

1  
2  
3  
4  
5  
6  
7  
8

OpenPrinting  
Vector Printer Driver  
Application Program Interface  
(PDAPI)  
Specification  
Version-1.0 RC3  
(2006-10-20)

9



**Copyright © 2006 Free Standards Group**

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

2

3 Copyright © 2006 Free Standards Group

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

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

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

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

17 Linux is a trademark of Linus Torvalds.

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

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

20 PostScript is a registered trademark of Adobe Systems Inc.

21

# Table of Contents

I.Introduction.....	6
II.Linking with Render.....	7
Loading and Calling Printer Driver.....	7
Notes about Function Parameters.....	7
III.Graphic Model.....	8
Coordinate System.....	8
Objects.....	8
CTM.....	8
Color.....	9
Alpha Composition.....	9
Scan Rule.....	9
IV.Operations.....	10
Creating and Managing Print Contexts.....	10
fsgpdOpenPrinter.....	10
fsgpdClosePrinter.....	10
Job Control Operations.....	12
fsgpdStartJob.....	12
fsgpdEndJob.....	12
fsgpdAbortJob.....	12
fsgpdStartDoc.....	13
fsgpdEndDoc.....	13
fsgpdStartPage.....	14
fsgpdEndPage.....	14
fsgpdQueryDeviceCapability.....	14
fsgpdQueryDeviceInfo.....	15
Attribute of Job, Document and Page for Job Control Operations.....	17
Graphics State Object Operations.....	19
fsgpdResetCTM.....	19
fsgpdSetCTM.....	19
fsgpdGetCTM.....	20
fsgpdInitGS.....	20
fsgpdSaveGS.....	20
fsgpdRestoreGS.....	21
fsgpdQueryColorSpace.....	21
fsgpdSetColorSpace.....	22
fsgpdGetColorSpace.....	22
fsgpdQueryROP.....	22
fsgpdSetROP.....	23
fsgpdGetROP.....	23
fsgpdSetFillMode.....	24
fsgpdGetFillMode.....	24
fsgpdSetAlphaConstant.....	24
fsgpdGetAlphaConstant.....	25
fsgpdSetLineWidth.....	25
fsgpdGetLineWidth.....	25
fsgpdSetLineDash.....	26
fsgpdGetLineDash.....	26
fsgpdSetLineDashOffset.....	27
fsgpdGetLineDashOffset.....	27

fsgpdSetLineStyle.....	28
fsgpdGetLineStyle.....	28
fsgpdSetLineCap.....	28
fsgpdGetLineCap.....	29
fsgpdSetLineJoin.....	29
fsgpdGetLineJoin.....	29
fsgpdSetMiterLimit.....	30
fsgpdGetMiterLimit.....	30
fsgpdSetMiterLimitPaintMode.....	30
fsgpdGetMiterLimitPaintMode.....	31
fsgpdSetStrokeColor.....	31
fsgpdSetFillColor.....	32
fsgpdSetBgColor.....	32
Path Operations.....	34
fsgpdNewPath.....	34
fsgpdEndPath.....	34
fsgpdStrokePath.....	34
fsgpdFillPath.....	35
fsgpdStrokeFillPath.....	35
fsgpdSetClipPath.....	35
fsgpdResetClipPath.....	36
fsgpdSetCurrentPoint.....	36
fsgpdLinePath.....	37
fsgpdPolygonPath.....	37
fsgpdRectanglePath.....	38
fsgpdRoundRectanglePath.....	38
fsgpdBezierPath.....	39
fsgpdArcPath.....	39
Text Operations.....	41
Bitmap Image Operations.....	42
fsgpdDrawImage.....	42
fsgpdStartDrawImage.....	43
fsgpdTransferDrawImage.....	43
fsgpdEndDrawImage.....	44
Scan Line Operations.....	45
fsgpdStartScanline.....	45
fsgpdScanline.....	45
fsgpdEndScanline.....	46
Raster Image Operations.....	47
fsgpdStartRaster.....	47
fsgpdTransferRasterData.....	47
fsgpdSkipRaster.....	48
fsgpdEndRaster.....	48
Stream Data Operations.....	49
fsgpdStartStream.....	49
fsgpdTransferStreamData.....	49
fsgpdEndStream.....	49
V. Graphic Operation Fallback.....	51
Renderer Fallback.....	51
Driver Fallback.....	51
VI. Macros, Types, Enums and Structures.....	52

CTM.....	54
Device Information and Capabilites.....	54
Sample Header File.....	55
Brush.....	55
Misc. Flags.....	55
Line.....	55
Error Codes.....	55
Basic Types.....	55
Image Formats.....	56
Color Presentation.....	56
Fill, Paint, Clip.....	56
VII.Authors and Contributors.....	61
Editors.....	61
Authors.....	61
Contributers.....	61
VIII.History.....	62

1

# I.Introduction

---

1

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

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

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

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

1

### 1 Loading and Calling Printer Driver

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

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

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

### 1 Notes about Function Parameters

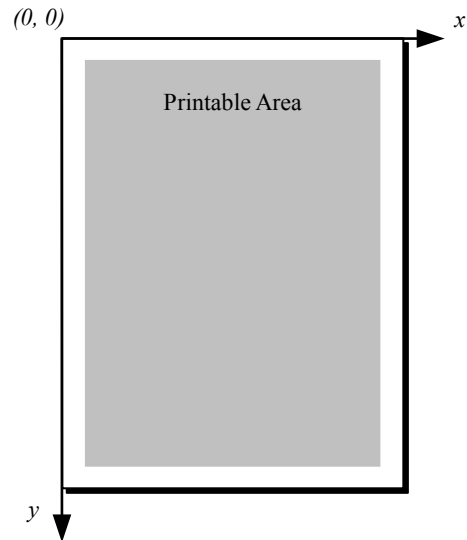
2 In case that a driver is linked as a library, data area which is designated by pointers of arguments of each functions are possibly  
3 referred until the process of the page on which drawing functions are executed. Therefore, the caller should keep the area until  
4 it calls fsgpdEndPage() operation.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

## 1 Coordinate System

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

6 The type of coordinate takes signed, fixed point 32 bits value, where integer  
 7 takes 24 bits and decimal takes 8 bits (type "FSGPD\_FIX").

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

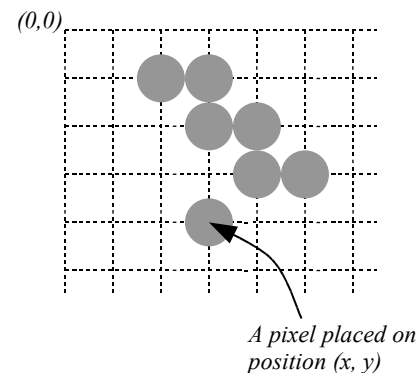


## 1 Objects

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

- 4
- 5 • Path
- 6 • Bitmap Image
- 7 • Scanline
- 8 • Raster Image
- 9 • Text

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



## 1 CTM

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

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

6  
 7 For detail of CTM, refer description in fsgpdResetCTM/fsgpdSetCTM/fsgpdGetCTM.



