OpenPrinting

Vector Printer Driver Application Program Interface (OPVP) Specification

Version-1.0 RC3

(2007-5-24)

OpenPrinting Vector Printer Driver Application Program Interface Specification

Table of Contents

1

1.Notation and Terminology

1.1.Notational Conventions

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

Font~~d~~ and Style	Description	Examples

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

	Function parameters .	`printerContext`

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

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

1.2.Conformance 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 an Application Program Interface (API) which is used in the printing environment offered by OpenPrinting. In this document, the word “printer driver” or “driver” refers to a software library. The d~~D~~river translates document data generated by application programs into a printer command data stream which is consumed by a printer.

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

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

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

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

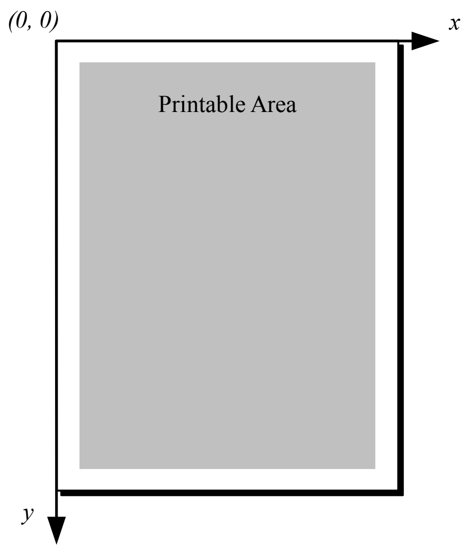
Printer driver name and its model name are managed by driver data base. The driver data base should be supplied in UPDF and/or PPD format.

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

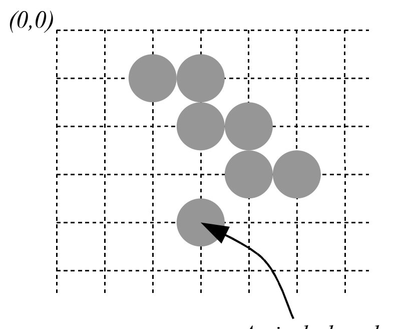


The origin of the coordinate system as~~which is~~ defined and used in this ~~document~~specification is the physical upper left corner of the media handled by a~~each~~ device. ~~The u~~ Units of the coordinate system ~~is~~are based on ~~each~~the device resolution. Positive direction on the x-axis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

~~T~~The type of the coordinate value is a si~~ng~~gned fixed point 32 bit~~s~~ value, where the integer part uses~~takes~~ 24 bits and the fractional part~~decimal takes~~ 8 bits. The ~~(~~type name is `opvp_fix_t`~~)~~.

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

3.2.Objects



~~F~~The following kind~~s~~ of graphics objects are handled in this document~~.~~:

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

3.3.Graphics State Object

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

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

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

Drawing Attributes	Related functions
Color~~S~~ space	`opvpQueryColorSpace(), opvpSetColorSpace(), opvpGetColorSpace`
Raster operation code	`opvpQueryROP(), opvpSetROP(), opvpGetROP()`★削除？
Filling mode	`opvpSetFillMode(), opvpGetFillMode()`
Alpha blending constant	`opvpSetAlphaConstant(), opvpGetAlphaConstant()`
Stroke line width	`opvpSetLineWidth(), opvpGetLineWidth()`
Line dash ~~style~~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~~ing~~ color	`opvpSetFillColor()`
Background color	`opvpSetBgColor()`

3.4.CTM

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

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

To ~~set or get~~get, set or reset a CTM from the Graphics State Object, ~~refer the description of~~use the `opvp~~Reset~~GetCTM()`, `opvpSetCTM()`, `opvp~~Get~~ResetCTM()` functions.

3.5.Color

The f~~F~~ollowing color spaces are defined ~~in~~by this document.

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

Only `OPVP_CSPACE_BW`, `OPVP_CSPACE_DEVICEGRAY` and `OPVP_CSPACE_STANDARDRGB` ~~should~~need to be supported by printer driver ~~which i~~implementations 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 Printer Contexts

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

4.1.1.opvpOpenPrinter

Name

opvpOpenPrinter – Creates a printer context.

Synopsis

```
opvp_dc_t opvpOpenPrinter(
        opvp_int_t outputFD,
        opvp_char_t *printerModel,
        opvp_int_t apiVersion[2],
~~        opvp_int_t *nApiEntry,★この引数は必要なのか？必要。Caller側がチェックする MUST。~~
        struct _opvp_api_procs **apiEntry);~~★この書き方であっているか？何故２重ポインタ？~~
```

Arguments

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

`printerModel` – Printer ~~M~~model ~~N~~name ~~in~~ (UTF-8 encoded).

`apiVersion` – Vector ~~P~~printer ~~D~~driver ~~S~~specification ~~V~~version ~~number which~~supported by the driver ~~supports~~. ~~A~~`apiVersion[0]` is the major and `apiVersion[1]` is the minor value of the version respectively.~~`nApiEntry` – Number of `apiEntry` array elements.~~

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

Description

This function initializes the driver. ~~C~~ The caller MUST specify the file descriptor for writing the printing data stream, and the driver MUST write~~generates~~ the printing data stream ~~and MUST write~~ it generates ~~in~~to the file descriptor. The caller may specify ~~the~~a UTF-8 encoded printer model name ~~by~~via `printerModel`~~ in UFT-8~~. If the caller ~~sets~~passes `NULL` ~~in~~for `printerModel`, the driver SHOULD use the default printer model of the driver. ~~P~~The printer model name should be defined in the printer model data base.

The d~~D~~river may write ~~its~~ debugging messages ~~in~~to `stderr`. Therefore `stderr` MUST not be ~~given for~~passed as `outputFD` by the caller.

The d~~D~~river MUST allocate the `struct _opvp_api_procs` buffer and store the address of each driver API entry ~~address of the driver~~ into ~~each~~to corresponding member of `apiEntry`. If the driver does not ~~prepare~~support some API entries, ~~the driver~~it MUST store `NULL` into the corresponding `apiEntry` members.~~ドライバ側がエリアを確保して返す。~~

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

The~~P~~ printer driver MUST return a printer context as the return value of this function. The printer context MUST be a unique number which is managed by the driver. The c~~C~~aller ~~set~~passes the printer context as~~to~~ the first ~~parameter~~argument ~~to~~when calling other API entries.

Return Value

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

4.1.2.opvpClosePrinter

Name

opvpClosePrinter – Deletes a printer context.

Synopsis

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

Arguments

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

Description

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

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

Return Value

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

4.2.Job, Document and Page Operations

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

4.2.1.opvpStartJob

Name

opvpStartJob – Declare~~ to~~s the start of a printing job.

