is as following.

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

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

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

All bitmap operation functions are OPTIONAL.

4.7.1.opvpDrawImage

Name

opvpDrawImage – Draw a bitmap image

Synopsis

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

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

Arguments

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

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

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

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

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

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

`destinationWidth` – Destination drawing width (pixels).

`destinationHeight` – Destination drawing height (pixels).

`imageData` – Pointer to actual source image data.

Description

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

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

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

Color space enum which is set to the `opvpSetColorSpace()` function MUST be one of the color space enum that the driver supports.★これでよいか確認

Only `OPVP_IFORMAT_RAW` MUST be supported by driver, and caller MUST set `OPVP_IFORMAT_RAW` in `imageFormat`. Other image format enum (`OPVP_IFORMAT_JPEG`, `OPVP_IFORMAT_PNG`, `OPVP_IFORMAT_RLE`, etc) are reserved for future use.

After drawing image, driver MUST not change the current point.

Return Value

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

4.7.2.opvpStartDrawImage

Name

opvpStartDrawImage – Start to draw a bitmap image

Synopsis

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

       opvp_int_t destinationWidth,
       opvp_int_t destinationHeight,
       );
```

Arguments

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

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

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

`sourcePitch` – Repetition pitch (bytes) of source image.

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

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

`destinationWidth` – Destination drawing width (pixels).

`destinationHeight` – Destination drawing height (pixels).

Description

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

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

After caller calls this function, caller MUST call `opvpTransferDrawImage()` function once or more times to transfer the actual image data to driver. If caller calls any functions except the `opvpTransferDrawImage()` function between the `opvpStartDrawImage()` and `opvpEndDrawImage()` function, driver MUST return error and set the error code `OPVP_BADREQUEST` in `opvpErrorNo`.

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

Only `OPVP_IFORMAT_RAW` MUST be supported by driver, and caller MUST set `OPVP_IFORMAT_RAW` in `imageFormat`.

Other image format enum (`OPVP_IFORMAT_JPEG`, `OPVP_IFORMAT_PNG`, `OPVP_IFORMAT_RLE`, etc) are reserved for future use.

After drawing image, driver MUST not change the current point.

Return Value

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

4.7.3.opvpTransferDrawImage

Name

opvpTransferDrawImage – Transfer the actual bitmap image data

Synopsis

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

Arguments

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

`count` – Number of image bytes to transfer.

`imageData` – Pointer to actual source image data.

Description

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

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

Return Value

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

4.7.4.opvpEndDrawImage

Name

opvpEndDrawImage – Declare the end of transferring the image data

Synopsis

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

Arguments

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

Description

This function declares the end of transferring the image data.

Return Value

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

4.8. Scan Line Operations

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

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

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

4.8.1. opvpStartScanline

Name

`opvpStartScanline` – Start a scan line drawing

Synopsis

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

Arguments

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

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

Description

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

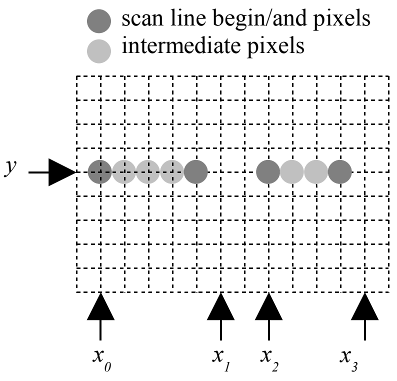
Return Value

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

4.8.2. opvpScanline

Name

`opvpScanline` – Draw the scan lines



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

Synopsis

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

Arguments

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

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

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

Description

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

Return Value

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

4.8.3.opvpEndScanline

Name

`opvpEndScanline` – Terminte the scan line drawing

Synopsis

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

Arguments

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

Description

This function terminates the scan line drawing.

Return Value

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

4.9.Raster Image Operations

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

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

4.9.1.opvpStartRaster

Name

`opvpStartRaster` – Declare to start a raster image drawing

Synopsis

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

Arguments

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

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

Description

This function declares to start a raster data drawing.

Driver MUST draw the raster image from the current point.

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

★三原さんの図 MSB Byte オーダ

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

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

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

Return Value

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

4.9.2.opvpTransferRasterData

Name

`opvpTransferRasterData` – Transfer raster image data

Synopsis

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

Arguments

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

`count` – Number of pixel data elements.

`data` – Pointer to pixel data array.

Description

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

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

Return Value

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

4.9.3.opvpSkipRaster

Name

`opvpSkipRaster` – Skips lines during raster image drawing

Synopsis

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

Arguments

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

`count` – Number of lines to be skipped

Description

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

Return Value

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

4.9.4.opvpEndRaster

Name

`opvpEndRaster` – Terminates the raster image drawing

Synopsis

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

Arguments

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

Description

This function terminates the raster image drawing.

Return Value

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

4.10.Stream Data Operations

Stream data operations provide the support that an application directly creates a PDL and sends the data to a device. 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.

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

4.10.1.opvpStartStream

Name

opvpStartStream – Start streaming data transfer

Synopsis

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

Arguments

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

Description

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

Return Value

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

4.10.2.opvpTransferStreamData

Name

opvpTransferStreamData – send stream data

Synopsis

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

Arguments

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

`count` – Number of data bytes to transfer

`data` – Pointer to the stream data

Description

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

Return Value

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

4.10.3.opvpEndStream

Name

opvpEndStream – Declare the end of streaming data transfer

Synopsis

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

Arguments

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

Description

This function terminates the streaming data transfer.

Return Value

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

6.Macros, Types, Enums and Structures

6.1.Return Values

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

6.2.Error Codes

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

6.3.Basic Types

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

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

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

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

6.4.Image Formats

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

6.5.Color Presentation

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

typedef enum _opvp_cspace {
      OPVP_CSPACE_BW            = 0,
      OPVP_CSPACE_DEVICEGRAY    = 1,
      OPVP_CSPACE_DEVICECMY     = 2,
      OPVP_CSPACE_DEVICECMYK    = 3,
      OPVP_CSPACE_DEVICERGB      = 4,
      OPVP_CSPACE_DEVICEKRGB     = 5,
      OPVP_CSPACE_STANDARDRGB    = 6,
      OPVP_CSPACE_STANDARDRGB64  = 7
} opvp_cspace_t;
```