# 1 Color

2 Following color mapping can be used.

Color Mapping Macro	Color Mapping
FSGPD_CMAP_DIRECT	Direct Color
FSGPD_CMAP_INDEXED	Indexed Color

3 Following color space can be used.

Color Space Macro	Color Space
FSGPD_CSPACE_BW	Black and White
FSGPD_CSPACE_DEVICEGRAY	Grayscale
FSGPD_CSPACE_DEVICECMY	CMY
FSGPD_CSPACE_DEVICECMYK	CMY and Black
FSGPD_CSPACE_DEVICERGB	Device RGB
FSGPD_CSPACE_DEVICEKRGB	Device KRGB
FSGPD_CSPACE_STANDARDRGB	sRGB
FSGPD_CSPACE_STANDARDRGB64	scRGB

4

# 1 Alpha Composition

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

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

# 1 Scan Rule

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

## 1 Creating and Managing Print Contexts

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

### 3 **fsgpdOpenPrinter**

#### 4 **Name**

5 fsgpdOpenPrinter – Create printer context.

#### 6 **Synopsis**

```
7 FSGPD_DC fsgpdOpenPrinter(  
8     FSGPD_INT outputFD,  
9     FSGPD_CHAR *printerModel,  
10    FSGPD_INT apiVersion[2],  
11    FSGPD_INT *nApiEntry,  
12    FSGPD_INT (**apiEntry[])());
```

#### 13 **Arguments**

14 outputFD – file descriptor to write print data stream

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

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

18 nApiEntry – array size of apiEntry.

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

#### 20 **Description**

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

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

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

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

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

#### 31 **Return Value**

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

### 34 **fsgpdClosePrinter**

#### 35 **Name**

36 fsgpdClosePrinter – Delete printer context.

#### 37 **Synopsis**

```
38 FSGPD_RESULT fsgpdClosePrinter(  
39     FSGPD_DC printerContext);
```

#### 40 **Arguments**

41 printerContext – printer context value returned by fsgpdOpenPrinter()

1 **Description**

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

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

6 **Return Value**

7 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

# 1 Job Control Operations

2 These functions operate job control. These functions MUST be supported by all drivers. One print job consist of one or more  
3 documents. One document consist of one or more pages. If a job consist of only one document, fsgpdStartDoc() and  
4 fsgpdEndDoc() functions can be omitted. fsgpdStartJob(), fsgpdEndJob(), fsgpdStartPage() and fsgpdEndPage() functions  
5 cannot be omitted.

## 6 **fsgpdStartJob**

### 7 **Name**

8 fsgpdStartJob – Information that start print job.

### 9 **Synopsis**

```
10 FSGPD_RESULT fsgpdStartJob(  
11     FSGPD_DC printerContext,  
12     FSGPD_CHAR *jobInfo);
```

### 13 **Arguments**

14 printerContext – printer context value returned by fsgpdOpenPrinter()

15 jobInfo – Set property of print job. If NULL, default values by printer driver are set. Format and contents are described in  
16 section “Attribute of Job, Document and Page for Job Control Operations.”

### 17 **Description**

18 fsgpdStartJob() informs the start of print job.

19 Property of the print job is set by jobInfo.

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

### 22 **Return Value**

23 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 24 **fsgpdEndJob**

### 25 **Name**

26 fsgpdEndJob –Information that end print job.

### 27 **Synopsis**

```
28 FSGPD_RESULT fsgpdEndJob(  
29     FSGPD_DC printerContext);
```

### 30 **Arguments**

31 printerContext – printer context value returned by fsgpdOpenPrinter()

### 32 **Description**

33 fsgpdEndJob() informs the end of print job.

34 Job property values are reset to initial values.

### 35 **Return Value**

36 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 37 **fsgpdAbortJob**

### 38 **Name**

39 fsgpdAbortJob – Cleanup for abort print job.

### 40 **Synopsis**

```
41 FSGPD_RESULT fsgpdAbortJob(  
42     FSGPD_DC printerContext);
```

1 FSGPD\_DC printerContext);

## 2 **Arguments**

3 printerContext – printer context value returned by fsgpdOpenPrinter()

## 4 **Description**

5 fsgpdAbortJob() designates cleanup operation to abort print job. It may perform termination of print job and create print data to  
6 cleanup printer state. It is depends on the timing when this function is called whether the print job are sent to target printer or  
7 not, so calling this function does not guarantee that nothing is printed on the printer, and papers may be wasted. However, after  
8 this function is executed, the printer should be initial state and next job has to be accepted normally.

9 Job property values are reset to initial values.

## 10 **Return Value**

11 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 12 **fsgpdStartDoc**

### 13 **Name**

14 fsgpdStartDoc – Start document.

### 15 **Synopsis**

```
16 FSGPD_RESULT fsgpdStartDoc(  
17     FSGPD_DC printerContext,  
18     FSGPD_CHAR *docInfo);
```

### 19 **Arguments**

20 printerContext – printer context value returned by fsgpdOpenPrinter()

21 docInfo – Set property of print document. If NULL, default values by printer driver are set. Format and contents are described  
22 in section “Attribute of Job, Document and Page for Job Control Operations.”Description

23 fsgpdStartDoc() starts printing of a document.

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

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

### 30 **Return Value**

31 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 32 **fsgpdEndDoc**

### 33 **Name**

34 fsgpdEndDoc – End document

### 35 **Synopsis**

```
36 FSGPD_RESULT fsgpdEndDoc(  
37     FSGPD_DC printerContext);
```

### 38 **Arguments**

39 printerContext – printer context value returned by fsgpdOpenPrinter()

### 40 **Description**

41 fsgpdEndDoc() ends document printing.

### 42 **Return Value**

43 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 1 **fsgpdStartPage**

### 2 **Name**

3 fsgpdStartPage – Start print page.

### 4 **Synopsis**

```
5 FSGPD_RESULT fsgpdStartPage(  
6     FSGPD_DC printerContext,  
7     FSGPD_CHAR *pageInfo);
```

### 8 **Arguments**

9 printerContext – printer context value returned by fsgpdOpenPrinter()

10 pageInfo – Page property. If NULL, default values by printer driver are set. Format and contents are described in section  
11 “Attribute of Job, Document and Page for Job Control Operations.”

### 12 **Description**

13 fsgpdStartPage() starts printing of a page.

### 14 **Return Value**

15 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 16 **fsgpdEndPage**

### 17 **Name**

18 fsgpdEndPage – End print page.

### 19 **Synopsis**

```
20 FSGPD_RESULT fsgpdEndPage(  
21     FSGPD_DC printerContext);
```

### 22 **Arguments**

23 printerContext – printer context value returned by fsgpdOpenPrinter()

### 24 **Description**

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

### 26 **Return Value**

27 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 28 **fsgpdQueryDeviceCapability**

### 29 **Name**

30 fsgpdQueryDeviceCapability – Query device capabilities.