Synopsis

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

Arguments

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

`jobInfo` – Printing job property string.

Description

This function declares ~~to~~the start of a printing job.

The c~~C~~aller MUST call this function before calling the `opvpStartDoc()` or -`opvpStartPage()` functions or any other drawing API entries.

The c~~C~~aller can set printing job properties via~~into~~ `jobInfo`. The p~~**P**~~**rinter** driver MUST keep the given printing job properties until the `opvpEndJob()` function is called. When the caller calls the `opvpStartJob()` function again after calling the `opvpEndJob()` function, the driver MUST override the former `jobInfo` with~~by~~ the new `jobInfo` specified~~given~~ by the latest `opvpStartJob()` function call.

If the caller passes~~sets~~ `NULL` for the~~to~~ `jobInfo`, the printer driver SHOULD use its default printing job properties. The d~~D~~ata format of -`jobInfo` is described ~~later~~ in the section "Attribute of Job, Document and Page Operations" in this document.

It depends on printer and driver capabilities whether nest~~ing~~ed printing jobs are supported. A nested printing job is one where the ~~(~~`opvpStartJob()` and `opvpEndJob()` functions ~~can be~~are called between an enclosing pair of the `opvpStartJob()` and `opvpEndJob()` functions~~ are called) can be handled~~. If a printer or printer driver does not ~~allow~~support nest~~ing~~ed printing jobs, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST` ~~in `opvpErrorNo`~~.~~BADREQUEST にセット？errorno.★何を~~

Return Value

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

4.2.2.opvpEndJob

Name

opvpEndJob –Declare~~ to terminate~~s the end of a printing job.

Synopsis

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

Arguments

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

Description

This function declares ~~to terminate~~the end of a~~ the~~ printing job.

The c~~C~~aller MUST call this function after it finishes processinging ~~each~~a printing job~~ processing~~.

Return Value

`OPVP_OK` or -1 ~~when~~in case of error. In the latter case,~~ and~~ the driver MUST store the detailed error code in ~~`errorno`~~`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 – Declares ~~to~~the start of a printing document.

Synopsis

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

Arguments

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

`docInfo` – Printing document property string.

Description

This function declares ~~to~~the start of a printing document.

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

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

If the caller ~~sets~~passes `NULL` ~~to~~for the `docInfo`, the printer driver SHOULD use its default printing document properties. The d~~D~~ata format of -`docInfo` is described ~~later~~ in the section "Attributes ~~o~~for Job, Document and Page Operations" in this document.

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

Return Value

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

4.2.5.opvpEndDoc

Name

opvpEndDoc – Declares the end of a ~~to terminate the~~ printing document

Synopsis

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

Arguments

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

Description

This function declares the end of a~~o terminate the~~ printing document.

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

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

Return Value

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

4.2.6.opvpStartPage

Name

opvpStartPage – Declares ~~to~~the start of a printing page.

Synopsis

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

Arguments

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

pageInfo – Printing page property string.

Description

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

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

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

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

Return Value

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

4.2.7.opvpEndPage

Name

opvpEndPage – Declares ~~to terminate~~ the end of a printing page.

Synopsis

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

Arguments

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

Description

This function declares ~~to terminate~~ the end of a printing page.

The c~~C~~aller MUST call this function after it finishes processing a~~each~~ page~~ processing~~.

Return Value

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

4.3.Query Operations

This section defines f~~F~~unctions to query device capabilities and information. Driver support for t~~T~~hese functions ~~are~~is OPTIONAL.

4.3.1.opvpQueryDeviceCapability

Name

opvpQueryDeviceCapability – Quer~~y~~ies for device capabilities.

Synopsis

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

Arguments

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

`queryflag` – Flag ~~which~~ specif~~ies~~ying which ~~the~~ device capability to query.

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

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

Description

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

~~★どこに長さを返すのか？If caller sets `NULL` to `infoBuf`, printer driver MUST return the number of bytes to store all the capabilities in integer format. In this case, the caller also MUST set the `queryflag` and `buflen`.~~The c~~C~~aller MUST set one or more bit~~flag~~s of~~into~~ `queryflag` to query capabilities. The values in the f~~F~~ollowing enum~~s~~erations should be supported by the driver:~~;~~

```
typedef enum _opvp_queryinfoflags {
  OPVP_QF_DEVICERESOLUTION       = 0x00000001,
  OPVP_QF_MEDIASIZE              = 0x00000002,
  OPVP_QF_PAGEROTATION           = 0x00000004,
  OPVP_QF_MEDIANUP               = 0x00000008,
  OPVP_QF_MEDIADUPLEX            = 0x00000010,
  OPVP_QF_MEDIASOURCE            = 0x00000020,
  OPVP_QF_MEDIADESTINATION       = 0x00000040,
  OPVP_QF_MEDIATYPE              = 0x00000080,
  OPVP_QF_MEDIACOPY              = 0x000~~10000~~00100,/* Maximum copy number supported */
  OPVP_QF_PRINTREGION            = 0x000~~2~~10000 /* only for opvpQueryDeviceInfo use */
} opvp_queryinfoflags_t;
```

~~値を返す Max の MediaCopy★ OPVP_QF_MEDIASIZE も opvpQueryDeviceInfo のみだったか？(ヘッダではそう書いてあるが日本語版は。。。)大谷さんに確認。MediaCopy では？値も違う。~~

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

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

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

Return Value

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

4.3.2.opvpQueryDeviceInfo

Name

opvpQueryDeviceInfo – Quer~~y~~ies for device information.

Synopsis

```
opvp_result_t opvpQueryDeviceInfo(
        opvp_dc_t printerContext,
        opvp_queryinfoflags_t queryflag,※★~~ヘッダでは opvp_flag_tになっているが？~~
        opvp_int_t *buflen,※~~を返す ―――――バッファサイズ~~
        opvp_char_t *infoBuf);※★~~opvpQueryDeviceCapabilityは opvp_byte_t*になっているが？~~
```

Arguments

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

queryflag – Flag ~~which~~ specif~~ying~~ies which~~ the~~ device information to query.

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

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

Description

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

~~追記ポインタに返す。★どこに長さを返すのか？If caller sets NULL to infoBuf, printer driver MUST return the number of bytes to store all the information in integer format In this case, the caller also MUST set the queryflag and buflen.~~

The c~~C~~aller MUST set one or more bit~~ flag~~s ~~in~~of queryflag to query for information. Values of the s~~S~~ame enumerations~~ which are~~ used for the opvpQueryDeviceCapability() function MUST be used.