6.6.Fill, Paint, Clip

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

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

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

6.7.Line

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

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

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

6.8.Brush

```
typedef struct _opvp_brushdata {
        opvp_bdtype_t type;
        opvp_int_t width, height, pitch;
        opvp_byte_t data[];       /* must be defined as data[1] for GCC 2.x     */

} opvp_brushdata_t;

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

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

6.9.Misc. Flags

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

typedef enum _opvp_arcdir {
       OPVP_CLOCKWISE              = 0,
       OPVP_COUNTERCLOCKWISE       = 1
} opvp_arcdir_t;

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

6.10.CTM

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

6.11.Device Information and Capabilities

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

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

6.12.Sample Header File

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

```
#ifndef _OPVP_H_
#define _OPVP_H_

/* Return Values and Error Codes */
#define OPVP_OK            0      /* -1 for errors */
#define OPVP_FATALERROR    -1     /* error: cannot be recovered */
#define OPVP_BADREQUEST    -2     /* error: called where it should not be called */
#define OPVP_BADCONTEXT    -3     /* error: invalid printer context */
#define OPVP_NOTSUPPORTED  -4     /* error: combination of parameters are set */
                                  /*   which cannot be handled by driver or printer */
#define OPVP_JOBCANCELED   -5     /* error: job has been canceled by some cause */
#define OPVP_PARAMERROR    -6     /* error: invalid parameter */

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

/* for opvp_fix_t */
#define OPVP_FIX_FRACT_WIDTH    8
#define OPVP_FIX_FRACT_DENOM    (1<<OPVP_FIX_FRACT_WIDTH)
#define OPVP_FIX_FLOOR_WIDTH    (sizeof(int)*8-OPVP_FIX_FRACT_WIDTH)
```

```
/* convert macro */
#define      OPVP_I2FIX(i,fix)   (fix=i<<OPVP_FIX_FRACT_WIDTH)
#define      OPVP_F2FIX(f,fix)   (fix=((int)floor(f)<<OPVP_FIX_FRACT_WIDTH)\
                              |((int)((f-floor(f))*OPVP_FIX_FRACT_DENOM)\
                                &(OPVP_FIX_FRACT_DENOM-1)))

/* graphic elements */
typedef struct _opvp_point {
      opvp_fix_t x, y;
} opvp_point_t;

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

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

/* Image Formats */
typedef enum _opvp_imageformat {
      OPVP_IFORMAT_RAW          = 0,
      OPVP_IFORMAT_RLE          = 1,
      OPVP_IFORMAT_JPEG         = 2,
      OPVP_IFORMAT_PNG          = 3
} opvp_imageformat_t;