### 31 **Synopsis**

```
32 FSGPD_ERROR fsgpdQueryDeviceCapability(  
33     FSGPD_DC printerContext,  
34     FSGPD_FLAG queryflag,  
35     FSGPD_INT buflen,  
36     FSGPD_BYTE *infoBuf);
```

### 37 **Arguments**

38 printerContext – printer context value returned by fsgpdOpenPrinter()

39 queryflag – flags showing device capability queried.

40 buflen – buffer length which is prepared as \*infoBuf

41 infoBuf – a buffer to put device capability information

## 1 **Description**

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

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

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

```
6 typedef enum _FSGPD_QUERYINFOFLAGS {  
7     FSGPD_QF_DEVICERESOLUTION = 0x00000001,  
8     FSGPD_QF_MEDIASIZE = 0x00000002,  
9     FSGPD_QF_PAGEROTATION = 0x00000004,  
10    FSGPD_QF_MEDIANUP = 0x00000008,  
11    FSGPD_QF_MEDIADUPLEX = 0x00000010,  
12    FSGPD_QF_MEDIASOURCE = 0x00000020,  
13    FSGPD_QF_MEDIADESTINATION = 0x00000040,  
14    FSGPD_QF_MEDIATYPE = 0x00000080,  
15    FSGPD_QF_MEDIASIZE = 0x00010000, /* only for fsgpdQueryDeviceInfo */  
16    FSGPD_QF_PRINTREGION = 0x00020000 /* only for fsgpdQueryDeviceInfo */  
17 } FSGPD_QUERYINFOFLAGS;
```

18  
19 The driver returns queried information in ASCII text format. If buffer doesn't have enough length to write, the written  
20 information is truncated. In this case, the driver doesn't set error code and returns without error.

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

24 "updf:DeviceResolution=deviceResolution\_600x600,deviceResolution\_1200x1200"

## 25 **Return Value**

26 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 27 **fsgpdQueryDeviceInfo**

### 28 **Name**

29 fsgpdQueryDeviceInfo – Query device information.

### 30 **Synopsis**

```
31 FSGPD_RESULT fsgpdQueryDeviceInfo(  
32     FSGPD_DC printerContext,  
33     FSGPD_FLAG queryflag,  
34     FSGPD_INT buflen,  
35     FSGPD_CHAR *infoBuf);
```

### 36 **Arguments**

37 printerContext – printer context value returned by fsgpdOpenPrinter()

38 queryflag – flags showing device capability queried.

39 buflen – buffer length which is prepared as \*infoBuf

40 infoBuf – a buffer to put device capability information

### 41 **Description**

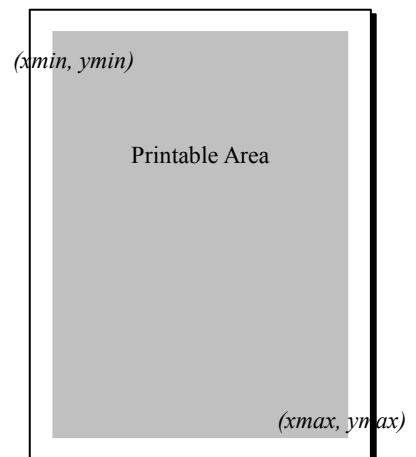
42 fsgpdQueryDeviceInfo() is used to query current setting of printer device or driver.

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

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

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

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



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

3 PrintRegion=xmin,ymin,xmax,ymax

4 **Return Value**

5 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.



# 1 Attribute of Job, Document and Page for Job Control Operations

2 Properties for job, document and page are set as parameter of fsgpdStartJob(), fsgpdStartDoc(), fsgpdStartPage(). Supported  
 3 parameters and their values are provided by printer driver database file distributed with drivers.

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

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

- 7  
 8
- `<scheme>`: The specification which defines name space.
  - `<key>`: Name of the property.
  - `<value>`: Value of the property.
- 9  
 10

11  
 12 Several `<value>` can be designated separating by "," (comma). If several values are designated, the printer driver searches it  
 13 from begging of the string and takes possible value.

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

15  
 16  
 17 Conforming implementations of this PDAPI specification MUST support the "updf" value for `<scheme>` as defined in IEEE-  
 18 ISTO PWG 5101.4 "Universal Printer Definition Format" (May 2004) developed by the Printer Working Group.

19 Conforming implementations of this PDAPI specification MUST ignore unknown properties, for compatibility with future  
 20 vendor or standard extensions.

21 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 ...	✓	✓	✓
Media Copy	MediaCopy	1, 2, ...			

Property	Name	Value	Effective in		
			Job	Doc	Page
Print Quality	PrintQuality	draft high normal	✓	✓	✓

22

23

24

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.

# 1 Graphics State Object Operations

2 These functions operate graphics state object. All of these functions are OPTIONAL.

## 3 **fsgpdResetCTM**

### 4 **Name**

5 fsgpdResetCTM – Initialize CTM of the Graphics State Object.

### 6 **Synopsis**

```
7 FSGPD_RESULT fsgpdResetCTM(  
8     FSGPD_DC printerContext);
```

### 9 **Arguments**

10 printerContext – printer context value returned by fsgpdOpenPrinter()

### 11 **Description**

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

$$13 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

### 14 **Return Value**

15 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 16 **fsgpdSetCTM**

### 17 **Name**

18 fsgpdSetCTM – Set CTM to the Graphics State Object.

### 19 **Synopsis**

```
20 FSGPD_RESULT fsgpdSetCTM(  
21     FSGPD_DC printerContext,  
22     FSGPD_CTM *pCTM);
```

### 23 **Arguments**

24 printerContext – printer context value returned by fsgpdOpenPrinter()

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

### 26 **Description**

27 fsgpdSetCTM() sets a CTM to the Graphics State Object.

28 Device coordinate system value [xdev, ydev] is represented by CTM and render coordinate system value [xren, yren] in  
29 following equation.

$$30 \begin{bmatrix} x_{dev} & y_{dev} & 1 \end{bmatrix} = \begin{bmatrix} x_{ren} & y_{ren} & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{bmatrix}$$

### 31 **Structures**

```
32 typedef struct _FSGPD_CTM {  
33     FSGPD_FLOAT a, b, c, d, e, f;  
34 } FSGPD_CTM;
```

1 **Return Value**

2 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

3 **fsgpdGetCTM**

4 **Name**

5 fsgpdGetCTM – Get CTM from the Graphics State Object.

6 **Synopsis**

```
7 FSGPD_RESULT fsgpdGetCTM(  
8     FSGPD_DC printerContext;  
9     FSGPD_CTM *pCTM);
```

10 **Arguments**

11 printerContext – printer context value returned by fsgpdOpenPrinter()

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

13 **Description**

14 fsgpdGetCTM() retrieves a CTM from Graphics State Object.

15 **Structures**

```
16 typedef struct _FSGPD_CTM {  
17     FSGPD_FLOAT a, b, c, d, e, f;  
18 } FSGPD_CTM;
```

19 **Return Value**

20 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

21 **fsgpdInitGS**

22 **Name**

23 fsgpdInitGS – Initialize the Graphics State parameters.

