OpenPrinting

Vector Printer Driver Application Program Interface (PDAPI) Specification

Version-1.0 RC1

(2005-7-20)

OpenPrinting Vector Printer Driver Application Interface Specification

Table of Contents

1

I.Introduction

This specification defines Application Program Interface (API) which is used in printing environment offered by OpenPrinting. Here, the word “printer driver” refers software library which translate printing data generated by application programs into data stream which can be interpreted by printers. The driver offers functions for translating each drawing commands.

This specification tries to make abstraction of graphics drawing functions which is supported by printer languages and offers them as API, and make enable to access printer graphics drawing functions without printer model specific knowledge by render.

This specification covers from ink jet printers which handles raster data only to high-end laser printers which have high-level graphics functions. Specifically, by providing access method to high level graphics drawing function, it improves printing performance. The specification also covers black and white to full color printers.

As examples of render, there exist Ghostscript and X Print Server. The printer driver based on this specification does not assume any specific type of render.

II.Linking with Render

Loading and Calling Printer Driver

Printer drivers can be provided in forms of static or dynamic library. Liking to the library module will be followed by methods prepared by operating systems, therefore, it is not specified in this document.

Printer drivers can be loaded as a library code into the same memory space of render. It is also executed as server process which communicate with render via RPC call (the driver is loaded into different memory space of render). In case of server type, the server process links printer drivers. The specification for RPC, other documents should care of that.

Printing data stream, which is generated by printer driver corresponding to each drawing APIs, are send back to render through file descriptor, which is designated by OpenPrinter() call. The printer driver does not need to generate data stream API by API, rather than, it may generates data stream asynchronously to API call or generates whole data at EndPage() call. In order to receive the steam data properly, the render should handle input stream from the driver in any timing between OpenPrinter() and ClosePrinter() calls.

Printer Driver Database (TBD)

The printer driver name and printer model names which are supported by the driver will be under control of driver data base. The driver data base will be supplied in UPDF and/or PPD format.

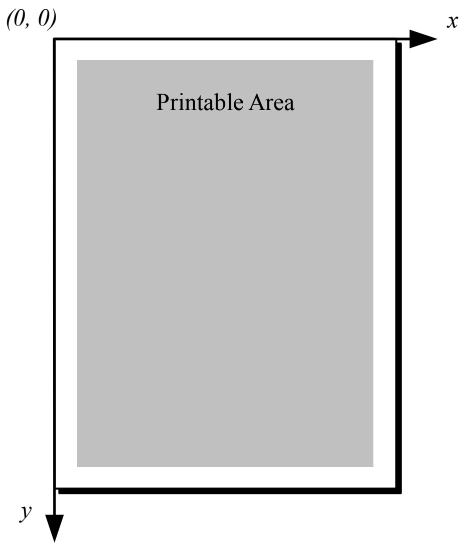
${BASE_DIR}/share/OpenPrinting/<driver-name>xml or ${BASE_DIR}/share/OpenPrinting/<driver-name>ppd

Notes about Function Parameters

In case that a driver is linked as a library, data area which is designated by pointers of arguments of each functions are possibly referred until the process of the page on which drawing functions are executed. Therefore, the caller should keep the area until it calls EndPage() operation. In addition, deallocation from memory of such area should be done by the caller after EndPage() operation.

III.Graphic Model

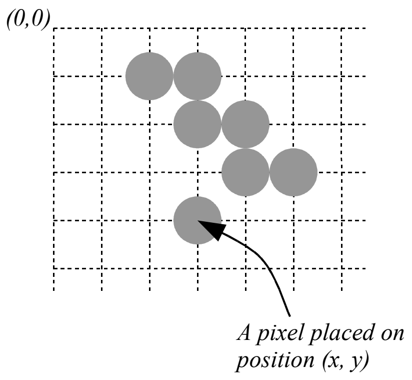
Coordinate System



It takes the coordinate system such that the origin takes physical upper left corner of media and the unit is device pixel. Positive value in abscissa moves origin to right direction, positive value in ordinate moves origin to lower direction.

The type of coordinate takes singed, fixed point 32 bits value, where integer takes 24 bits and decimal takes 8 bits (type "Fix").

A coordinate of a pint shows the horizontal and vertical distance from origin to intersection of grid. Pixels are assumed to be placed on intersections of grid (Grid Intersection Model). The relationship between coordinate grid and pixels are shown in the right figure.



Objects

Following kinds of graphics objects are recognized and handled under this specification.

- Path
- Bitmap Image
- Scanline
- Raster Image
- Text

The printer driver maintains "Graphics State", in which contains properties for drawing graphics. Only one graphics state is effective at one time, however, it can be switched by SaveGS() and RestoreGS().

CTM

Coordinate Translation Matrix (CTM), which is used to translate coordinate system to render coordinate to device coordinate, is maintained in Graphics State. CTM is presented in following matrix form.

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

For detail of CTM, refer description in ResetCTM/SetCTM/GetCTM.

Color

As Color mapping, cmapDirect (direct color) and cmapIndexed (indexed color) will be supported. At this time, cmapIndexed is pending.

Following color space can be used.

Color Space Macro	Color Space
cspaceBW	Black and White
cspaceDeviceGray	Grayscale
cspaceDeviceCMY	CMY
cspaceDeviceCMYK	CMY and Black
cspaceDeviceRGB	Device RGB
cspaceStandardRGB	sRGB
cspaceStandardRGB64	scRGB

At this time cspaceBW, cspaceDeviceGray and cspaceStandardRGB will be supported.

Alpha Composition

This specification supports Alpha Composition. Alpha value which is saved in the Graphics State and is effective to whole drawing and alpha for pixel by pixel of bitmap data which is represented by aRGB format.

(However, alpha is not supported at this moment – we need more consideration on this)

Scan Rule

Pixel placement on painting of path and scan line operation should be done without painting problem which is sometimes occurred as a result of raster operation when painting a region by dividing into small polygon regions. As an example of scan rule, there is right-bottom exclusive method. For actual implementation, it depends on page description language, therefore this specification does not determine specific implementation.

IV.Operations

Creating and Managing Print Contexts

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

OpenPrinter

Name

OpenPrinter – Create printer context.

Synopsis

```
int OpenPrinter(
        int outputFD,
        char *printerModel,
        int *nApiEntry,
        int (**apiEntry[])());
```

Arguments

outputFD – file descriptor to write print data stream

printerModel – Printer Model Name in UTF-8. If NULL, it assumes default model by the printer driver.

nApiEntry – array size of apiEntry.

apiEntry – address of function pointer array to each API entry.

Description

This function initializes printer driver and designates the file descriptor for writing printing data stream. Printing data generated by drawing function and others are written through this file descriptor. For printerModel, the printer model which is defined in printer model data base.

The driver writes debug messages to stderr, so stderr cannot be designated

OpenPrinter() stores function pointers for each API entry to apiEntry and returns. NULL will be stored for functions not supported by the driver. Array size is set to nApiEntry.

Only OpenPrinter() is exported by printer driver library. For other APIs, they are referred by entry addresses stored in apiEntry.

When it finished normally, printer context is returned as return value. The printer context is a index number or pinter value to manager opened printer and is designated as first parameter for APIs thereafter. So, the caller should remember this value.

Return Value

If successful, this function returns printer context value (positive value). This function returns -1 when error is occurred and error code is stored in errorno.

ClosePrinter

Name

ClosePrinter – Delete printer context.