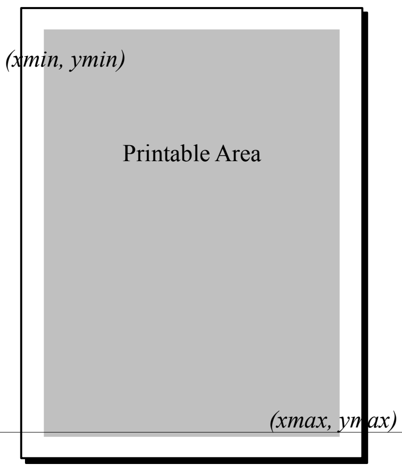
~~エラーを返す★The driver MUST return queried information into the infoBuf buffer in ASCII text format. If the buffer does not have enough length to store all the information queried, the stored information MUST be truncated by driver. In this case, driver MUST return without error.~~

The~~D~~ driver MUST return the queried information in ASCII text format into the infoBuf buffer~~ in ASCII text format~~, and also return the number of bytes of th~~e information~~is text ~~into~~via *buflen. If the buffer ~~does not have enough length~~is too small to store ~~all~~ the ~~information queried~~query result, the driver MUST return the necessary number of bytes to retrieve ~~all the information~~the query result in *buflen, ~~and~~ return an error and set the detailed error code OPVP_PARAMERROR~~ into opvpErrorNo .~~. If the caller ~~sets~~passes NULL ~~in~~as the infoBuf pointer, the driver MUST return the number of ~~the~~ bytes required to store the query result via~~of all the information in~~ *buflen.

The format of the information name~~- and~~ value pairs ~~format of each information~~stored in the infoBuf buffer ~~has~~is the same ~~format~~as that of the jobInfo ~~which is~~used ~~for~~with the opvpStartJob() function.

When the caller sets the OPVP_QF_PRINTREGION ~~into~~bit of the queryflag~~D –~~, the driver MUST ~~respond~~provide the printable area ~~for its query in~~using the current resolution setting. The printable area~~ values~~ format MUST be as given below~~following~~. ~~Each~~ The values ~~represents~~correspond to the *x* and *y* coordinates of the left top and right bottom corners, respectively, of the current printable area setting as shown in the figure. These values depend on the current media orientation setting.

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



Return Value

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

4.4.Attributes ~~of~~for Job, Document and Page Operations

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

~~The a~~Attribute names and values MUST be ~~given in~~specified as ASCII strings in the following format;

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

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

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

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

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

Conforming d~~D~~rivers~~ which conforms to this specification~~ MUST ignore unknown properties to ~~keep the~~maintain compatibility with future vendor or standard extensions.

Major UPDF attributes are shown in the table below:~~;~~

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

Attribute	Name	Value	Effective in		
			Job	Doc	Page
		high normal			

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

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

4.5.Graphics State Object Operations

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

4.5.1.opvpResetCTM

Name

opvpResetCTM – Initializes the CTM of ~~the~~a Graphics State Object.

Synopsis

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

Arguments

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

Description

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

~~.~~the identity matrix:

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

Return Value

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

4.5.2.opvpSetCTM

Name

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

Synopsis

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

Arguments

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

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

Description

This function sets the~~a~~ CTM ~~to the~~of a Graphics State Object.

The p~~P~~rinter (device) coordinate system ~~value~~coordinates [x_{dev}, y_{dev}] ~~is represented by CTM~~ and the caller (renderer) coordinate system ~~value~~coordinates [x_{ren}, y_{ren}] are related via the CTM as shown in the following equation:~~.~~

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

Structures

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

Return Value

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

4.5.3.opvpGetCTM

Name

opvpGetCTM – Gets the CTM ~~from the~~of a Graphics State Object.

Synopsis

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

Arguments

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

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

Description

This function gets the CTM ~~from the~~of a Graphics State Object.

Structures

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

Return Value

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

4.5.4.opvpInitGS

Name

opvpInitGS – Initializes the parameters ~~in the~~of a Graphics State Object.

Synopsis

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

Arguments

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

Description

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

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

Return Value

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

4.5.5.opvpSaveGS

Name

opvpSaveGS – Saves the Graphics State Object parameters.

Synopsis

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

Arguments

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

Description

This function saves the Graphics State Object parameters o~~i~~nto the driver's stack.

~~Driver MUST be able to save at least one set of the Graphics State Object parameters into its stack. If caller calls this function more than once without calling opvpRestoreGS() function, this function may return error.★何のエラーを返す~~

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 ~~errorno~~opvpErrorNo.

Return Value

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

4.5.6.opvpRestoreGS

Name

opvpRestoreGS – Restores ~~the~~ Graphics State Object parameters.

Synopsis

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

Arguments

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

Description

This function retrieves ~~the~~ Graphics State Object parameters from the driver's stack and restores 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 ~~errorno~~opvpErrorNo.