24 **Synopsis**

```
25 FSGPD_RESULT fsgpdInitGS(  
26     FSGPD_DC printerContext);
```

27 **Arguments**

28 printerContext – printer context value returned by fsgpdOpenPrinter()

29 **Description**

30 fsgpdInitGS() 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 fsgpdSaveGS() and fsgpdRestoreGS() operations described in this document. These operations are used when changing graphics state temporary and restores later.

35 Graphics state object keeps its setting values between fsgpdStartJob() and fsgpdEndJob(), unless fsgpdInitGS() is called. At fsgpdStartJob(), same values as fsgpdInitGS() is called are set.

37 **Return Value**

38 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

39 **fsgpdSaveGS**

40 **Name**

41 fsgpdSaveGS – Save the Graphics State parameters.

1 **Synopsis**  
2 FSGPD\_RESULT fsgpdSaveGS(  
3 FSGPD\_DC printerContext);

4 **Arguments**  
5 printerContext – printer context value returned by fsgpdOpenPrinter()

6 **Description**  
7 fsgpdSaveGS() pushes graphics state object parameters onto the driver managed stack.  
8 The size of the stack is limited, but at least n times of this operation can be called. If more than n times this function is called  
9 without calling fsgpdRestoreGS(), error is returned.  
10 After n times fsgpdSaveGS() is called, same times fsgpdRestoreGS() can be executed. More than n time fsgpdRestoreGS() is  
11 called, it results in error (BADREQUEST).

12 **Return Value**  
13 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 14 fsgpdRestoreGS

15 **Name**  
16 fsgpdRestoreGS – Restore the Graphics State parameters.

17 **Synopsis**  
18 FSGPD\_RESULT fsgpdRestoreGS(  
19 FSGPD\_DC printerContext);

20 **Arguments**  
21 printerContext – printer context value returned by fsgpdOpenPrinter()

22 **Description**  
23 Saved graphics state object is restored to current value.  
24 After n times fsgpdSaveGS() is called, same times fsgpdRestoreGS() can be executed. More than n time fsgpdRestoreGS() is  
25 called, it results in error (BADREQUEST).

26 **Return Value**  
27 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 28 fsgpdQueryColorSpace

29 **Name**  
30 fsgpdQueryColorSpace – Obtain the color space which can be used by the driver.

31 **Synopsis**  
32 FSGPD\_RESULT fsgpdQueryColorSpace(  
33 FSGPD\_DC printerContext,  
34 FSGPD\_CSPACE \*pcspace  
35 FSGPD\_INT \*pnum);

36 **Arguments**  
37 printerContext – printer context value returned by fsgpdOpenPrinter()  
38 pcspace – Pointer to the Color Space enum. array.  
39 pnum – Pointer to the number of the Color Space enum. array elements.

40 **Description**  
41 Retrieves list of color space which can be treated by the driver from graphics state object.  
42 The caller should allocate at least n as the number of elements of an array pcspace[] which stores color space. Also the caller  
43 should call this function by setting its maximum number of elements in \*pnum. The number of elements retrieved is set to

1 \*pnum. If the number of elements exceeds the number of elements designated, this function returns necessary number in  
2 \*pnum.  
3 The driver returns the color space into array in preferable order.

#### 4 **Return Value**

5 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 6 **fsgpdSetColorSpace**

#### 7 **Name**

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

#### 9 **Synopsis**

```
10 FSGPD_RESULT fsgpdSetColorSpace(  
11     FSGPD_DC printerContext,  
12     FSGPD_CSPACE cspace);
```

#### 13 **Arguments**

14 printerContext – printer context value returned by fsgpdOpenPrinter()

15 cspace –Color Space enum.

#### 16 **Description**

17 fsgpdSetColorSpace() sets color space handled by the driver into graphics state object.

#### 18 **Return Value**

19 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 20 **fsgpdGetColorSpace**

#### 21 **Name**

22 fsgpdGetColorSpace – Get the current color space from the Graphics State Object.

#### 23 **Synopsis**

```
24 FSGPD_RESULT fsgpdGetColorSpace(  
25     FSGPD_DC printerContext,  
26     FSGPD_CSPACE *pcspace);
```

#### 27 **Arguments**

28 printerContext – printer context value returned by fsgpdOpenPrinter()

29 pcspace – Pointer to the Color Space enum.

#### 30 **Description**

31 fsgpdGetColorSpace() retrieves color space treated by the driver from graphics state object.

#### 32 **Return Value**

33 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 34 **fsgpdQueryROP**

#### 35 **Name**

36 fsgpdQueryROP – Obtain the ROP which can be used by the driver.

#### 37 **Synopsis**

```
38 FSGPD_RESULT fsgpdQueryROP(  
39     FSGPD_DC printerContext,  
40     FSGPD_INT *pnum,  
41     FSGPD_ROP *prop);
```

1 **Arguments**  
2 printerContext – printer context value returned by fsgpdOpenPrinter()  
3 pnum – Pointer to the number of the ROP mode array elements.  
4 prop –Pointer to the ROP mode. array.

5 **Description**  
6 fsgpdQueryROP retrieves a list of ROP (Raster Operations) which can be treated by the driver from graphics state object.  
7 If NULL is set in \*prop, the number of ROP supported is returned in \*pnum.  
8 So, at least the number of ROP supported by the operation above should be allocated for the array prop[].  
9 The number of elements is returned in \*pnum. If the number of ROP exceeds the size of array designated, it returns an error  
10 (FSGPD\_PARAMERROR) and necessary number of elements is returned in \*pnum.

11 **Return Value**  
12 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 13 **fsgpdSetROP**

14 **Name**  
15 fsgpdSetROP – Set the ROP mode to the Graphics State Object.

16 **Synopsis**  
17 FSGPD\_RESULT fsgpdSetROP(  
18 FSGPD\_DC printerContext,  
19 FSGPD\_ROP rop);

20 **Arguments**  
21 printerContext – printer context value returned by fsgpdOpenPrinter()  
22 rop –ROP mode (ROP3 code)

23 **Description**  
24 fsgpdSetROP() sets raster operation code designated by rop to graphics state object.

25 **Return Value**  
26 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 27 **fsgpdGetROP**

28 **Name**  
29  
30 fsgpdGetROP – Get the ROP mode from the Graphics State Object.

31 **Synopsis**  
32 FSGPD\_RESULT fsgpdGetROP(  
33 FSGPD\_DC printerContext,  
34 FSGPD\_ROP \*prop);

35 **Arguments**  
36 printerContext – printer context value returned by fsgpdOpenPrinter()  
37 prop – Pointer to the ROP mode.

38 **Description**  
39 fsgpdGetROP() retrieves raster operation code currently set in graphics state.

40 **Return Value**  
41 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 1 **fsgpdSetFillMode**

### 2 **Name**

3 fsgpdSetFillMode – Set the filling mode.