Synopsis

```
int ClosePrinter(
        int printerContext);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

Description

It finished printing process and deletes printer context. Closing of outputFD, which is designated at OpenPrinter(), should be closed by caller side.

If this function is called when printing job is not completed, such as a case no EndJob() is called, data in process is discarded, but it does not guarantee that printing data which is cancel printing job normally is generated.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

Job Control Operations

These functions operate job control. These functions MUST be supported by all drivers. One print job consist of one or more documents. One document consist of one or more pages. If a job consist of only one document, StartDoc() and EndDoc() functions can be omitted. StartJob(), EndJob(), StartPage() and EndPage() functions cannot be omitted.

StartJob

Name

StartJob – Start print job.

Synopsis

```
int StartJob(
        int printerContext,
        char *jobInfo);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

jobInfo – Set property of print job. If NULL, default values by printer driver are set. Format and contents are described later.

Description

StartJob() starts print job.

Property of the print job is set by jobInfo.

It is depend on printer device characteristics whether job nesting (StartJob() - EndJob() are called between StartJob() - EndJob()) is possible. If the printer does not allow job nesting, the driver returns error for this call.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

EndJob

Name

EndJob –End print job.

Synopsis

```
int EndJob(
        int printerContext);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

Description

EndJob() finishes print job.

Job property values are reset to initial values.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

StartDoc

Name

StartDoc – Start document.

Synopsis

```
int StartDoc(
        int printerContext,
        char *docInfo);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

docInfo – Set property of print document. If NULL, default values by printer driver are set. Format and contents are described later.

Description

StartDoc() starts printing of a document.

Document property is set through docInfo. The document property is effective until EndDoc() is called. When StartDoc() is called again after the EndDoc() call, the docInfo at this call is effective and properties not in docInfo are reset to initial value.

It is depend on printer device characteristics whether document nesting (StartDoc() - EndDoc() are called between StartDoc() - EndDoc()) is possible. If the printer does not allow document nesting, the driver returns error for this call

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

EndDoc

Name

EndDoc – End document

Synopsis

```
int EndDoc(
        int printerContext);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

Description

EndDoc() ends document printing.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

StartPage

Name

StartPage – Start print page.

Synopsis

```
int StartPage(
        int printerContext,
        char *pageInfo);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

pageInfo – Page property. If NULL, default values by printer driver are set. Format and contents are described later.

Description

StartPage() starts printing of a page.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

EndPage

Name

EndPage – End print page.

Synopsis

```
int EndPage(
      int printerContext);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

Description

EndPage() ends a page printing. After this function is called, memory area which is allocated by caller can be released.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

QueryDeviceCapability

Name

QueryDeviceCapabilit y – Query device capabilities.

Synopsis

```
int QueryDeviceCapability(
      int printerContext,
      int queryflag,
      int buflen,
      char *infoBuf);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

queryflag – flags showing device capability queried.

buflen – buffer length which is prepared as *infoBuf

infoBuf – a buffer to put device capability information

Description

This function is used to query capabilities which is supported by printer device or driver.

If infoBuf is set to NULL, the printer driver returns buffer length to save information in int format In this case, the caller should queryflag and buflen correctly.

Queryflag is used to set bit flag to be queried. More than two flags can be set. Following macro should be used.

```
typedef enum _pdQueryInfoFlags {
      pdQFDeviceResolution =     0x00000001,
      pdQFMediaSize =            0x00000002,
      pdQFPageRotation =         0x00000004,
      pdQFMediaNUp =             0x00000008,
      pdQFMediaDuplex =          0x00000010,
      pdQFMediaSource =          0x00000020,
      pdQFMediaDestination =     0x00000040,
      pdQFMediaType =            0x00000080,
      pdQFMediaCopy =            0x00000100,
      pdQFPrintRegion =          0x00010000   // only for QueryDeviceInfo
} pdQueryInfoFlags;
```

The driver returns queried information in ASCII text format. If buffer doesn't have enough length to write, the written information is truncated. In this case, the driver doesn't set error code and returns without error.

The format of information written in infoBuf follows the property format which is used by StartJob() and other functions. For example, if resolution is queried, the resolution list supported is written in following format. The first one of the list is default setting.

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

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

QueryDeviceInfo

Name
QueryDeviceInfo – Query device information.

Synopsis
```
int QueryDeviceInfo(
        int printerContext,
        int queryflag,
        int buflen,
        char *infoBuf);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

queryflag – flags showing device capability queried.

buflen – buffer length which is prepared as *infoBuf

infoBuf – a buffer to put device capability information

Description
QueryDeviceInfo() is used to query current setting of printer device or driver.

If infoBuf is set to NULL, the printer driver returns buffer length to save information in int format In this case, the caller should queryflag and buflen correctly.

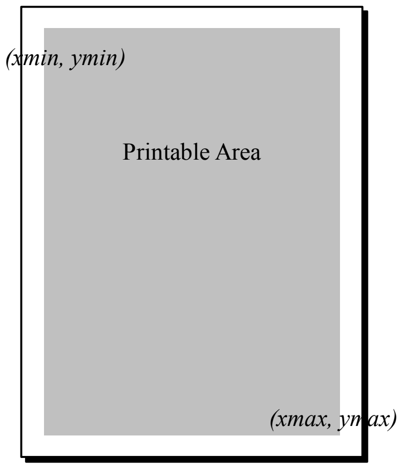
Queryflag is used to set bit flag to be queried. More than two flags can be set. Same macro which is used in QueryDeviceCapability() should be used.

The driver returns queried information in ASCII text format. If buffer doesn't have enough length to write, the written information is truncated. In this case, the driver doesn't set error code and returns without error.

The format of information written in infoBuf follows the property format which is used by StartJob() and other functions.

Query about printable area is respond in current printing resolution and format is shown as follow (see the figure below.) These values follows in orientation setting.

PrintRegion=xmin,ymin,xmax,ymax



Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

Attribute of Job, Document and Page for Job Control Operations

Properties for job, document and page are set as parameter of StartJob(), StartDoc(), StartPage(). Supported parameters and their values are provided by printer driver database file distributed with drivers.

The property values are set in ASCII strings, thus follows the format below.

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

- *<scheme>*: The specification which defines name space.
- *<key>*: Name of the property.
- *<value>*: Value of the property.

Several *<value>* can be designated separating by ”,” (comma). If several values are designated, the printer driver searches it from begging of the string and takes possible value.

Several pairs - *<key>*=*<value>*{,*<value>*}* can be designated separating by ”;” (semi-colon)

For *<scheme>, “*updf” can be designated.

If scheme is updf, the format follows the definition by UPDF (Universal Printer Driver Description File) by Printer Working Group.

It if preferable that the driver ignores unknown property, for future extensions.

Major properties are shown in the table below.

Property	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	✔	✔	✔

Property	Name	Value	Effective in		
			Job	Doc	Page
		...			
Media Copy	MediaCopy	1, 2, ...			
Print Quality	PrintQuality	draft high normal	✔	✔	✔

The values are always effective in order of Job < Doc < Page. For example, in a job which is set to landscape, if a page is set to portrait, the page is set to portrait and other pages are set to landscape.

Graphics State Object Operations

ResetCTM

Name

ResetCTM – Initialize CTM of the Graphics State Object.

Synopsis

```
int ResetCTM(
        int printerContext);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

Description

ResetCTM() initializes CTM of Graphics State Object. The initial setting of CTM is as follow.