Return Value

OPVP_OK or -1 ~~when~~in case of error. In the latter case,~~ and~~ the driver MUST store the detailed error code in ~~errorno~~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？
~~);opvp_int_t *pnum~~
```

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.

~~`pnum` – Pointer to the buffer which stores the number of the color space enum array elements.~~

Description

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

~~Driver MUST return the number of the color space enum retrieved in `*pnum`. If the number of enums exceeds the number of elements prepared by the caller, driver MUST return the necessary number to retrieve all the enums in `*pnum`.★aller MUST set the number of the `pcspace` elements in `*pnum`.c heTCaller MUST prepare at least 4 elements for the `pcspace` array to store the color space enums.★4でよいか？ Also t~~

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 ~~`errorno`~~`opvpErrorNo`.

4.5.8.opvpSetColorSpace

Name

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

Synopsis

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

Arguments

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

`cspace` – Color space value~~enum~~.

Description

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

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

Return Value

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

4.5.9.opvpGetColorSpace

Name

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

Synopsis

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

Arguments

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

`pcspace` – Pointer to the ~~buffer to store the~~color space ~~enum~~value to be returned.

Description

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

~~する OpvpSetColorSpace()関数を呼ばないで返る値はドライバに依存★デフォルト値は？~~

The i~~I~~nitial color space ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.

Return Value

`OPVP_OK` or -1 ~~when~~in case of error. In the latter case,~~ and~~ the driver MUST store the detailed error code in ~~`errorno`~~`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, ~~D~~driver MUST return ROPs in `prop` array and also return the number of the ROPs retrieved in `*pnum`. ~~If the number of ROPs exceeds the number of elements prepared by the caller, driver MUST return the necessary number to retrieve all the ROPs in `*pnum`.~~

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 ~~`errorno`~~`opvpErrorNo`.~~Driver returns ROPs in the array by its preferable order.★QueryColorSpace と違う？~~

Return Value

`OPVP_OK` or -1 when error , and the driver MUST store the detailed error code in ~~`errorno`~~`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 ~~`errorno`~~`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.
~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.13.opvpSetFillMode

Name

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

Synopsis

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

Arguments

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

`fillmode` – Filling mode enumeration value. `OPVP_FILLMODE_EVENODD` (e~~E~~ven-odd rule)~~,~~ and `OPVP_FILLMODE_WINDING` (~~N~~non-zero winding number rule) can be set.

Description

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

Return Value

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

4.5.14.opvpGetFillMode

Name

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

Synopsis

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

Arguments

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

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

Description

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

The i~~I~~nitial filling mode ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.15.opvpSetAlphaConstant

Name

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

Synopsis

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

Arguments

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

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

Description

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

Return Value

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

4.5.16.opvpGetAlphaConstant

Name

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

Synopsis

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

Arguments

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

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

Description

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

The i~~I~~nitial alpha blending constant ~~in the~~of a Graphics State Object ~~returned by the opvpOpenPrinter() function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.17.opvpSetLineWidth

Name

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

Synopsis

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

Arguments

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

width – Line width value.

Description

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

T~~he t~~reatment of line widths less than one depends on the device or driver implementation. Similarly, ~~and~~ the maximum line width that can be set also depends on the device or driver implementation.

Return Value

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

4.5.18.opvpGetLineWidth

Name

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

Synopsis

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

Arguments

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

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

Description

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

The i~~I~~nitial line width ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.19.opvpSetLineDash

Name

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

Synopsis

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

Arguments

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

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

`num` – Number of ~~the line dash pattern array~~ elements in `pdash`.

Description

This function sets a ~~stroke~~line dash pattern ~~to~~for use by a~~ the~~ Graphics State Object.

When the painting mode is `OPVP_PAINTMODE_OPAQUE` (o~~O~~paque mode), the odd numbered (1st, 3rd, 5th, 7th ...) elements ~~in~~of the `pdash` array ~~is~~indicate the length for dashes in the foreground color~~,~~ and the even numbered (2nd, 4th, 6th, 8th...) elements ~~is~~indicate the~~ a~~ length for background color dashes.

When the painting mode is `OPVP_PAINTMODE_TRANSPARENT` (t~~T~~ransparent mode), the odd numbered (1st, 3rd, 5th, 7th ...) elements ~~in~~of the `pdash` array ~~is~~indicate the length for dashes in the foreground color~~,~~ and the even numbered (2nd, 4th, 6th, 8th...) elements ~~is a~~indicate the length of ~~a~~ line segments ~~which is~~that are not painted.

The lengths set in the `pdash` array MUST be in ~~the~~ device coordinate system units.

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

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

If the caller ~~set~~passes zero ~~in~~for **`num`**, the driver MUST draw solid lines.

Return Value

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

4.5.20.opvpGetLineDash

Name

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

Synopsis

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

Arguments

printerContext – Printer context value returned by the 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 in pdash.

Description

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

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

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

The i~~I~~nitial line dash pattern ~~in the~~of a Graphics State Object ~~returned by the opvpOpenPrinter() function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.21.opvpSetLineDashOffset

Name

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

Synopsis

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

Arguments

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

offset – Offset value ~~for applying~~of the line dash pattern.

Description

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

Return Value

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

4.5.22.opvpGetLineDashOffset

Name

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

Synopsis

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

Arguments

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

poffset – Pointer to ~~the buffer to store~~ the offset value of the line dash pattern to be returned.

Description

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

The i~~I~~nitial line dash pattern offset ~~in the~~of a Graphics State Object ~~returned by the opvpOpenPrinter() function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.23.opvpSetLineStyle

Name

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

Synopsis

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

Arguments

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

linestyle – Line style enumeration value. OPVP_LINESTYLE_SOLID (s~~S~~olid line) and~~or~~ OPVP_LINESTYLE_DASH (d~~D~~ashed line) can be set.~~★図は不要か？~~

Description

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

Return Value

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

4.5.24.opvpGetLineStyle

Name

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

Synopsis

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

Arguments

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

plinestyle – Pointer to ~~the buffer to store~~ the line style enumeration value to be returned.

Description

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

The i~~I~~nitial line style ~~in the~~of a Graphics State Object ~~returned by the opvpOpenPrinter() function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.25.opvpSetLineCap

Name

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

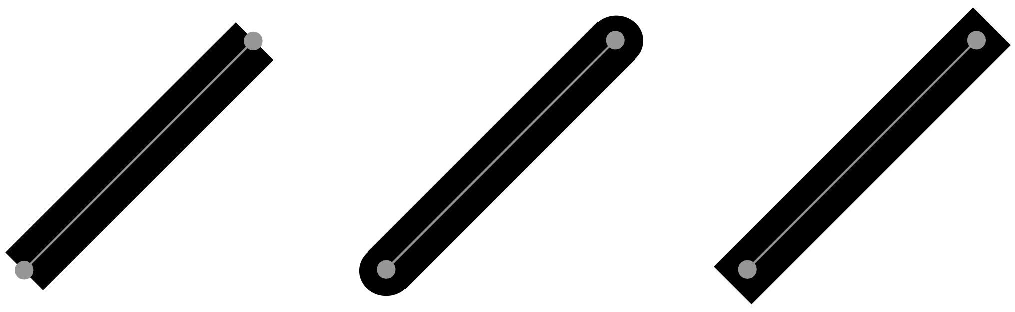
Synopsis

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

Arguments

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

`linecap` – Line cap style enumeration value. `OPVP_LINECAP_BUTT` (b~~B~~utt cap), `OPVP_LINECAP_ROUND` (r~~R~~ound cap) and~~or~~ `OPVP_LINECAP_SQUARE` (s~~S~~quare projection cap) can be set.



Butt cap Round cap Square projection cap

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

Description

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

Return Value

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

4.5.26.opvpGetLineCap

Name

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

Synopsis

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

Arguments

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

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

Description

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

The i~~I~~nitial line cap style ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.27.opvpSetLineJoin

Name

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

Synopsis

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

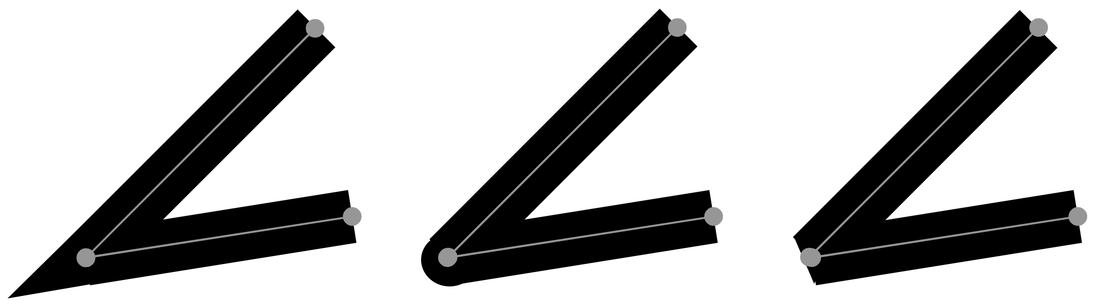
Arguments

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

`linejoin` – Line join style enumeration value. `OPVP_LINEJOIN_MITER` (m~~M~~iter join), `OPVP_LINEJOIN_ROUND` (r~~R~~ound join) ~~or~~and `OPVP_LINEJOIN_BEVEL` (b~~B~~evel join) can be set.

Description

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



Miter join   Round join   Bevel join

Return Value

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

4.5.28.opvpGetLineJoin

Name

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

Synopsis

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

Arguments

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

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

Description

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