### 4 **Synopsis**

```
5 FSGPD_RESULT fsgpdSetFillMode(  
6     FSGPD_DC printerContext,  
7     FSGPD_FILLMODE fillmode);
```

### 8 **Arguments**

9 printerContext – printer context value returned by fsgpdOpenPrinter()

10 fillmode – Filling mode enum. Either of FSGPD\_FILLMODE\_EVENODD, FSGPD\_FILLMODE\_WINDING can be  
11 designated.

### 12 **Description**

13 fsgpdSetFillMode() sets a fill mode to graphics state object.

### 14 **Return Value**

15 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 16 **fsgpdGetFillMode**

### 17 **Name**

18 fsgpdGetFillMode – Get the filling mode.

### 19 **Synopsis**

```
20 FSGPD_RESULT fsgpdGetFillMode(  
21     FSGPD_DC printerContext,  
22     FSGPD_FILLMODE *pfillmode);
```

### 23 **Arguments**

24 printerContext – printer context value returned by fsgpdOpenPrinter()

25 pfillmode – Pointer to the filling mode enum.

### 26 **Description**

27 fsgpdGetFillMode retrieves the fill mode currently set in graphics state object.

### 28 **Return Value**

29 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 30 **fsgpdSetAlphaConstant**

### 31 **Name**

32 fsgpdSetAlphaConstant – Set the alpha blending constant.

### 33 **Synopsis**

```
34 FSGPD_RESULT fsgpdSetAlphaConstant(  
35     FSGPD_DC printerContext,  
36     FSGPD_FLOAT alpha);
```

### 37 **Arguments**

38 printerContext – printer context value returned by fsgpdOpenPrinter()

39 alpha – Alpha blending constant. Between 0.0 and 1.0

### 40 **Description**

41 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  
42 extends this range, it will be truncated to 0.0 or 1.0.



1 **Return Value**  
2 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

### 3 **fsgpdGetAlphaConstant**

4 **Name**  
5 fsgpdGetAlphaConstant – Get the alpha blending constant.

6 **Synopsis**  
7 FSGPD\_RESULT fsgpdGetAlphaConstant(  
8 FSGPD\_DC printerContext,  
9 FSGPD\_FLOAT \*palpha);

10 **Arguments**  
11 printerContext – printer context value returned by fsgpdOpenPrinter()  
12 palpha – Pointer to the alpha constant value.

13 **Description**  
14 fsgpdGetAlphaConstant() retrieves alpha constant setting from graphics state object.

15 **Return Value**  
16 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

### 17 **fsgpdSetLineWidth**

18 **Name**  
19 fsgpdSetLineWidth – Set the line width.

20 **Synopsis**  
21 FSGPD\_RESULT fsgpdSetLineWidth(  
22 FSGPD\_DC printerContext,  
23 FSGPD\_FIX width);

24 **Arguments**  
25 printerContext – printer context value returned by fsgpdOpenPrinter()  
26 width – Line width value.

27 **Description**  
28 fsgpdSetLineWidth() sets line width for stroke operations to graphics state object. The line width should be set by units in  
29 device coordinate system.  
30 The treatment of a line width less than one depends on the device or driver implementation.  
31 The maximum line width depends on device.

32 **Return Value**  
33 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

### 34 **fsgpdGetLineWidth**

35 **Name**  
36 fsgpdGetLineWidth – Get the line width.

37 **Synopsis**  
38 FSGPD\_RESULT fsgpdGetLineWidth(  
39 FSGPD\_DC printerContext,  
40 FSGPD\_FIX \*pwidth);

1 **Arguments**

- 2 printerContext – printer context value returned by fsgpdOpenPrinter()
- 3 width – Pointer to the Line width value.

4 **Description**

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

7 **Return Value**

8 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

9 **fsgpdSetLineDash**

10 **Name**

11 fsgpdSetLineDash – Set the line dash pattern.

12 **Synopsis**

```
13 FSGPD_RESULT fsgpdSetLineDash(  
14     FSGPD_DC printerContext,  
15     FSGPD_FIX pdash[],  
16     FSGPD_INT num);
```

17 **Arguments**

- 18 printerContext – printer context value returned by fsgpdOpenPrinter()
- 19 pdash – Pointer to the line dash pattern array.
- 20 num – Number of the line dash pattern array elements.

21 **Description**

22 fsgpdSetLineDash() sets a dash pattern to graphics state object.

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

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

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

28 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  
29 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  
30 background color in case of paintOpaque mode and for not-painted length in case of paintTransparent mode.

31 The maximum number of elements depend on the device.

32 If num is zero, solid lines are drawn.

33 **Return Value**

34 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

35 **fsgpdGetLineDash**

36 **Name**

37 fsgpdGetLineDash – Get the line dash pattern.

38 **Synopsis**

```
39 FSGPD_RESULT fsgpdGetLineDash(  
40     FSGPD_DC printerContext,  
41     FSGPD_FIX pdash[],  
42     FSGPD_INT *pnum);
```

43 **Arguments**

- 44 printerContext – printer context value returned by fsgpdOpenPrinter()

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

### 3 **Description**

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

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

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

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

### 11 **Return Value**

12 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 13 **fsgpdSetLineDashOffset**

### 14 **Name**

15 fsgpdSetLineDashOffset – Set the line dash pattern offset.

### 16 **Synopsis**

```
17 FSGPD_RESULT fsgpdSetLineDashOffset (  
18     FSGPD_DC printerContext,  
19     FSGPD_FIX offset);
```

### 20 **Arguments**

21 printerContext – printer context value returned by fsgpdOpenPrinter()

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

### 23 **Description**

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

### 25 **Return Value**

26 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 27 **fsgpdGetLineDashOffset**

### 28 **Name**

29 fsgpdGetLineDashOffset – Get the line dash pattern offset.

### 30 **Synopsis**

```
31 FSGPD_RESULT fsgpdGetLineDashOffset (  
32     FSGPD_DC printerContext,  
33     FSGPD_FIX *poffset);
```

### 34 **Arguments**

35 printerContext – printer context value returned by fsgpdOpenPrinter()

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

### 37 **Description**

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

### 39 **Return Value**

40 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 1 **fsgpdSetLineStyle**

### 2 **Name**

3 fsgpdSetLineStyle – Set the line style.

### 4 **Synopsis**

```
5 FSGPD_RESULT fsgpdSetLineStyle(  
6     FSGPD_DC printerContext,  
7     FSGPD_LINESTYLE linestyle);
```

### 8 **Arguments**

9 printerContext – printer context value returned by fsgpdOpenPrinter()

10 linestyle – Line style enum. FSPGD\_LINESTYLE\_SOLID or FSGPD\_LINESTYLE\_DASH can be designated.

### 11 **Description**

12 fsgpdSetLineStyle() sets line style to graphics state object.

### 13 **Return Value**

14 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 15 **fsgpdGetLineStyle**

### 16 **Name**

17 fsgpdGetLineStyle – Get the line style.

### 18 **Synopsis**

```
19 FSGPD_RESULT fsgpdGetLineStyle(  
20     FSGPD_DC printerContext,  
21     FSGPD_LINESTYLE *plinestyle);
```

### 22 **Arguments**

23 printerContext – printer context value returned by fsgpdOpenPrinter()

24 plinestyle – Pointer to the line style enum.