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

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetCTM

Name

SetCTM – Set CTM to the Graphics State Object.

Synopsis

```
int SetCTM(
        int printerContext,
        CTM *pCTM);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

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

Description

SetCTM() sets a CTM to the Graphics State Object.

Device coordinate system value [xdev, ydev] is represented by CTM and render coordinate system value [xren, yren] in 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 _CTM {
        float a, b, c, d, e, f;
} CTM;
```

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetCTM

Name

GetCTM – Get CTM from the Graphics State Object.

Synopsis

```
int GetCTM(
        int printerContext;
        CTM *pCTM);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

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

Description

GetCTM() retrieves a CTM from Graphics State Object.

Structures

```
typedef struct _CTM {
        float a, b, c, d, e, f;
} CTM;
```

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

InitGS

Name

InitGS – Initialize the Graphics State parameters.

Synopsis

```
int InitGS(
        int printerContext);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

Description

InitGS() initializes parameters in Graphics State Object. Graphics state is a set of values which is managed by printer driver and contains parameters for drawing mode and others. There is always one Graphics state object which is effective as current graphics state and used as current setting. Graphics state object can be pushed or popped on/from driver managed stack by SaveGS() and RestoreGS() operations described later in this document. These operations are used when changing graphics state temporary and restores later.

Graphics state object keeps its setting values between StartJob() and EndJob(), unless InitGS() is called. At StartJob(), same values as InitGS() is called are set.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SaveGS

Name

SaveGS – Save the Graphics State parameters.

Synopsis

```
int SaveGS(
        int printerContext);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

Description
SaveGS() pushes graphics state object parameters onto the driver managed stack.

The size of the stack is limited, but at least n times of this operation can be called (TBD). If more than n times this function is called without calling RestoreGS(), error is returned.

After n times SaveGS() is called, same times RestoreGS() can be executed. More than n time RestoreGS() is called, it results in error (BADREQEST).

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

RestoreGS

Name
RestoreGS – Restore the Graphics State parameters.

Synopsis
```
int RestoreGS(
        int printerContext);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

Description
Saved graphics state object is restored to current value.

After n times SaveGS() is called, same times RestoreGS() can be executed. More than n time RestoreGS() is called, it results in error (BADREQEST).

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

QueryColorSpace

Name
QueryColorSpace – Obtain the color space which can be used by the driver.

Synopsis
```
int QueryColorSpace(
        int printerContext,
        ColorSpace *pcspace
        int *pnum);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

pcspace –Pointer to the Color Space enum. array.

pnum – Pointer to the number of the Color Space enum. array elements.

Description
Retrieves list of color space which can be treated by the driver from graphics state object.

The caller should allocate at least n (TBD) as the number of elements of an array pcspace[] which stores color space. Also the caller should call this function by setting it's maximum number of elements in *pnum. The number of elements retrieved is set to *pnum. If the number of elements exceeds the number of elements designated, this function returns necessary number in *pnum.

The driver returns the color space into array in preferable order.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetColorSpace

Name
SetColorSpace – Set the current color space which can be dealt by the driver to the Graphics State Object.

Synopsis
```
int SetColorSpace(
        int printerContext,
        ColorSpace cspace);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

cspace –Color Space enum.

Description
SetColorSpace() sets color space handled by the driver into graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetColorSpace

Name
GetColorSpace – Get the current color space from the Graphics State Object.

Synopsis
```
int GetColorSpace(
        int printerContext,
        ColorSpace *pcspace);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

pcspace – Pointer to the Color Space enum.

Description
GetColorSpace() retrieves color space treated by the driver from graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

QueryROP

Name
QueryROP – Obtain the ROP which can be used by the driver.

Synopsis
```
int QueryROP(
        int printerContext,
        int *pnum,
        int *prop);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

pnum – Pointer to the number of the ROP enum. array elements.

prop –Pointer to the ROP mode. array.

Description
QueryROP retrieves a list of ROP (Raster Operations) which can be treated by the driver from graphics state object.

If NULL is set in *prop, the number of ROP supported is returned in *pnum.

So, at least the number of ROP supported by the operation above should be allocated for the array prop[].

The number of elements is returned in *pnum. If the number of ROP exceeds the size of array designated, it returns an error (PARAMERROR) and necessary number of elements is returned in *pnum.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetROP

Name
SetROP – Set the ROP mode to the Graphics State Object.

Synopsis
```
int SetROP(
        int printerContext,
        int rop);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

rop –ROP mode (ROP3 code)

Description
SetROP() sets raster operation code designated by rop to graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetROP

Name

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

Synopsis
```
int GetROP(
        int printerContext,
        int *prop);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

prop – Pointer to he ROP mode.

Description
GetROP() retrieves raster operation code currently set in graphics state.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetFillMode

Name
SetFillMode – Set the filling mode.

Synopsis
```
int SetFillMode(
        int printerContext,
        FillMode fillmode);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

fillmode – Filling mode enum. Either of fillModeEvenOdd, fillModeWinding can be designated.

Description
SetFillMode() sets a fill mode to graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetFillMode

Name
GetFillMode – Get the filling mode.

Synopsis
```
int GetFillMode(
        int printerContext,
        FillMode pfillmode);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

pfillmode – Pointer to the filling mode enum.

Description
GetFillMode retrieves the fill mode currently set in graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetAlphaConstant

Name
SetAlphaConstant – Set the alpha blending constant.

Synopsis
```
int SetAlphaConstant(
        int printerContext,
        float alpha);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

alpha – Alpha blending constant. Between 0.0 and 1.0

Description
Set alpha constant, which is transparent ratio, to the graphics state object. The value should be in range of 0.0 to 1.0. If a value extends this range, it will be truncated to 0.0 or 1.0.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetAlphaConstant

Name
GetAlphaConstant – Get the alpha blending constant.

Synopsis
```
int GetAlphaConstant(
        int printerContext,
        float *palpha);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

palpha – Pointer to the alpha constant value.

Description
GetAlphaConstant() retrieves alpha constant setting from graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetLineWidth

Name
SetLineWidth – Set the line width.

Synopsis
```
int SetLineWidth(
        int printerContext,
        Fix width);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

width – Line width value.

Description
SetLineWidth() sets line width for stroke operations to graphics state object. The line width should be set by units in device coordinate system.

The treatment of a line width less than one depends on the device or driver implementation.

The maximum line width depends on device.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetLineWidth

Name
GetLineWidth – Get the line width.

Synopsis
```
int GetLineWidth(
        int printerContext,
        Fix *pwidth);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

width – Pointer to the Line width value.

Description
GetLineWidth() retrieves line width for stroke operations to graphics state object. The line width should be set by units in device coordinate system.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetLineDash

Name
SetLineDash – Set the line dash pattern.

Synopsis
```
int SetLineDash(
        int printerContext,
        Fix pdash[],
        int num);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

pdash – Pointer to the line dash pattern array.

num – Number of the line dash pattern array elements.

Description
SetLineDash() sets a dash pattern to graphics state object.

When the painting mode is paintOpaque, the first element in the array pdash[] is the length for foreground color, the next element is a length for background color, and so on.

When the painting mode is paintTransparent, the first element in the array pdash[] is the length for foreground color, the next element shows the length of a line segment which is NOT painted, and so on.

The length designated in the array pdash[] is designated by the unit in the device coordinate system.

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

The maximum number of elements depend on the device.

If num is zero, solid lines are drawn.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetLineDash

Name
GetLineDash – Get the line dash pattern.

Synopsis
```
int GetLineDash(
        int printerContext,
        Fix pdash[],
        int *pnum);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

pdash – Pointer to the line dash pattern array.