The i~~I~~nitial line join style ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

Return Value

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

4.5.29.opvpSetMiterLimit

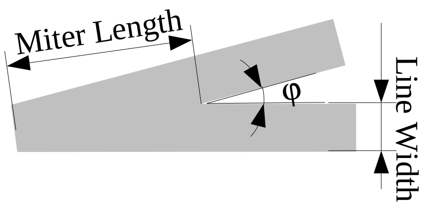
Name

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

Synopsis

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

Arguments



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

`miterlimit` – Maximum miter length.

Description

This function sets the maximum length of the miter ~~to the~~created by a Graphics State Object. The miter limit is effective only when the line join style is `OPVP_LINEJOIN_MITER`. The length MUST be set in ~~the~~ device coordinate system units. ~~説明も反映三原さんの図を入れる~~

~~図は不要か？★~~

Return Value

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

4.5.30.opvpGetMiterLimit

Name

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

Synopsis

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

Arguments

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

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

Description

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

The ~~I~~initial miter limit value ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.

~~依存ドライバ★デフォルト値は？~~

Return Value

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

4.5.31.opvpSetPaintMode

Name

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

Synopsis

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

Arguments

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

`paintmode` – Painting mode enumeration value. `OPVP_PAINTMODE_OPAQUE` (o~~O~~paque mode) ~~or~~and `OPVP_PAINTMODE_TRANSPARENT` (t~~T~~ransparent mode) can be set.

Description

This function sets the background color painting mode ~~to the~~of a Graphics State Object.★~~図は不要か？~~

Return Value

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

4.5.32.opvpGetPaintMode

Name

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

Synopsis

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

Arguments

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

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

Description

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

The ~~i~~Initial painting mode ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.~~依存ドライバ~~★~~デフォルト値は？~~

Return Value

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

4.5.33.opvpSetStrokeColor

Name

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

Synopsis

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

Arguments

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

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

Description

This function 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.★~~合ってるよね？~~ The actual pattern data is specified by `data` array★~~あってる？~~. 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~~in case of error. In the latter case, ~~and~~ the driver MUST store the detailed error code in ~~`errorno`~~`opvpErrorNo`.

4.5.34.opvpSetFillColor

Name

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

Synopsis

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

Arguments

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

`brush` – Pointer to brush structure data.

Description

This function registers the color or pattern 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.

The ~~i~~Initial fill color ~~in the~~of a Graphics State Object ~~returned by the `opvpOpenPrinter()` function depends on each~~is driver dependent.

~~ドライバ依存デフォルト値は？★~~

Return Value

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

4.5.35.opvpSetBgColor

Name

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

Synopsis

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

Arguments

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

brush – Pointer to brush structure data.

Description

This function registers the 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.

The i~~I~~nitial background color ~~in the~~of a Graphics State Object ~~returned by the opvpOpenPrinter() function depends on each~~is driver dependent.

~~ドライバ依存★デフォルト値は？~~

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 ~~errorno~~opvpErrorNo.

Return Value

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

4.6.Path Operations

This sections defines functions for path operations.

A p~~P~~ath is used for ~~the ~~drawing commands~~,~~ such as “stroke”, “fill” and “stroke and fill”, as well as for defining ~~the~~a clipping area. ~~One~~Any Graphics State Object ~~keeps~~maintains only one path at ~~once~~any one time. A p~~P~~ath is affected by the CTM and is registered ~~into~~with the Graphics State Object ~~-~~when the caller calls ~~each~~a path operation function.

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

4.6.1.opvpNewPath

Name

opvpNewPath – Starts a new path.

Synopsis

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

Arguments

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

Description

This function d~~D~~eletes the Graphics State Object's current path~~ kept in the Graphics State Object~~ and starts a new path~~ which.~~. The new path MUST be empty.

The i~~I~~nitial path ~~in the~~of a Graphics State Object ~~returned by the opvpOpenPrinter() function~~ MUST be empty.~~★初期状態のパスは？空でなければならない。~~

Return Value

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

4.6.2.opvpEndPath

Name

opvpEndPath – Declares the end of the current path.

Synopsis

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

Arguments

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

Description

This function d~~D~~eclares the end of the Graphics State Object's current path.

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

Return Value

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

4.6.3.opvpStrokePath

Name

opvpStrokePath – Draws lines along the current path.

Synopsis

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

Arguments

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

Description

This function draws lines along the current path according to the drawing attributes registered ~~in~~with 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~~in case of error. In the latter case, ~~and~~ the driver MUST store the detailed error code in ~~`errorno`~~`opvpErrorNo`.

4.6.4.opvpFillPath

Name

opvpFillPath – Fills the interior of~~inside~~ the current path.

Synopsis

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

Arguments

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

Description

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

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

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

Return Value

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

4.6.5.opvpStrokeFillPath

Name

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

Synopsis

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

Arguments

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

Description

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

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

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

Return Value
OPVP_OK or -1 ~~when~~in case of error. In the latter case,~~ and~~ the driver MUST store the detailed error code in ~~errorno~~opvpErrorNo.

4.6.6.opvpSetClipPath

Name
opvpSetClipPath – Sets the current path as ~~a~~the clipping region.

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

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

clipRule – Clipping rule enumeration value. OPVP_CLIPRULE_EVENODD (e~~E~~ven-odd rule) ~~or~~and OPVP_CLIPRULE_WINDING (n~~N~~on-zero winding number rule) can be set.

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

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

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

The clipping region also MUST be retained ~~except~~until the caller invokes one of:~~calls the~~ opvpSetClipPath(), opvpResetClipPath(), opvpRestoreGS() or opvpStartPage()~~function~~. When the caller calls the opvpStartPage() function, the driver MUST reset the clipping region ~~the printable area of the mediawholein the Graphics State Object to cover~~ as if the opvpResetClipPath() function had been called.★~~表現変更、このような仕様でよいか？~~

Return Value
OPVP_OK or -1 ~~when~~in case of error. In the latter case,~~ and~~ the driver MUST store the detailed error code in ~~errorno~~opvpErrorNo.

4.6.7.opvpResetClipPath

Name
opvpResetClipPath – Resets the clipping region.

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

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

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

Return Value
OPVP_OK or -1 ~~when~~in case of error. In the latter case,~~ and~~ the driver MUST store the detailed error code in ~~errorno~~opvpErrorNo.

4.6.8.opvpSetCurrentPoint

Name
opvpSetCurrentPoint – Sets the current point.

Synopsis

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

Arguments

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

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

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

Description

This function s~~S~~et the current point of the Graphics State Object to ***(x, y)***.

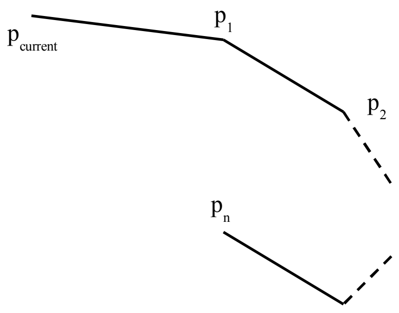
Return Value

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

4.6.9.opvpLinePath

Name

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



Synopsis

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

Arguments

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

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

`npoints` – Number of ~~the `opvp_point_t` structure array~~ elements in `points`.

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

Description

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

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

Structures

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

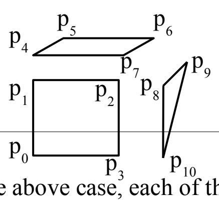
Return Value

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

4.6.10.opvpPolygonPath

Name

opvpPolygonPath – Adds polygons to the current path.



For the above case, each of the arguments is as follows:
npolygons=3
*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 the opvpOpenPrinter() function.

npolygons – Number of polygons to add.

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

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

Description

This function adds the polygons specified ~~by the points array in~~via its arguments to the current path.~~ from the current point.★カレントポイントをどこにあわせて追加する？~~

After ~~driver appends~~the polygons have been added~~ from the current point~~, the driver MUST ~~set~~make the last point of the points array ~~as~~ the new current point.

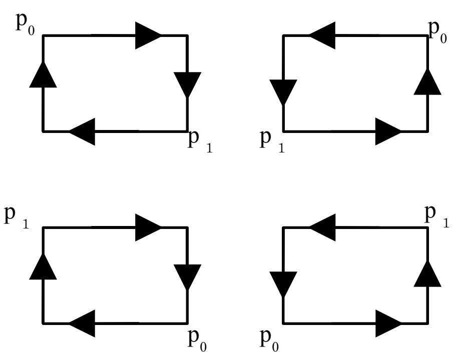
Return Value

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

4.6.11.opvpRectanglePath

Name

opvpRectanglePath – Adds rectangles to the current path.



Synopsis

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

Arguments

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

nrectangles – Number of rectangles to add.

rectangles – Pointer to ~~the~~an opvp_rectangle structure array.

Description

This function adds rectangles ~~in~~to the current path.~~ from the current point.★カレントポイントをどこにあわせて追加?~~

After ~~driver appends~~the rectangles have been appended~~ from the current point~~, the driver MUST ~~set~~make the starting point of the last rectangle appended ~~as~~ the new current point.

The ~~d~~Direction of the paths of each rectangle are specified by the starting~~ point~~ and diagonal points as ~~above~~illustrated in the figure above. The ~~p~~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)*.

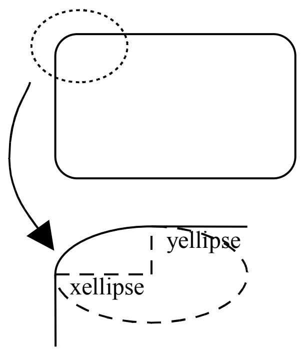
Structures