### 25 **Description**

26 fsgpdGetLineStyle() retrieves line style for stroke from graphics state object.

### 27 **Return Value**

28 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 29 **fsgpdSetLineCap**

### 30 **Name**

31 fsgpdSetLineCap – Set the line cap style.

### 32 **Synopsis**

```
33 FSGPD_RESULT fsgpdSetLineCap(  
34     FSGPD_DC printerContext,  
35     FSGPD_LINECAP linecap);
```

### 36 **Arguments**

37 printerContext – printer context value returned by fsgpdOpenPrinter()

38 linecap – Line cap style enum. Either of FSGPD\_LINECAP\_BUTT, FSGPD\_LINECAP\_ROUND,  
39 FSGPD\_LINECAP\_SQUARE can be set.

### 40 **Description**

41 fsgpdSetLineCap() sets the style for line cap to graphics state object.

1 **Return Value**

2 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

3 **fsgpdGetLineCap**

4 **Name**

5 fsgpdGetLineCap – Get the line cap style.

6 **Synopsis**

```
7 FSGPD_RESULT fsgpdGetLineCap(  
8     FSGPD_DC printerContext,  
9     FSGPD_LINECAP *plinecap);
```

10 **Arguments**

11 printerContext – printer context value returned by fsgpdOpenPrinter()

12 plinecap – Pointer to the line cap style enum.

13 **Description**

14 fsgpdGetLineCap() retrieves the style for line cap from graphics state object.

15 **Return Value**

16 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

17 **fsgpdSetLineJoin**

18 **Name**

19 fsgpdSetLineJoin – Set the line join style.

20 **Synopsis**

```
21 FSGPD_RESULT fsgpdSetLineJoin(  
22     FSGPD_DC printerContext,  
23     FSGPD_LINEJOIN linejoin);
```

24 **Arguments**

25 printerContext – printer context value returned by fsgpdOpenPrinter()

26 linejoin – Line join style enum. Either of FSGPD\_LINEJOIN\_MITER, FSGPD\_LINEJOIN\_ROUND and  
27 FSGPD\_LINEJOIN\_BEVEL can be set.

28 **Description**

29 fsgpdSetLineJoin() sets the line joining method to graphics state object.

30 **Return Value**

31 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

32 **fsgpdGetLineJoin**

33 **Name**

34 fsgpdGetLineJoin – Get the line join style.

35 **Synopsis**

```
36 FSGPD_RESULT fsgpdGetLineJoin(  
37     FSGPD_DC printerContext,  
38     FSGPD_LINEJOIN *plinejoin);
```

39 **Arguments**

40 printerContext – printer context value returned by fsgpdOpenPrinter()

41 plinejoin – Pointer to the line join style enum.

1 **Description**  
2 fsgpdGetLineJoin() retrieves the line joining method from graphics state object.

3 **Return Value**  
4 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

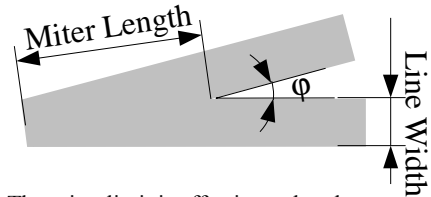
## 5 fsgpdSetMiterLimit

6 **Name**  
7 fsgpdSetMiterLimit – Set the miter limit value.

8 **Synopsis**  
9 FSGPD\_RESULT fsgpdSetMiterLimit(  
10 FSGPD\_DC printerContext,  
11 FSGPD\_FIX miterlimit);

$$\text{Miter Limit} = \frac{\text{Miter Length}}{\text{Line Width}} = \frac{1}{\sin\left(\frac{\phi}{2}\right)}$$

12 **Arguments**  
13 printerContext – printer context value returned by fsgpdOpenPrinter()  
14 miterlimit – Maximum miter length.



15 **Description**  
16 fsgpdSetMiterLimit() sets the maximum length of miter to graphics state object. The miter limit is effective only when  
17 line joining style is FSGPD\_LINEJOIN\_MITER. As to the definition of the miter limit, refer the figure right. The unit is in  
18 device coordinate system.

19 **Return Value**  
20 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 21 fsgpdGetMiterLimit

22 **Name**  
23 fsgpdGetMiterLimit – Get the miter limit value.

24 **Synopsis**  
25 FSGPD\_RESULT fsgpdGetMiterLimit(  
26 FSGPD\_DC printerContext,  
27 FSGPD\_FIX \*pmiterlimit);

28 **Arguments**  
29 printerContext – printer context value returned by fsgpdOpenPrinter()  
30 pmiterlimit – Pointer to the maximum miter length.

31 **Description**  
32 fsgpdGetMiterLimit() retrieves the maximum length of miter from graphics state object.

33 **Return Value**  
34 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 35 fsgpdSetPaintMode

36 **Name**  
37 fsgpdSetPaintMode – Set the background color painting mode.

38 **Synopsis**  
39 FSGPD\_RESULT fsgpdSetPaintMode(  
40 FSGPD\_DC printerContext,  
41 FSGPD\_PAINTMODE paintmode);

## 1 **Arguments**

2 printerContext – printer context value returned by fsgpdOpenPrinter()  
3 paintmode – Painting mode enum. Either of FSGPD\_PAINTMODE\_OPAQUE and FSGPD\_PAINTMODE\_TRANSPARENT  
4 can be set.

## 5 **Description**

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

## 7 **Return Value**

8 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 9 **fsgpdGetPaintMode**

### 10 **Name**

11 fsgpdGetPaintMode – Get the background color painting mode.

### 12 **Synopsis**

```
13 FSGPD_RESULT fsgpdGetPaintMode(  
14     FSGPD_DC printerContext,  
15     FSGPD_PAINTMODE ppaintmode);
```

### 16 **Arguments**

17 printerContext – printer context value returned by fsgpdOpenPrinter()

18 ppaintmode – Pointer to the painting mode enum.

### 19 **Description**

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

### 21 **Return Value**

22 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 23 **fsgpdSetStrokeColor**

### 24 **Name**

25 fsgpdSetStrokeColor – Define stroke color and pattern for fsgpdStrokePath() and fsgpdStrokeFillPath() operations.

### 26 **Synopsis**

```
27 FSGPD_RESULT fsgpdSetStrokeColor(  
28     FSGPD_DC printerContext,  
29     FSGPD_BRUSH *brush);
```

### 30 **Arguments**

31 printerContext – printer context value returned by fsgpdOpenPrinter()

32 brush – Pointer to brush definition.

### 33 **Description**

34 fsgpdSetStrokeColor() registers the color or pattern for stroke drawing of fsgpdStrokePath() and fsgpdStrokeFillPath()  
35 operations as a brush to graphics state object.

36 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  
37 color[4] in the color space by colorSpace.