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

Description

GetLineDash() retrieves dash pattern for stroke from graphics state object.

The caller should allocate at least n (TBD) elements for array pdash[]. Also, the caller should set the maximum number of element of pdash[] to *pnum. The number elements returned is set in *pnum. If the number of elements exceeds the *pnum set by the caller, it returns error and necessary number of elements are set to *pnum.

If odd elements are set as dash pattern, odd number of elements are returned. If even elements are set, even number of elements are returned.

If NULL is set to pdash, it returns necessary number of elements for pdash[] in *pnum.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetLineDashOffset

Name

SetLineDashOffset – Set the line dash pattern offset.

Synopsis

```
int SetLineDashOffset(
        int printerContext,
        Fix offset);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

offset – Offset value for applying the line dash pattern.

Description

SetLineDashOffset() sets offset for dash pattern of stroke operations in units of device coordinate system.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetLineDashOffset

Name

GetLineDashOffset – Get the line dash pattern offset.

Synopsis

```
int GetLineDashOffset(
        int printerContext,
        Fix *poffset);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

poffset – Pointer to the offset value for applying the line dash pattern.

Description

GetLineDashOffset() retrieves offset value of dash pattern from graphics state object.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetLineStyle

Name
SetLineStyle – Set the line style.

Synopsis
```
int SetLineStyle(
       int printerContext,
       LineStyle linestyle);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

linestyle – Line style enum. LineStyleSolid or lineStyleDash can be designated.

Description
SetLineStyle() sets line style to graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetLineStyle

Name
GetLineStyle – Get the line style.

Synopsis
```
int GetLineStyle(
       int printerContext,
       LineStyle *plinestyle);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

plinestyle – Pointer to the line style enum.

Description
GetLineStyle() retrieves line style for stroke from graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetLineCap

Name
SetLineCap – Set the line cap style.

Synopsis
```
int SetLineCap(
       int printerContext,
       LineCap linecap);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

linecap – Line cap style enum. Either of lineCapButt, lineCapRound, lineCapSquare can be set.

Description
SetLineCap() sets the style for line cap to graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetLineCap

Name
GetLineCap – Get the line cap style.

Synopsis
```
int GetLineCap(
        int printerContext,
        LineCap *plinecap);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

plinecap – Pointer to the line cap style enum.

Description
GetLineCap() retrieves the style for line cap from graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetLineJoin

Name
SetLineJoin – Set the line join style.

Synopsis
```
int SetLineJoin(
        int printerContext,
        LineJoin linejoin);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

linejoin – Line join style enum. Either of lineJoinMiter, lineJoinRound and lineJoinBevel can be set. Default is (TBD).

Description
SetLineJoin() sets the line joining method to graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetLineJoin

Name
GetLineJoin – Get the line join style.

Synopsis
```
int GetLineJoin(
        int printerContext,
        LineJoin *plinejoin);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

plinejoin – Pointer to the line join style enum.

Description
GetLineJoin() retrieves the line joining method from graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetMiterLimit

Name
SetMiterLimit – Set the miter limit value.

Synopsis
```
int SetMiterLimit(
        int printerContext,
        Fix miterlimit);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

miterlimit – Maximum miter length.

Description
SetMiterLimit() sets the maximum length of miter to graphics state object. The miter limit is effective only when line joining style is lineJoinMiter. The unit is in device coordinate system.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetMiterLimit

Name
GetMiterLimit – Get the miter limit value.

Synopsis
```
int GetMiterLimit(
        int printerContext,
        Fix *pmiterlimit);
```

Arguments
printerContext – printer context value returned by OpenPrinter()

pmiterlimit – Pointer to the maximum miter length.

Description
GetMiterLimit() retrieves the maximum length of miter from graphics state object.

Return Value
If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetPaintMode

Name
SetPaintMode – Set the background color painting mode.

Synopsis
```
int SetPaintMode(
        int printerContext,
        PaintMode paintmode);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

paintmode – Painting mode enum. Either of paintModeOpaque and paintModeTransparent can be set. Default is (TBD).

Description

SetPaintMode() sets paint mode for background to graphics state object.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

GetPaintMode

Name

GetPaintMode – Get the background color painting mode.

Synopsis

```
int GetPaintMode(
        int printerContext,
        PaintMode ppaintmode);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

ppaintmode – Pointer to the painting mode enum.

Description

GetPaintMode() retrieves paint mode for background from graphics state object.

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetStrokeColor

Name

SetStrokeColor – Define stroke color and pattern for StrokePath() and StrokeFillPath() operations.

Synopsis

```
int SetStrokeColor(
        int printerContext,
        Brush *brush);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

brush – Pointer to brush definition.

Description

SetStrokeColor() registers the color or pattern for stroke drawing of StrokePath() and StrokeFillPath() operations as a brush to graphics state object.

If pbrush in Brush structure is set to NULL, it specifies a solid brush. In this case, color of the solid brush is specified in color[4] in the color space by colorSpace.

If the pointer to the BrushData structure is set in pbrush, it specifies pattern brush. In the BrushData structure, width, hight and repetition pitch are specified in pixel value and pointer to the actual pattern data is specified by data. T[illegible]lor space for the pattern is specified by ColorSpace in Brush structure. In this case, color[4] specifies foreground color.

Structures

```
typedef struct _Brush {
        ColorSpace colorSpace;
        int color[4];             // aRGB quadruplet
        int xorg, yorg;           // brush origin
```

```
                            // ignored for SetBgColor
        BrushData *pbrush;  // pointer to brush data
                            // solid brush used, if null
} Brush;

typedef struct _BrushData {
        BrushDataType type;
        int width, height, pitch;
        char data[];        // data[1] for GCC 2.x
} BrushData;

typedef enum _BrushDataType {bdtypeNormal = 0} BrushDataType;
```

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetFillColor

Name

SetFillColor – Define fill color and pattern for FillPath() and StrokeFillPath() operations.

Synopsis

```
int SetFillColor(
        int printerContext,
        Brush *brush);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

brush – Pointer to brush definition.

Description

SetFillColor() registers the color or pattern for FillPath() and StrokeFillPath() operations as a brush to graphics state object.

If successful, this function return OK. If an error is occurred

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

SetBgColor

Name

SetBgColor – Define background color and pattern for StrokePath(), FillPath() and StrokeFillPath() operations.

Synopsis

```
int SetBgColor(
        int printerContext,
        Brush *brush);
```

Arguments

printerContext – printer context value returned by OpenPrinter()

brush – Pointer to brush definition.

Description

SetBgColor() registers the background or pattern for StrokePath(), FillPath() and StrokeFillPath() operations as brush.

The Members colorSpace and color[4] of the BRUSH structure are referred as brush data.

Xorg and yorg are ignored. The caller should set NULL to pbrush, otherwise it returned as an error (BADREQUEST).

Return Value

If successful, this function return OK. If an error is occurred, it returns -1 and stores error code in errorno.

Path Operations

Path is used for the drawing command, such as “stroke”, “fill” and “stroke and fill” as well as for defining the clipping area. One Graphics State Object has only one path. Path is effected by CTM, and registered into the Graphics State along the CTM when calling each path operation function.

All path operation functions are OPTIONAL.

NewPath

Name

NewPath – start a new path

Synopsis

```
int NewPath(
        int printerContext);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

Description

Delete the current path and start a new path.

Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

EndPath

Name

EndPath – end current path

Synopsis

```
int EndPath(
        int printerContext);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

Description

Declare the end of the current path, and the Graphics State Object retains the current path.

Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

StrokePath

Name

StrokePath – stroke current path

Synopsis

```
int StrokePath(
        int printerContext);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

Description

Draw lines along the current path. The lines are drawn along the drawing attributes registered in the Graphics State Object when calling this function. The Graphics State Object retains the current path.

Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

FillPath

Name

FillPath – fill current path

Synopsis

```
int FillPath(
        int printerContext);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

Description

Fill in the current path with the filling mode registered in the Graphics State Object. Filling is done along the “right and bottom exclusive” method to avoid a problem caused by filling the adjoining area with ROP. The way of how to draw each pixel depends on the driver implementation.

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

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

Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

StrokeFillPath

Name

StrokeFillPath – stroke and fill current path

Synopsis

```
int StrokeFillPath(
        int printerContext);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

Description

Draw lines along the current path, and fill in the current path.

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

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

Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

SetClipPath

Name

SetClipPath – Set current path as clipping region

Synopsis

```
int SetClipPath(
        int printerContext,
        ClipRule clipRule);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

clipRule – Clipping rule enum. clipRuleEvenOdd or clipRuleWinding

Description

Set the current path as the clipping region.

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

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

Clipping region is reset for covering the printable area of the page by calling the `StartPage()` function. The clipping region set by this function is valid within the current page.

Return Value

OK when no error, or –1 when error and an error code is set into "errorno".

ResetClipPath

Name

ResetClipPath – Reset clipping region

Synopsis

```
int ResetClipPath(
        int printerContext);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

Description

Reset the current clipping region for covering the printable area of the page.

Return Value

OK when no error, or –1 when error and an error code is set into "errorno".

SetCurrentPoint

Name

SetCurrentPoint – Set current point

Synopsis

```
int SetCurrentPoint(
        int printerContext,
        Fix x,
        Fix y);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

x – x coordinate value for setting the current point.

y – y coordinate value for setting the current point.

Description

Set the current point to (x, y).

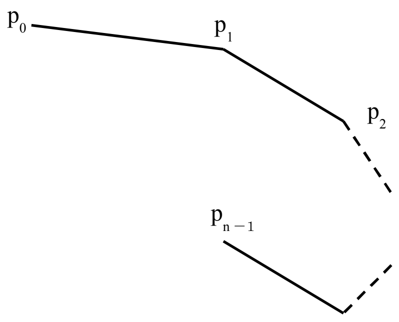
Return Value

OK when no error, or –1 when error and an error code is set into "errorno".

LinePath

Name

LinePath – add multiple connected lines to the current path



Synopsis

```
int LinePath(
        int printerContext,
        int flag;
        int npoints,
        Point *points);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

flag – PathClose: Close the sub path. In this case, the line connecting the first point and the last point in the array **points* is appended. PathOpen: Do not close the path.

npoints – Number of points in array.

points – Pointer to an array of points.

Description

Append lines specified by the array of points **points* from the current point.

In case of the *flag* is PathOpen, the current point is set to the last point in the array of points **points*. In case of the *flag* is PathClose, the current point is set to the first point in the array of points **points*.

Structures

```
typedef struct _Point {
        Fix x, y;
} Point;
```

Return Value

OK when no error, or –1 when error and an error code is set into "errorno".

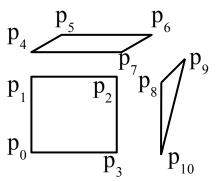
PolygonPath

Name

PolygonPath – add polygons to current path

Synopsis

```
int PolygonPath(
        int printerContext,
        int npolygons,
        int *nvertexes,
        Point *points);
```



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

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

npolygons – Number of polygon appended to the path.

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

points – Pointer to an array of points. The number of points in an array must be *nvertexes*.

Description

Append polygons specified by the array of points **points* into the current path from the current point.

The current point is set to the last point in the array of points **points*.

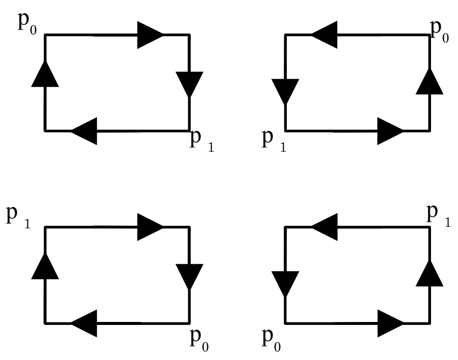
Return Value

OK when no error, or –1 when error and an error code is set into "errorno".

RectanglePath

Name

RectanglePath – Add rectangles to current path.



Synopsis

```
int RectanglePath(
        int printerContext,
        int nrectangles,
        Rectangle *rectangles);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

nrectangles – Number of rectangles.

rectangles – Pointer to an array of rectangles.

Description

Append rectangles into the current path.

The current point is set to the starting point of the last rectangle appended.

Direction of the path of each rectangle depends on how to set the starting point and diagonal point as above figure.

Path is appended in order of (x0, y0)-(x1, y0)-(x1, y1)-(x0, y1)-(x0, y0) where the starting point “p0” is (x0,y0) and the diagonal point “p1” is (x1,y1).

Structures

```
typedef struct _Rectangle {
        Point p0;    // Starting point.
        Point p1;    // Diagonal point.
} Rectangle;
```

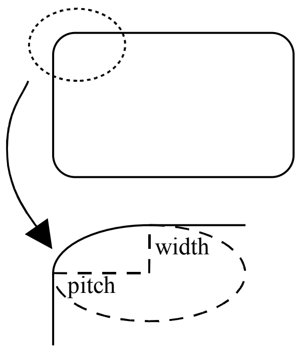
Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

RoundRectanglePath

Name

RoundRectanglePath – Add round rectangles to current path.



Synopsis

```
int RoundRectanglePath(
        int printerContext,
        int nrectangles,
        RoundRectangle *rectangles);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

nrectangles – Number of round rectangles.

rectangles – Pointer to an array of RoundRectangle structure.

Description

Append round rectangles into the path.

Each corner of round rectangle is connected by elliptic arc defined by *xellippse* and *yellipse* in the RoundRectangle structure. Lines of a round rectangle are drawn as a rectangle path.

Structures

```
typedef struct _RoundRectangle {
        Point p0;    // Starting point.
        Point p1;     // Diagonal point.
        Fix xellipse, yellipse;
} RoundRectangle;
```

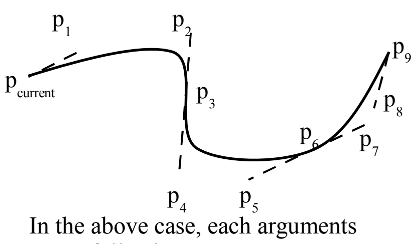
Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

BezierPath

Name

BezierPath – Append a 3D-bezier path into the current path.