```
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~~in case of error. In the latter case,~~ and~~ the driver MUST store the detailed error code in ~~errorno~~opvpErrorNo.

4.6.12.opvpRoundRectanglePath



Name

opvpRoundRectanglePath – Adds rounded rectangles to the current path.

Synopsis

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

Arguments

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

`nrectangles` – Number of round rectangles to add.~~★図修正　with, pitch ではない~~

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

Description

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

Each corner of a rounded rectangle is connected by an elliptic arc defined by the values of the `xellip~~p~~se` and `yellipse` ~~in~~members of the `opvp_roundrectangle_t` structure. After ~~driver appends~~the rounded rectangles have been appended ~~from the current point~~, the driver MUST ~~set~~make the starting point of the last rounded rectangle appended ~~as~~ the new current point.~~★カレントポイントは更新するのか？~~

The d~~D~~irection of the paths of each rounded rectangle MUST ~~be treated as~~follow the conventions as specified for the `opvpRectanglePath()` function.

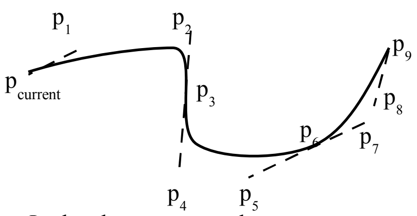
Structures

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

Return Value

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

4.6.13.opvpBezierPath



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

Name

opvpBezierPath – Adds 3D ~~b~~B~~e~~é**zier** paths to the current path.

Synopsis

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

Arguments

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

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

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

Description

This function adds multiple ~~bezier~~Bézier paths specified by the current point and end points and the control points given by the `points` array. The first ~~bezier~~Bézier 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~~Bézier path. Following ~~bezier~~Bézier 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~~Bézier 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 ~~errorno~~opvpErrorNo.

Return Value

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

4.6.14.opvpArcPath

Name

opvpArcPath – Adds 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 the `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.~~三原さんの図★図が欲しいが。~~。

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~~in case of error. In the latter case, ~~and~~ the driver MUST store the detailed error code in ~~`errorno`~~`opvpErrorNo`.

~~★Text は削除~~

4.7.Bitmap Image Operations

This section defines functions for bitmap operations.

A b~~B~~itmap is a pixel oriented image format for~~data which is drawn in a~~ rectangul~~are~~are images~~region~~. A b~~B~~itmap is drawn ~~according to~~using the drawing attributes registered ~~in the~~with a Graphics State Object.