/* Color Presentation */
typedef enum _opvp_colormapping {
      OPVP_CMAP_DIRECT          = 0,
      OPVP_CMAP_INDEXED         = 1
} opvp_colormapping_t;

typedef enum _opvp_cspace {
      OPVP_CSPACE_BW            = 0,
      OPVP_CSPACE_DEVICEGRAY    = 1,
      OPVP_CSPACE_DEVICECMY     = 2,
      OPVP_CSPACE_DEVICECMYK    = 3,
      OPVP_CSPACE_DEVICERGB     = 4,
      OPVP_CSPACE_DEVICEKRGB    = 5,
      OPVP_CSPACE_STANDARDRGB   = 6,
      OPVP_CSPACE_STANDARDRGB64 = 7
} opvp_cspace_t;

/* Fill, Paint, Clip */
typedef enum _opvp_fillmode {
      OPVP_FILLMODE_EVENODD     = 0,
      OPVP_FILLMODE_WINDING     = 1
} opvp_fillmode_t;

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

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

/* Line */
typedef enum _opvp_linestyle {
      OPVP_LINESTYLE_SOLID      = 0,
      OPVP_LINESTYLE_DASH       = 1
} opvp_linestyle_t;

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

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

/* Brush */
typedef enum _opvp_bdtype {
        OPVP_BDTYPE_NORMAL         = 0
} opvp_bdtype_t;

typedef struct _opvp_brushdata {
        opvp_bdtype_t type;
        opvp_int_t width, height, pitch;
#if defined(__GNUC__) && __GNUC__ <= 2
        opvp_byte_t data[1];
#else
        opvp_byte_t data[];
#endif

} opvp_brushdata_t;

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

/* Misc. Flags */
typedef enum _opvp_arcmode {
        OPVP_ARC                   = 0,
        OPVP_CHORD                 = 1,
        OPVP_PIE                   = 2
} opvp_arcmode_t;

typedef enum _opvp_arcdir {
        OPVP_CLOCKWISE             = 0,
        OPVP_COUNTERCLOCKWISE      = 1
} opvp_arcdir_t;

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

/* CTM */
typedef struct _opvp_ctm {
        opvp_float_t a, b, c, d, e, f;
} opvp_ctm_t;

/* Device Information and Capabilites */
typedef enum _opvp_queryinfoflags {
  OPVP_QF_DEVICERESOLUTION = 0x00000001,
  OPVP_QF_MEDIASIZE        = 0x00000002,
  OPVP_QF_PAGEROTATION     = 0x00000004,
  OPVP_QF_MEDIANUP         = 0x00000008,
  OPVP_QF_MEDIADUPLEX      = 0x00000010,
  OPVP_QF_MEDIASOURCE      = 0x00000020,
  OPVP_QF_MEDIADESTINATION = 0x00000040,
  OPVP_QF_MEDIATYPE        = 0x00000080,

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

```

```
/* API Procedure Entries */
typedef      struct _opvp_api_procs {
       opvp_dc_t      (*opvpOpenPrinter)(opvp_int_t,opvp_char_t*,opvp_int_t*,struct
_opvp_api_procs**);※引数削除
       opvp_result_t (*opvpClosePrinter)(opvp_dc_t);
       opvp_result_t (*opvpStartJob)(opvp_dc_t,opvp_char_t*);
       opvp_result_t (*opvpEndJob)(opvp_dc_t);
       opvp_result_t (*opvpAbortJob)(opvp_dc_t);
       opvp_result_t (*opvpStartDoc)(opvp_dc_t,opvp_char_t*);
       opvp_result_t (*opvpEndDoc)(opvp_dc_t);
       opvp_result_t (*opvpStartPage)(opvp_dc_t,opvp_char_t*);
       opvp_result_t (*opvpEndPage)(opvp_dc_t);
       opvp_result_t
(*opvpQueryDeviceCapability)(opvp_dc_t,opvp_flag_t,opvp_int_t,opvp_byte_t*);
       opvp_result_t
(*opvpQueryDeviceInfo)(opvp_dc_t,opvp_flag_t,opvp_int_t,opvp_char_t*);
       opvp_result_t (*opvpResetCTM)(opvp_dc_t);
       opvp_result_t (*opvpSetCTM)(opvp_dc_t,opvp_ctm_t*);
       opvp_result_t (*opvpGetCTM)(opvp_dc_t,opvp_ctm_t*);
       opvp_result_t (*opvpInitGS)(opvp_dc_t);
       opvp_result_t (*opvpSaveGS)(opvp_dc_t);
       opvp_result_t (*opvpRestoreGS)(opvp_dc_t);