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

Synopsis

```
int BezierPath(
        int printerContext,
        int npoints,
        Point *points);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

npoints –Number of points in an array **points*. It must be a multiple number of “3”.

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

Description

Append multiple bezier path with the current point, end points and the control points specified by **points*. The first bezier path is specified by the current point and third point in an array of *points as the end points and the first point and second point as the control points. Other bezier paths are specified by the previous path’s end point as the first point of bezier path, and the following points as the control points and end points as the previous bezier path.

The current point is set to the last point of the bezier path.

If *npoints* is not a multiple number of “3”, this function returns error, and the error code is set as PARAMERROR.

Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

ArcPath

Name

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

Synopsis

```
int ArchPath(
        int printerContext,
        int kind,
        int dir,
        Fix bbx0, Fix bby0, Fix bbx1, Fix bby1,
        Fix x0, Fix y0,
        Fix x1, Fix y1);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

kind – Arc: an arc、Chord: a chord、Pie: a pie.

dir – The direction of the path。Clockwise: clockwise、Counterclockwise: counter-clockwise.

(bbx0, bby0)-(bbx1, bby1) – Circumscribe rectangle.

(x0, y0) – Starting point.

(x1, y1) – End point.

Description

Append an arc, chord or a pie to the current path. The center point of ellipse is the middle point of the circumscribe rectangle. The direction of the path is specified by *dir*. When “Arc” is set into *kind* and the same point is set into both *start* and *end*, an ellipse is appended into the path. If the circumscribe rectangle is a square, a circle is appended into the path.

In case of an arc, the current point is set as the end point of an arc. In case of a chord or pie, the current point is set as the left-top point of the circumscribe rectangle.

Return Value

OK when no error, or –1 when error and an error code is set into “errorno”.

Text Operations

Text operations are used for obtain the information of device fonts,draw the device fonts,register the information of device fonts and erase the information of device fonts.

All text operation functions are OPTIONAL.

Detailed operation is not determined yet.

Bitmap Image Operations

Bitmap is the bit oriented image data which is drawn in rectangle region. When drawing the bitmap,the attribute of drawing set as Graphics State Object is applied.

The typical draw of bitmap is

- StartDrawImage() – specified the image data
- TransferDrawImage() – transfer the image data(1 time or more)
- EndDrawImage() – end of transfer image data and draw it.

If called except TransferDrawImage() function between StartDrawImage() and EndDrawImage() function ,it returns error and the error code is set as BADREQEST.

DrawImage() is a function for performing these operations by functions call once.

All bitmap operation functions are OPTIONAL.

DrawImage

Name

DrawImage – Draw bitmap image

Synopsis

```
int DrawImage(
        int printerContext,
        int sourceWidth
        int sourceHeight;
        int colorSpace;
        ImageFormat imageFormat,
        Rectangle destinationSize,
        int count,
        void *imageData);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

sourceWidth – Width of source image(pixel)

sourceHeight – Height of source image(pixel)

colorDepth – The number of bits per pixel of source image.

imageFormat– Image format enum.

destinationSize – Size of destination image.

count – The number of bytes of imageData.

imageDate – The pointer of bitmap image data.

Description

Draw the rectangle bitmap and the current point is left-upper.

If current point is not integer,the reference point is revalued to the integer and the current point is not updated.

Image format can be used which the device obtained from device capability is supporting.（iformatRaw, iformatRLE, iformatJPEG, iformatPNG or the vendor supports uniquely can be specified）

Return Value

OK when no error,or -1 when error and an error code is set into “errorno”.

StartDrawImage

Name

StartDrawImage – start draw bitmap image

Synopsis

```
int StartDrawImage(
        int printerContext,
        int sourceWidth
        int sourceHeight;
        int colorSpace;
        ImageFormat imageFormat,
        Rectangle destinationSize);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

sourceWidth – Width of source image(pixel)

sourceHeight – Height of source image（pixel）

colorDepth – The number of bits per pixel of source image.

imageFormat– Image format enum.

destinationSize – Size of destination image.

Description

Draw the rectangle bitmap and the current point is left-upper.

If current point is not integer,the reference point is revalued to the integer and the current point is not updated.

Image format can be used which the device obtained from device capability is supporting.（iformatRaw, iformatRLE, iformatJPEG, iformatPNG or the vendor supports uniquely can be specified）

Return Value

OK when no error,or -1 when error and an error code is set into “errorno”.

TransferDrawImage

Name

TransferDrawImage – transfer bitmap image data

Synopsis

```
int TransferDrawImage(
        int printerContext,
        int count,
        void *imageData);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

count – The number of bytes of imageData.

imageDate – The pointer of bitmap image data.

Description

Transfer bitmap image data which is started by StartDrawImage() function.

Return Value

OK when no error,or -1 when error and an error code is set into “errorno”.

EndDrawImage

Name

EndDrawImage – end draw bitmap image

Synopsis

```
int EndDrawImage(
        int printerContext);
```

Arguments

printerContext – Printer Context obtained by the `OpenPrinter()` function.

Description

Declare the end of drawing image data which is started by StartDrawImage() or DrawImage() functions.

Return Value

OK when no error,or -1 when error and an error code is set into “errorno”.

Scan Line Operations

Scan lines provide support for horizontal line drawing and are defined by a begin point and an end point. Scan lines are drawn according to the draw mode settings of the Graphics State Object.

All scan line operations are optional. However support for scan line operations is recommended in the case where path operations are not supported and you want to switch the color scheme from bitmap oriented to path oriented.

StartScanline

Name

`StartScanline` – initiates scan line drawing

Synopsis

```
int StartScanline(
        int printerContext,
        int yposition);
```

Arguments

`printerContext` – a printer context returned by a call to `OpenPrinter()`

`yposition` – the vertical position to start a scan line

Description

Initiates scan line drawing. Scan lines are drawn by StartScanline() to declare the begging scan lines, Scanline()'s to set actual scan line data, then EndScanline() to end the operations. Current point is not modified by StartScanline().

Return Value

Upon successful completion returns OK. Otherwise, -1 is returned and the variable `errorno` is set to indicate the error.

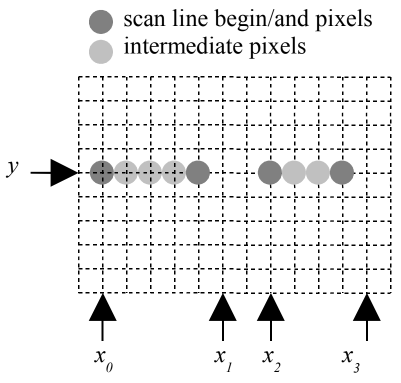
Scanline

Name

`Scanline` – draws scan lines

Synopsis

```
int Scanline(
        int printerContext,
        int nscanpairs,
        int *scanpairs);
```



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

Arguments

`printerContext` – a printer context returned by a call to `OpenPrinter()`

`nscanpairs` – number of scan line begin and end point pairs

`scanpairs` – pointer to an array of scan line begin and end point pairs

Description

Draws the scan lines passed. After the draw operation has completed, the current vertical coordinate is incremented by 1. Current point is not modified.

Return Value

Upon successful completion returns OK. Otherwise, -1 is returned and the variable `errorno` is set to indicate the error.

EndScanline

Name

`EndScanline` – terminates scan line drawing

Synopsis

```
int EndScanline(
        int printerContext);
```

Arguments

`printerContext` – a printer context returned by a call to `OpenPrinter()`

Description

Terminates scan line drawing.

Return Value

Upon successful completion returns OK. Otherwise, -1 is returned and the variable `errorno` is set to indicate the error.

Raster Image Operations

The raster image operations provide support for raster image drawing. This functionality is required for devices that support neither bitmap nor path operations.

StartRaster

Name

`StartRaster` – initiates raster image drawing

Synopsis