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

### 41 **Structures**

```
42 typedef struct _FSGPD_BRUSH {  
43     FSGPD_CSPACE colorSpace;  
44     FSGPD_INT color[4]; /* aRGB quadruplet */
```

```

1         FSGPD_INT xorg, yorg;      /* brush origin */
2         FSGPD_BRUSHDATA *pbrush;  /* ignored for fsgpdSetBgColor */
3         FSGPD_BRUSHDATA *pbrush; /* pointer to brush data */
4         /* solid brush used, if NULL */
5     } FSGPD_BRUSH;
6
7     typedef struct _FSGPD_BRUSHDATA {
8         FSGPD_BDTYPE type;
9         FSGPD_INT width, height, pitch;
10        FSGPD_BYTE data[];        /* FSGPD_BYTE data[1] for GCC 2.x */
11    } FSGPD_BRUSHDATA;
12
13    typedef enum _FSGPD_BDTYPE {FSGPD_BDTYPE_NORMAL = 0} FSGPD_BDTYPE;

```

#### 14 **Return Value**

15 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 16 **fsgpdSetFillColor**

#### 17 **Name**

18 fsgpdSetFillColor – Define fill color and pattern for fsgpdFillPath() and fsgpdStrokeFillPath() operations.

#### 19 **Synopsis**

```

20 FSGPD_RESULT fsgpdSetFillColor(
21     FSGPD_DC printerContext,
22     FSGPD_BRUSH *brush);

```

#### 23 **Arguments**

24 printerContext – printer context value returned by fsgpdOpenPrinter()

25 brush – Pointer to brush definition.

#### 26 **Description**

27 fsgpdSetFillColor() registers the color or pattern for fsgpdFillPath() and fsgpdStrokeFillPath() operations as a brush to graphics state object.

29 If successful, this function return FSGPD\_OK. If an error is occurred

#### 30 **Return Value**

31 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

## 32 **fsgpdSetBgColor**

#### 33 **Name**

34 fsgpdSetBgColor – Define background color and pattern for fsgpdStrokePath(), fsgpdFillPath() and fsgpdStrokeFillPath() operations.

#### 36 **Synopsis**

```

37 FSGPD_RESULT fsgpdSetBgColor(
38     FSGPD_DC printerContext,
39     FSGPD_BRUSH *brush);

```

#### 40 **Arguments**

41 printerContext – printer context value returned by fsgpdOpenPrinter()

42 brush – Pointer to brush definition.

#### 43 **Description**

44 fsgpdSetBgColor() registers the background or pattern for fsgpdStrokePath(), fsgpdFillPath() and fsgpdStrokeFillPath() operations as brush.

46 The Members colorSpace and color[4] of the FSGPD\_BRUSH structure are referred as brush data.

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



1 **Return Value**

2 If successful, this function return FSGPD\_OK. If an error is occurred, it returns -1 and stores error code in fsgpdErrorNo.

# 1 Path Operations

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

5 All path operation functions are OPTIONAL.

## 6 **fsgpdNewPath**

### 7 **Name**

8 fsgpdNewPath – start a new path

### 9 **Synopsis**

```
10 FSGPD_RESULT fsgpdNewPath(  
11     FSGPD_DC printerContext);
```

### 12 **Arguments**

13 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

### 14 **Description**

15 Delete the current path and start a new path.

### 16 **Return Value**

17 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.

## 18 **fsgpdEndPath**

### 19 **Name**

20 fsgpdEndPath – end current path

### 21 **Synopsis**

```
22 FSGPD_RESULT fsgpdEndPath(  
23     FSGPD_DC printerContext);
```

### 24 **Arguments**

25 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

### 26 **Description**

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

### 28 **Return Value**

29 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.

## 30 **fsgpdStrokePath**

### 31 **Name**

32 fsgpdStrokePath – stroke current path

### 33 **Synopsis**

```
34 FSGPD_RESULT fsgpdStrokePath(  
35     FSGPD_DC printerContext);
```

### 36 **Arguments**

37 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

### 38 **Description**

39 Draw lines along the current path. The lines are drawn along the drawing attributes registered in the Graphics State Object when

1 calling this function. The Graphics State Object retains the current path.

2 **Return Value**

3 FSGPD\_OK when no error, or -1 when error and an error code is set into "fsgpdErrorNo".

4 **fsgpdFillPath**

5 **Name**

6 fsgpdFillPath – fill current path

7 **Synopsis**

```
8 FSGPD_RESULT fsgpdFillPath(  
9     FSGPD_DC printerContext);
```

10 **Arguments**

11 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

12 **Description**

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

16 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)  
17 and closed.

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

19 **Return Value**

20 FSGPD\_OK when no error, or -1 when error and an error code is set into "fsgpdErrorNo".

21 **fsgpdStrokeFillPath**

22 **Name**

23 fsgpdStrokeFillPath – stroke and fill current path

24 **Synopsis**

```
25 FSGPD_RESULT fsgpdStrokeFillPath(  
26     FSGPD_DC printerContext);
```

27 **Arguments**

28 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

29 **Description**

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

31 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)  
32 and closed.

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

34 **Return Value**

35 FSGPD\_OK when no error, or -1 when error and an error code is set into "fsgpdErrorNo".

36 **fsgpdSetClipPath**

37 **Name**

38 fsgpdSetClipPath – Set current path as clipping region

39 **Synopsis**

```
40 FSGPD_RESULT fsgpdSetClipPath(  
41     FSGPD_DC printerContext,
```

1 FSGPD\_CLIPRRULE clipRule);

## 2 **Arguments**

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

4 clipRule – Clipping rule enum. `FSGPD_CLIPRULE_EVENODD` or `FSGPD_CLIPRULE_WINDING` can be set.

## 5 **Description**

6 Set the current path as the clipping region.

7 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)  
8 and closed.

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

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

## 12 **Return Value**

13 `FSGPD_OK` when no error, or `-1` when error and an error code is set into “`fsgpdErrorNo`”.

## 14 **fsgpdResetClipPath**

### 15 **Name**

16 `fsgpdResetClipPath` – Reset clipping region

### 17 **Synopsis**

```
18 FSGPD_RESULT fsgpdResetClipPath(  
19     FSGPD_DC printerContext);
```

### 20 **Arguments**

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

### 22 **Description**

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

### 24 **Return Value**

25 `FSGPD_OK` when no error, or `-1` when error and an error code is set into “`fsgpdErrorNo`”.

## 26 **fsgpdSetCurrentPoint**

### 27 **Name**

28 `fsgpdSetCurrentPoint` – Set current point

### 29 **Synopsis**

```
30 FSGPD_RESULT fsgpdSetCurrentPoint(  
31     FSGPD_DC printerContext,  
32     FSGPD_FIX x,  
33     FSGPD_FIX y);
```

### 34 **Arguments**

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

36 x – x coordinate value for setting the current point.

37 y – y coordinate value for setting the current point.

### 38 **Description**

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

### 40 **Return Value**

41 `FSGPD_OK` when no error, or `-1` when error and an error code is set into “`fsgpdErrorNo`”.