       opvp_result_t (*opvpQueryColorSpace)(opvp_dc_t,opvp_int_t*, opvp_cspace_t*);※
引数順番変更
       opvp_result_t (*opvpSetColorSpace)(opvp_dc_t,opvp_cspace_t);
       opvp_result_t (*opvpGetColorSpace)(opvp_dc_t,opvp_cspace_t*);
       opvp_result_t (*opvpQueryROP)(opvp_dc_t,opvp_int_t*,opvp_rop_t*);
       opvp_result_t (*opvpSetROP)(opvp_dc_t,opvp_rop_t);
       opvp_result_t (*opvpGetROP)(opvp_dc_t,opvp_rop_t*);
       opvp_result_t (*opvpSetFillMode)(opvp_dc_t,opvp_fillmode_t);
       opvp_result_t (*opvpGetFillMode)(opvp_dc_t,opvp_fillmode_t*);
       opvp_result_t (*opvpSetAlphaConstant)(opvp_dc_t,opvp_float_t);
       opvp_result_t (*opvpGetAlphaConstant)(opvp_dc_t,opvp_float_t*);
       opvp_result_t (*opvpSetLineWidth)(opvp_dc_t,opvp_fix_t);
       opvp_result_t (*opvpGetLineWidth)(opvp_dc_t,opvp_fix_t*);
       opvp_result_t (*opvpSetLineDash)(opvp_dc_t,opvp_fix_t*,opvp_int_t);
       opvp_result_t (*opvpGetLineDash)(opvp_dc_t,opvp_fix_t*,opvp_int_t*);
       opvp_result_t (*opvpSetLineDashOffset)(opvp_dc_t,opvp_fix_t);
       opvp_result_t (*opvpGetLineDashOffset)(opvp_dc_t,opvp_fix_t*);
       opvp_result_t (*opvpSetLineStyle)(opvp_dc_t,opvp_linestyle_t);
       opvp_result_t (*opvpGetLineStyle)(opvp_dc_t,opvp_linestyle_t*);
       opvp_result_t (*opvpSetLineCap)(opvp_dc_t,opvp_linecap_t);
       opvp_result_t (*opvpGetLineCap)(opvp_dc_t,opvp_linecap_t*);
       opvp_result_t (*opvpSetLineJoin)(opvp_dc_t,opvp_linejoin_t);
       opvp_result_t (*opvpGetLineJoin)(opvp_dc_t,opvp_linejoin_t*);
       opvp_result_t (*opvpSetMiterLimit)(opvp_dc_t,opvp_fix_t);
       opvp_result_t (*opvpGetMiterLimit)(opvp_dc_t,opvp_fix_t*);
       opvp_result_t (*opvpSetPaintMode)(opvp_dc_t,opvp_paintmode_t);
       opvp_result_t (*opvpGetPaintMode)(opvp_dc_t,opvp_paintmode_t*);
       opvp_result_t (*opvpSetStrokeColor)(opvp_dc_t,opvp_brush_t*);
       opvp_result_t (*opvpSetFillColor)(opvp_dc_t,opvp_brush_t*);
       opvp_result_t (*opvpSetBgColor)(opvp_dc_t,opvp_brush_t*);
       opvp_result_t (*opvpNewPath)(opvp_dc_t);
       opvp_result_t (*opvpEndPath)(opvp_dc_t);
       opvp_result_t (*opvpStrokePath)(opvp_dc_t);
       opvp_result_t (*opvpFillPath)(opvp_dc_t);
       opvp_result_t (*opvpStrokeFillPath)(opvp_dc_t);
       opvp_result_t (*opvpSetClipPath)(opvp_dc_t,opvp_cliprule_t);
       opvp_result_t (*opvpResetClipPath)(opvp_dc_t);
       opvp_result_t (*opvpSetCurrentPoint)(opvp_dc_t,opvp_fix_t,opvp_fix_t);
       opvp_result_t
(*opvpLinePath)(opvp_dc_t,opvp_pathmode_t,opvp_int_t,opvp_point_t*);
       opvp_result_t
(*opvpPolygonPath)(opvp_dc_t,opvp_int_t,opvp_int_t*,opvp_point_t*);
       opvp_result_t (*opvpRectanglePath)(opvp_dc_t,opvp_int_t,opvp_rectangle_t*);
       opvp_result_t
(*opvpRoundRectanglePath)(opvp_dc_t,opvp_int_t,opvp_roundrectangle_t*);
       opvp_result_t (*opvpBezierPath)(opvp_dc_t,opvp_int_t,opvp_point_t*);
       opvp_result_t
(*opvpArcPath)(opvp_dc_t,opvp_arcmode_t,opvp_arcdir_t,opvp_fix_t,opvp_fix_t,opvp_fix_
t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t,opvp_fix_t);
        opvp_result_t
(*opvpDrawImage)(opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imageform
at_t,opvp_rectangle_t,void*);
        opvp_result_t
(*opvpStartDrawImage)(opvp_dc_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_int_t,opvp_imag
eformat_t,opvp_rectangle_t);
        opvp_result_t (*opvpTransferDrawImage)(opvp_dc_t,opvp_int_t,void*);
        opvp_result_t (*opvpEndDrawImage)(opvp_dc_t);
        opvp_result_t (*opvpStartScanline)(opvp_dc_t,opvp_int_t);
        opvp_result_t (*opvpScanline)(opvp_dc_t,opvp_int_t,opvp_int_t*);
        opvp_result_t (*opvpEndScanline)(opvp_dc_t);
        opvp_result_t (*opvpStartRaster)(opvp_dc_t,opvp_int_t);
        opvp_result_t (*opvpTransferRasterData)(opvp_dc_t,opvp_int_t,opvp_byte_t*);
        opvp_result_t (*opvpSkipRaster)(opvp_dc_t,opvp_int_t);
        opvp_result_t (*opvpEndRaster)(opvp_dc_t);
        opvp_result_t (*opvpStartStream)(opvp_dc_t);
        opvp_result_t (*opvpTransferStreamData)(opvp_dc_t,opvp_int_t,void*);
        opvp_result_t (*opvpEndStream)(opvp_dc_t);
} opvp_api_procs_t;