```
int StartRaster(
      int printerContext,
      int rasterWidth);
```

Arguments

`printerContext` – a printer context returned by a call to `OpenPrinter()`

`rasterWidth` – the width of the raster image in pixels (padding is not included)

Description

Initiates raster data drawing.

The raster image will be drawn from current point.

The color space as determined by the current Graphics State Object will be used. Possible values include: `cspaceBW`, `cspaceDeviceGray`, `cspaceDeviceCMY`、`cspaceDeviceCMYK`, `cspaceStandardRGB`, `cspaceDeviceRGB`.

The bit and/or byte order is also determined by the value set in the current Graphics State Object.

Raster data compression is not supported. The driver is expected to perform suitable compression.

The raster data format depends on the color space as tabulated below.

Color Space	# of planes	Bits per Plane	Bits per Pixel	Bytes per Pixel	Note
cspaceBW	1	1	1	1/8	bit order required padding bits are added to the rightmost byte if necessary
cspaceDeviceGray	1	8	8	1	
cspaceStandardRGB	3	8	24	3	
cspaceStandardRGB64	3	16	48	6	byte order required
cspaceDeviceCMY	3	8	24	3	
cspaceDeviceCMYK	4	8	32	4	
cmapIndexed	1	4, 8	4, 8	1/2, 1	bit order required

Return Value

Upon successful completion returns OK. Otherwise, -1 is returned and the variable `errorno` is set to indicate the error.

TransferRasterData

Name

`TransferRasterData` – transfers raster image data

Synopsis

```
int TransferRasterData(
      int printerContext,
      int count,
      unsigned char *data);
```

Arguments

`printerContext` – a printer context returned by a call to `OpenPrinter()`

`count` – number of pixel data elements

`data` – pointer to an array holding the pixel data

Description
Transfers `count` pixel data elements of raster `data`. The `data` constitutes a single row of raster data. Data exceeding the width of the raster image is silently ignored. Should less data elements be transferred than required for a single row, then the part that was not transferred shall be treated as if non-existent. The y-coordinate of current point is incremented by 1.

Return Value
Upon successful completion returns OK. Otherwise, -1 is returned and the variable `errorno` is set to indicate the error.

SkipRaster

Name
`SkipRaster` – skips lines during raster image drawing

Synopsis
```
int SkipRaster(
       int printerContext,
       int count);
```

Arguments
`printerContext` – a printer context returned by a call to `OpenPrinter()`

`count` – number of lines to be skipped

Description
Skips `count` lines in the vertical direction during raster image drawing, producing a horizontal empty space. The y-coordinate of current point is incremented by `count`.

Return Value
Upon successful completion returns OK. Otherwise, -1 is returned and the variable `errorno` is set to indicate the error.

EndRaster

Name
`EndRaster` – terminates raster image drawing

Synopsis
```
int EndRaster(
       int printerContext);
```

Arguments
`printerContext` – a printer context returned by a call to `OpenPrinter()`

Description
Terminates raster image drawing.

Return Value
Upon successful completion returns OK. Otherwise, -1 is returned and the variable `errorno` is set to indicate the error.

Stream Data Operations

The “stream data operations” are used when an application directly creates a PDL or when it sends EPS data to a PostScript device. Be careful when using these together with other drawing instructions because the result of that printing is not guaranteed.

StartStream

Name

StartStream – start of streaming data transfer

Synopsis

```
int StartStream(
        int printerContext);
```

Arguments

printerContext – Specifies a printer context obtained using OpenPrinter()

Description

StartStream designates the start of streaming data transfer. Streaming data is the data that directly goes to the printer device without being edited or checked by the driver.

Return Value

The return value is “OK” in case of normal finish. In case of error, the return value is –1 and error code is stored in “errorno”.

TransferStreamData

Name

TransferStreamData – send stream data

Synopsis

```
int TransferStreamData(
        int printerContext,
        int count,
        void *data);
```

Arguments

printerContext – Specifies a printer context obtained using OpenPrinter()

count – number of data bytes transferred

data – pointer to stream data

Description

Send streaming data between StartStream and EndStream.

Return Value

The return value is “OK” in case of normal finish. In case of error, the return value is –1 and error code is stored in “errorno”.

EndStream

Name

EndStream – end of streaming data transfer

Synopsis

```
int EndStream(
        int printerContext);
```

Arguments

printerContext – Specifies a printer context obtained using OpenPrinter()

Description

End of streaming data transfer.

Return Value

The return value is “OK” in case of normal finish. In case of error, the return value is –1 and error code is stored in “errorno”.

V.Graphic Operation Fallback

Normally, the printer driver declares the supported API through the OpenPrinter() function depending upon the available functionality of the printing device. However, in some conditions, support to an API may not be possible. A graphic operation which can not be supported by the driver or the printer device, may be implemented by changing it to a “low level operation”. Implementing using “low level operation” is called “fallback”. Though it is possible to provide high level functionality which is not supported by a printing device by implementing it in the printer driver (bringing down the level in the driver to the available functionality of the printing device and sending appropriate instructions to the printer), but a driver developer would want to avoid implementing it except in case of providing some additional functionality, for example, improvement of print quality by doing high degree of rendering in the driver.

This chapter explains how to handle the functionality not made available by the printer and the driver.

Renderer Fallback

It is a fallback due to renderer. When the pointer to driver API is NULL, or when the return code of the API call indicates “no support”, the renderer, depending on the API, calls the driver after further breaking down the processing steps in accordance with the available functionality of the low level driver.

Driver Fallback

It is a fallback due to the printer driver. Consider a situation in which a driver behaves like it supports a particular functionality towards the renderer, however the printer device does not support the requested graphics function. In this situation, the driver breaks down the request into drawing instructions that can be implemented by the printing device. The breakdown methods are; the processing by the driver itself and using external drawing library. In the external drawing library, depending on the Scan Conversion Extension (explained in the next section), vector drawing instruction can be broken down into scan line instructions. Another example of external library that can be used is libart.

Scan Conversion Extension

TBD

VI.Macros, Enums and Structures

Return Values

```
#define OK 0 // -1 for errors
```

Error Codes

```
#define FATALERROR  -1     // an error cannot be recovered is occurred in library
#define BADREQUEST  -2     // the function is called where it should not be called
#define BADCONTEXT  -3     // printer context in the parameter is invalid
#define NOTSUPPORTED       -4
                           // a combination of parameters are set
                           // which cannot be handled by driver or printer
#define JOBCANCELED -5     // the job is canceling by some cause
#define PARAMERROR  -6     // invalid parameter
```

Basic Types

```
typedef struct _Point {
      Fix x, y;
} Point;

typedef struct _Rectangle {
      Point p0;    // start point
      Point p1;    // diagonal point
} Rectangle;

typedef struct _RoundRectangle {
      Point p0;    // start point
      Point p1;    // diagonal point
      Fix xellipse, yellipse;
} RoundRectangle;
```

Image Formats

```
typedef enum _ImageFormat {
      iformatRaw = 0,
      iformatRLE = 1,
      iformatJPEG = 2,
      iformatPNG = 3
} ImageFormat;
```

Color Presentation