The typical bitmap drawing sequence is as follows:~~ing.~~

- `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 ~~`errorno`~~`opvpErrorNo`.

The ~~—~~`opvpDrawImage()` is the function ~~for~~that perfor~~ming~~ms 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_rectangle_t destinationSize,~~
        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` – ~~Number of~~ Repetition pitch (bytes) of ~~each raster of~~ source image.~~★4の倍数の必要はあるんだっけ？~~★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.

~~`destinationSize` – Destination drawing size.★サイズなのにRectangle？~~

`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.~~widht, height を足したところ★計算式が必要はどこに？~~

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

~~olor になる　　Bw のイメージでビット1は foreground c として RAW のみサポート予約~~★~~capability で取得できるのか？現在は iformatRaw しかないのであれば、現バージョンはそれのみで、後は予約でよいのでは？Image format can be used which the device supports. (OPVP_IFORMAT_RAW, OPVP_IFORMAT_RLE, OPVP_IFORMAT_JPEG, OPVP_IFORMAT_PNG or the vendor supports uniquely can be specified)~~

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.★~~一応確認~~

Return Value

`OPVP_OK` or -1 when error , and the driver MUST store the detailed error code in ~~`errorno`~~`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_rectangle_t destinationSize~~
        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` – Number of bytes of each line of source image.★4の倍数の必要はあるんだっけ？~~

`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.

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

`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 ~~DrawImage に同じ★計算式.はどこに？~~ 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 ~~`errorno`~~`opvpErrorNo`.

~~Image format can be used which the device supports.（OPVP_IFORMAT_RAW, OPVP_IFORMAT_RLE, OPVP_IFORMAT_JPEG, OPVP_IFORMAT_PNG or the vendor supports uniquely can be specified）★capability で取得できるのか？現在は iformatRaw しかないのであれば、現バージョンはそれのみで、後は予約でよいのでは？~~

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.★~~一~~応確認

Return Value

`OPVP_OK` or -1 when error , and the driver MUST store the detailed error code in ~~`errorno`~~`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.~~エラーの場合は End を呼ぶ MUST~~

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 ~~`errorno`~~`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.~~★DrawImage()はいらないよね？~~

Return Value

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

4.8.Scan Line Operations

This section defines functions for scan line operations.

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

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

If the caller calls any ~~functions except~~API entry other than `opvp~~Transfer~~Scanline()` ~~function~~ between the `opvpStartScanline()` and `opvpEndScanline()` functions, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST`~~ in opvpErrorNo~~.

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

4.8.1.opvpStartScanline

Name

`opvpStartScanline` – Declares the s~~S~~tart of a scan line draw~~ing~~ operation.

Synopsis

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

Arguments

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

`yposition` – Vertical position to draw the~~start a~~ scan line.

Description

This function declares th~~eo~~ start of a scan line draw~~ngi~~ operation. Scan lines are drawn ~~by~~using three functions. The `opvpStartScanline()` function declares the start of a scan line~~s~~ draw~~ing~~ operation, the **`opvpScanline()`** function transfers the actual scan line~~s~~ data, and the `opvpEndScanline()` function terminates the scan line draw~~ing~~ operation~~s~~. The d~~D~~river MUST not change the current point after these operations.

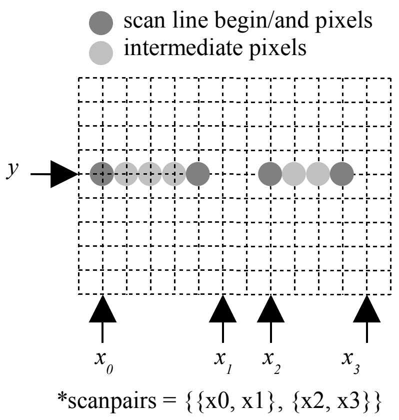
Return Value

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

4.8.2.opvpScanline

Name

`opvpScanline` – Draws the scan lines.



Synopsis

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

Arguments

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

`nscanpairs` – Number of ~~scan lines specified by~~ begin and end ~~positionx~~ point pairs `scanpairs` – Pointer to ~~scan line~~an array ~~specified by~~of begin and end ~~x position~~point pairs

Description

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

~~Y coodinate value と current point の違いを追記~~

Return Value

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

4.8.3.opvpEndScanline

Name

`opvpEndScanline` – Terminates ~~the~~a scan line draw~~ing~~ operation.

Synopsis

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

Arguments

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

Description

This function terminates ~~the~~a scan line draw~~ing~~ operation.

Return Value

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

4.9.Raster Image Operations

This section defines functions for raster image operations.

Raster image operations provide the support for raster image drawing. ~~All~~Driver support for any of the raster image operations ~~are~~is OPTIONAL. However, in case ~~of~~the driver ~~does not~~ supports neither bitmap image ~~operations~~ nor path operations, ~~these~~raster image operations ~~are~~ MUST be supported.

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

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

~~★このような書き方でよいか？~~

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)~~★実装とあっている？~~

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.

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

★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	~~bit order required★どこの定義？~~ padding bits are added to the rightmost byte if necessary
OPVP_CSPACE_DEVICEGRAY	1	8	8	1	
OPVP_CSPACE_STANDARDRGB	3	8	24	3	順番は RGB である
OPVP_CSPACE_STANDARDRGB64	3	16	48	6	~~byte order required★どこの定義？~~
OPVP_CSPACE_DEVICECMY	3	8	24	3	
OPVP_CSPACE_DEVICECMYK	4	8	32	4	
~~cmapIndexed~~	~~1~~	~~4, 8~~	~~4, 8~~	~~1/2, 1~~	~~bit order required★ない？~~

Return Value

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

4.9.2.opvpTransferRasterData

Name

`opvpTransferRasterData` – Transfers 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 the `opvpOpenPrinter()` function.

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

`data` – Pointer to pixel data array.

Description

This function transfers~~count~~ pixel data elements of a single row of raster data. The d~~D~~river MUST ~~.★どうやって判断するのか？ datanon-existentif as ピクセルは描画しない~~ data ~~残りの treat the which exceeds the given count. not draw the remaining data, driver MUST the rasterWidth width given by opvpStartRaster() function スタートラスターで指定された幅 pixel data elements of datathe number of count than sserelgreater the s. If caller setcount スタートラスターで指定された幅 by opvpStartRaster() functiongiven pixelsrasterWidth the the exceeds when the data pixel lengthwhichignore the given data~~ only transfer the lesser of `count` and `rasterWidth` pixels. If `count` is less than `rasterWidth`, the driver MUST not transfer “filler” pixels. In case `count` exceeds `rasterWidth`, the driver MUST ignore the additional pixels.