/* Function prototype */
opvp_dc_t opvpOpenPrinter(
        opvp_int_t outputFD,
        opvp_char_t *printerModel,
        opvp_int_t apiVersion[2],
        opvp_int_t *nApiEntry,
        opvp_api_procs_t **apiProcs);

/* error no */
extern opvp_int_t   opvpErrorNo;

#endif /* _OPVP_H_ */
```

7.Authors and Contributers

7.1.Editors

Osamu Mihara – Fuji Xerox Co., Ltd.

7.2.Authors

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

7.3.Contributers

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

8.History

Version 0.1 (Japanese) Mihara/Toratani

Version 0.2 (Japanese) Mihara

Openprinting-0.1.1 by opfc have implemented based on this version

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

Translation to English
Some feedback from opfc implementation

Version 1.0 RC1 2005-7-20 Mihara

Change copyright notice from FDL to MIT style copyright

Delete temporary font operations.

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

Add API version number to OpenPrinter() parameter

Add pitch and delete count to/from DrawImage()/StartDrawImage()

Add a figure for SetMiterLimit.

Change function names/constands/enum/structures names to add FSG prefixes.

Change copyright year and dates

Delete "Printer Driver Database"

Delete *cmap* from CSPASE chart.

Fix parameter colorSpace (wrong) to colorDepth (correct) for DrawImage()/StartDrawImage

Change description for fsgpdStartJob/fsgpdEndJob/fsgpdAbortJob

Add type FSGPD_CHAR for character type

Version 1.0 RC2 2006-10-20

Add FSGPD_CSPACE_DEVICEKRGB

Remove "current implementation" descriptions from Color Scheme and Color Space descriptions.

Remove "described later" descriptions

Make support of updf for scheme "MUST."

Version 1.0 RC2 2006-12-05

Format: Add line numbers and chapter numbers.

Format: Replace Microsoft Expressions object to OOo Expressions

Version 1.0 RC3 2007-5-19 Toratani

Append “opvp” or “OPVP” to function names, enums, symbols according to the naming rules.

Update parameters of `opvpOpenPrinter()` function and the bitmap image functions.

Rearrange the document sections, chapters and text formant.

Merge the RC2 2006-6-9 – 2006-12-05 updates made by Mihara