```
typedef enum _ColorMapping {
      cmapDirect = 0,
      cmapIndexed = 1
} ColorMapping;

typedef enum _ColorSpace {
      cspaceBW = 0,
      cspaceDeviceGray = 1,
      cspaceDeviceCMY = 2,
      cspaceDeviceCMYK = 3,
      cspaceDeviceRGB = 4,
      cspaceStandardRGB = 5,
      cspaceStandardRGB64 = 6
} ColorSpace;
```

Fill, Paint, Clip

```
typedef enum _FillMode {fillModeEvenOdd = 0, fillModeWinding = 1} FillMode;
typedef enum _PaintMode {paintModeOpaque = 0, paintModeTransparent = 1} PaintMode;
typedef enum _ClipRule {clipRuleEvenOdd = 0, clipRuleWinding = 1} ClipRule;
```

Line

```
typedef enum _LineStyle {lineStyleSolid = 0, lineStyleDash = 1} LineStyle;
typedef enum _LineCap {lineCapButt = 0, lineCapRound = 1, lineCapSquare = 2} LineCap;
typedef enum _LineJoin {lineJoinMiter = 0, lineJoinRound = 1, lineJoinBevel = 2}
```

```
LineJoin;
```

Brush

```
typedef struct _Brush {
      ColorSpace colorSpace;
      int color[4];       // aRGB quadruplet
      int xorg, yorg;     // brush origin
                          // ignored for SetBgColor
      BrushData *pbrush;  // pointer to brush data
                          // solid brush used, if null
} Brush;

typedef struct _BrushData {
      BrushDataType type;
      int width, height, pitch;
      char data[];        // “char data[1]” for GCC 2.x
} BrushData;

typedef enum _BrushDataType {bdtypeNormal = 0} BrushDataType;
```

Misc. Flags

```
#define Arc 0              // arc
#define Chord 1            // chord
#define Pie 2              // pie
#define Clockwise          // clockwise
#define Counterclockwise   // counter clockwise
#define PathClose 0 // Close path upon LinePath
#define PathOpen 1  // Do not close path upon LinePath
```

CTM

```
typedef struct _CTM {
      float a, b, c, d, e, f;
} CTM;
```

Device Information and Capabilites

```
typedef enum _pdQueryInfoFlags {
      pdQFDeviceResolution =    0x00000001,
      pdQFMediaSize =           0x00000002,
      pdQFPageRotation =        0x00000004,
      pdQFMediaNUp =            0x00000008,
      pdQFMediaDuplex =         0x00000010,
      pdQFMediaSource =         0x00000020,
      pdQFMediaDestination =    0x00000040,
      pdQFMediaType =           0x00000080,
      pdQFMediaSize =           0x00010000,  // only for QueryDeviceInfo
      pdQFPrintRegion =         0x00020000   // only for QueryDeviceInfo
} pdQueryInfoFlags;
```

Sample Header File

Here is a sample header file which can be used by driver or render based on this specification. A prefix “pd” is prepended to each symbols here.

```
/*
 * opdrv.h : OpenPrinting Vector Driver Definitions
 */

/* Return Codes */
#define pdOK 0             // error is -1
#define pdFATALERROR       // fatal error
#define pdBADREQUEST       // function is called in bad context
#define pdBADCONTEXT       // PrinterContext parameter is invalid
#define pdNOTSUPPORTED     // request cannot be handled by driver
#define pdJOBCANCELED      // job is canceled by any reason
#define pdPARAMERROR       // invalid combination of parameters

/* Basic Types */
```

```
typedef int pdFix;
typedef struct _pdPoint {
      pdFix x, y;
} pdPoint;

typedef struct _pdRectangle {
      pdPoint p0;  // start point
      pdPoint p1;  // diagonal point
} pdRectangle;

typedef struct _pdRoundRectangle {
      pdPoint p0;  // start point
      pdPoint p1;  // diagonal point
      pdFix xellipse, yellipse;
} pdRoundRectangle;

/* Image Formats */
typedef enum _pdImageFormat {
      pdiformatRaw = 0,
      pdiformatRLE = 1,
      pdiformatJPEG = 2,
      pdiformatPNG = 3
} pdImageFormat;

/* Color Presentation */
typedef enum _pdColorMapping {
      pdcmapDirect = 0,
      pdcmapIndexed = 1
} pdColorMapping;

typedef enum _pdColorSpace {
      pdcspaceBW = 0,
      pdcspaceDeviceGray = 1,
      pdcspaceDeviceCMY = 2,
      pdcspaceDeviceCMYK = 3,
      pdcspaceDeviceRGB = 4,
      pdcspaceStandardRGB = 5,
      pdcspaceStandardRGB64 = 6
} pdColorSpace;

/* Fill, Paint, Clip */
typedef enum _pdFillMode {pdfillModeEvenOdd = 0, pdfillModeWinding = 1} pdFillMode;
typedef enum _pdPaintMode {pdpaintModeOpaque = 0, pdpaintModeTransparent = 1}
pdPaintMode;
typedef enum _pdClipRule {pdclipRuleEvenOdd = 0, pdclipRuleWinding = 1} pdClipRule;

/* Line */
typedef enum _pdLineStyle {pdlineStyleSolid = 0, pdlineStyleDash = 1} pdLineStyle;
typedef enum _pdLineCap {pdlineCapButt = 0, pdlineCapRound = 1, pdlineCapSquare = 2}
pdLineCap;
typedef enum _pdLineJoin {pdlineJoinMiter = 0, pdlineJoinRound = 1, pdlineJoinBevel =
2} LineJoin;

/* Brush */
typedef enum _pdBrushDataType {pdbdtypeNormal = 0} pdBrushDataType;

typedef struct _pdBrushData {
      pdBrushDataType type;
      int width, height, pitch;
      char data[1];
} pdBrushData;

typedef struct _pdBrush {
      pdColorSpace colorSpace;
      int color[4];             // aRGB quadruplet
      pdBrushData *pbrush;      // pointer to brush data
                                // solid brush used, if null
      int xorg, yorg;           // brush origin
                                // ignored for SetBgColor
} pdBrush;

/* Misc. Flags */
```

```
#define pdArc 0            // arc
#define pdChord 1          // chord
#define pdPie 2            // pie
#define pdClockwise        // clockwise
#define pdCounterclockwise// counter clockwise
#define pdPathClose 0      // Close path upon LinePath
#define pdPathOpen 1       // Do not close path upon LinePath

/* CTM */
typedef struct _pdCTM {
      float a, b, c, d, e, f;
} pdCTM;

/* Function prototypes */
int pdOpenPrinter(
      int outputFD,
      char *printerModel,
      int *nApiEntry,
      int (**apiEntry[])());
```

VII.Authors and Contributers

Editors

Osamu Mihara – Fuji Xerox Printing Systems Co., Ltd.

Authors

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

Contributers

(alphabetical order) Hidekazu Hagiwara (MintWave), Masaki Iwata (AXE), Hidenori Kanjo (BBR), Shinpei Kitayama (Epson Kowa), Kenichi Maeda (E&D), Akio Maruyama (Ricoh), Hisao Nakamura (E&D), Koji Otani (AXE), Kenji Wakabayashi (MintWave), Toshihiro Yamagishi (Turbolinux), Akira Yoshiyama (NEC)

VIII.History

Version 0.1 (Japanese) Mihara/Toratani

Version 0.2 (Japanese) Mihara

Openprinting-0.1.1 by opfc have implemented based on this version

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

Translation to English
Some feedback from opfc implementation

Version 1.0 RC1 2005-7-20 Mihara

Change copyright notice from FDL to MIT style copyright

Delete temporary font operations.