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

Return Value

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

4.9.3.opvpSkipRaster

Name

`opvpSkipRaster` – Skips lines during raster image drawing.

Synopsis

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

Arguments

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

`count` – Number of lines to be skipped.

Description

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

Return Value

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

4.9.4.opvpEndRaster

Name

`opvpEndRaster` – Terminates ~~the~~a raster image draw~~ing~~ operation.

Synopsis

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

Arguments

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

Description

This function terminates ~~the~~a raster image draw~~ing~~ operation.

Return Value

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

4.10.Stream Data Operations

This section defines functions for stream data operations.

Stream data operations provide ~~the~~ support ~~that an~~for applications ~~that~~~~directly~~ create~~s a~~ printer native PDL data and sends th~~is~~e data to the~~a~~ device directly. Driver support for any of the~~All~~ stream data operations ~~are~~is 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 the caller calls any ~~functions except~~API entries other than `opvpTransferStreamData()` ~~function~~ between the `opvpStartStream()` and `opvpEndStream()` functions, the driver MUST return an error and set the detailed error code to `OPVP_BADREQUEST` ~~in opvpErrorNo~~.

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

4.10.1.opvpStartStream

Name

opvpStartStream – Declares the ~~s~~Start of a stream~~ing~~ data transfer.

Synopsis

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

Arguments

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

Description

This function declares the start~~s~~ of a stream~~ing~~ data transfer. Stream~~ing~~ data is ~~the~~ data that is ~~directly~~ sent to the printer device directly without ~~being~~ any process~~ed~~ing by the driver.

Return Value

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

4.10.2.opvpTransferStreamData

Name

opvpTransferStreamData – ~~s~~Sends stream data.

Synopsis

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

Arguments

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

`count` – Number of data bytes to transfer.

`data` – Pointer to the stream data.

Description

This function sends ~~the~~ stream~~ing~~ data directly to the printer device.

Return Value

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

4.10.3.opvpEndStream

Name

opvpEndStream – Declares the end of a streaming~~ing~~ data transfer.

Synopsis

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

Arguments

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

Description

This function terminates ~~the~~a stream~~ing~~ data transfer.

Return Value

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

5.Graphic Operation Fallback

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

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

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

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

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

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

6.Macros, Types, Enumerations 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 */
                                  /*    that cannot be handled by driver or printer */
#define OPVP_JOBCANCELED   -5     /* error: job has been canceled by some cause */
#define OPVP_PARAMERROR    -6     /* error: invalid parameter */
```

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~~.~~ellaneous 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_MEDIASIZE        = 0x00010000,/* only for opvpQueryDeviceInfo */★これも？~~
~~  OPVP_QF_PRINTREGION      = 0x00020000 /* only for opvpQueryDeviceInfo */~~
  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~~Following is a sample header file which can be used by driver ~~or~~and renderer implementations based on this specification.

```
~~/*~~
~~ * OpenPrinting Vector Printer Driver API Definitions [opvp.h]~~
~~ *~~
~~ * Copyright (c) 2006 Free Standards Group~~
~~ * Copyright (c) 2006 Fuji Xerox Printing Systems Co., Ltd.~~
~~ * Copyright (c) 2006 Canon Inc.~~
~~ * Copyright (c) 2003-2006 AXE Inc.★Copyrightはつけて良いか？~~
~~ *~~
~~ * All Rights Reserverd.~~
~~ *~~
~~ * Permission to use, copy, modify, distribute, and sell this software~~
~~ * and its documentation for any purpose is hereby granted without~~
~~ * fee, provided that the above copyright notice appear in all copies~~
~~ * and that both that copyright notice and this permission notice~~
~~ * appear in supporting documentation.~~
~~ *~~
~~ * The above copyright notice and this permission notice shall be~~
~~ * included in all copies or substantial portions of the Software.~~
~~ *~~
~~ * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,~~
~~ * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF~~
~~ * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND~~
~~ * NONINFRINGEMENT.  IN NO EVENT SHALL THE OPEN GROUP BE LIABLE FOR~~
~~ * ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF~~
~~ * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION~~
~~ * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.~~
```

```
~~*/~~
~~/*~~
~~ 2007 Modified  for OPVP 1.0 by BBR Inc.~~
~~*/~~

#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~~that cannot be handled by driver or printer
*/
#define OPVP_JOBCANCELED    -5      /* error: job has been canceled by some cause */
#define OPVP_PARAMERROR     -6      /* error: invalid parameter */

/* 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~~.~~ellaneous 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 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,
~~00,     /* only for opvpQueryDeviceInfo */~~
~~  OPVP_QF_PRINTREGION      = 0x00020000         /* only for opvpQueryDeviceInfo */100=~~
~~0x000          SIZE  OPVP_QF_MEDIA~~
  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*,~~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_cspace_t*,opvp_int_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 Contribut~~e~~ors

7.1.Editors

Osamu Mihara – Fuji Xerox Co., Ltd.

7.2.Authors

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

7.3.Contribut~~e~~ors

(~~a~~Alphabetical order) Hidekazu Hagiwara (MintWave), Masaki Iwata (AXE), Hidenori Kanjo (BBR), Shinpei Kitayama (~~Epson~~EPSON ~~Kowa~~~~Avasys~~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), Ak~~i~~eo Maruyama (Ricoh), Olaf Meeuwissen (EPSON AVASYS), ★小笠原さん追加しても良いか小笠原さんに確認 Hisao Nakamura (E&D), Koji Otani (~~AXE~~BBR), Kenji Wakabayashi (MintWave), Toshihiro Yamagishi (Turbolinux), Akira Yoshiyama (NEC)

8.History

Version 0.1 (Japanese) Mihara/Toratani

Version 0.2 (Japanese) Mihara

~~O~~openprinting-0.1.1 implementation by opfc~~ have implemented based on this version~~

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

Translation to English
~~Some f~~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/constan~~d~~ts/enumerations/structures names to add FSG prefixes.

Change copyright year and dates

Delete "Printer Driver Database"

Delete *cmap* from CSPASE chart.

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

Change description for fsgpdStartJob/fsgpdEndJob/fsgpdAbortJob

Add type FSGPD_CHAR for character type

Version 1.0 RC2 2006-10-20

Add FSGPD_CSPACE_DEVICEKRGB

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

Remove "described later" descriptions

Make support of updf for scheme "MUST."

Version 1.0 RC2 2006-12-05

Format: Add line numbers and chapter numbers.

Format: Replace Microsoft Expressions object to OOo Expressions

Version 1.0 RC~~2~~3 2007-5-~~9~~19 Toratani

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

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

Rearrange the document sections, chapters and text forma~~n~~t.

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