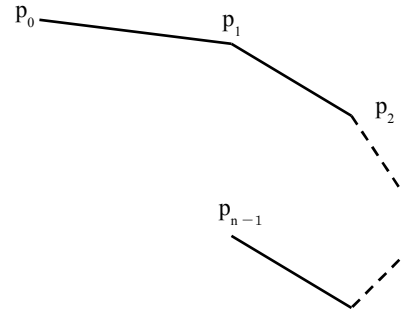
## 1 fsgpdLinePath

### 2 Name

3 fsgpdLinePath – add multiple connected lines to the current path

### 4 Synopsis

```
5 FSGPD_RESULT fsgpdLinePath(  
6     FSGPD_DC printerContext,  
7     FSGPD_PATHMODE flag;  
8     FSGPD_INT npoints,  
9     FSGPD_POINT *points);
```



### 10 Arguments

11 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

12 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.

14 npoints – Number of points in array.

15 points – Pointer to an array of points.

### 16 Description

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

18 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.

### 20 Structures

```
21 typedef struct _FSGPD_POINT {  
22     FSGPD_FIX x, y;  
23 } FSGPD_POINT;
```

### 24 Return Value

25 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.

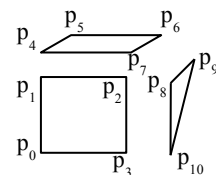
## 26 fsgpdPolygonPath

### 27 Name

28 fsgpdPolygonPath – add polygons to current path

### 29 Synopsis

```
30 FSGPD_RESULT fsgpdPolygonPath(  
31     FSGPD_DC printerContext,  
32     FSGPD_INT npolygons,  
33     FSGPD_INT *nvertexas,  
34     FSGPD_POINT *points);
```



### 35 Arguments

36 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

37 npolygons – Number of polygon appended to the path.

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

39 points – Pointer to an array of points. The number of points in an array must be nvertexas.

In the above case, each argument is as following:

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

### 40 Description

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

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

### 43 Return Value

44 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.

# 1 fsgpdRectanglePath

2 **Name**

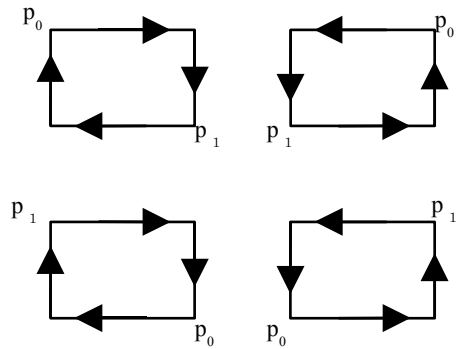
3 fsgpdRectanglePath – Add rectangles to current path.

4 **Synopsis**

```
5 FSGPD_RESULT fsgpdRectanglePath(
6     FSGPD_DC printerContext,
7     FSGPD_INT nrectangles,
8     FSGPD_RECTANGLE *rectangles);
```

9 **Arguments**

10 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.  
 11  
 12 nrectangles – Number of rectangles.  
 13 rectangles – Pointer to an array of rectangles.



14 **Description**

15 Append rectangles into the current path.  
 16 The current point is set to the starting point of the last rectangle appended.  
 17 Direction of the path of each rectangle depends on how to set the starting point and diagonal point as above figure.  
 18 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).  
 19

20 **Structures**

```
21 typedef struct _FSGPD_RECTANGLE {
22     FSGPD_POINT p0; /* Starting point. */
23     FSGPD_POINT p1; /* Diagonal point. */
24 } FSGPD_RECTANGLE;
```

25 **Return Value**

26 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.

# 27 fsgpdRoundRectanglePath

28 **Name**

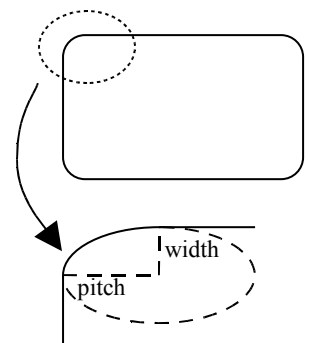
29 fsgpdRoundRectanglePath – Add round rectangles to current path.

30 **Synopsis**

```
31 FSGPD_RESULT fsgpdRoundRectanglePath(
32     FSGPD_DC printerContext,
33     FSGPD_INT nrectangles,
34     FSGPD_ROUNDRECTANGLE *rectangles);
```

35 **Arguments**

36 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.  
 37 nrectangles – Number of round rectangles.  
 38 rectangles – Pointer to an array of RoundRectangle structure.



39 **Description**

40 Append round rectangles into the path.  
 41 Each corner of round rectangle is connected by elliptic arc defined by *xellipse* and *yellipse* in the RoundRectangle structure.  
 42 Lines of a round rectangle are drawn as a rectangle path.

43 **Structures**

```
44 typedef struct _FSGPD_ROUNDRECTANGLE {
45     FSGPD_POINT p0; /* Starting point. */
```

```

1         FSGPD_POINT p1;      /* Diagonal point. */
2         FSGPD_FIX xellipse, yellipse;
3     } FSGPD_ROUNDRECTANGLE;

```

#### 4 **Return Value**

5 FSGPD\_OK when no error, or -1 when error and an error code is set into "fsgpdErrorNo".

## 6 **fsgpdBezierPath**

#### 7 **Name**

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

#### 9 **Synopsis**

```

10 FSGPD_RESULT fsgpdBezierPath(
11     FSGPD_DC printerContext,
12     FSGPD_INT npoints,
13     FSGPD_POINT *points);

```

#### 14 **Arguments**

15 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

17 npoints – Number of points in an array \*points. It must be a multiple number of "3".

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

#### 19 **Description**

20 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.

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

25 If npoints is not a multiple number of "3", this function returns error, and the error code is set as FSGPD\_PARAMERROR.

#### 26 **Return Value**

27 FSGPD\_OK when no error, or -1 when error and an error code is set into "fsgpdErrorNo".

## 28 **fsgpdArcPath**

#### 29 **Name**

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

#### 31 **Synopsis**

```

32 FSGPD_RESULT fsgpdArcPath(
33     FSGPD_DC printerContext,
34     FSGPD_ARCMODE kind,
35     FSGPD_ARCDIR dir,
36     FSGPD_FIX bbx0, FSGPD_FIX bby0, FSGPD_FIX bbx1, FSGPD_FIX bby1,
37     FSGPD_FIX x0, FSGPD_FIX y0,
38     FSGPD_FIX x1, FSGPD_FIX y1);

```

#### 39 **Arguments**

40 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

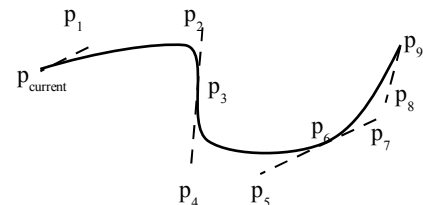
41 kind – Arc: an arc, Chord: a chord, Pie: a pie.

42 dir – The direction of the path. Clockwise: clockwise, Counterclockwise: counter-clockwise.

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

44 (x0, y0) – Starting point.

45 (x1, y1) – End point.



In the above case, each arguments are as following:  
npoints = 9  
\*points = {p<sub>1</sub>, p<sub>2</sub>, ... p<sub>9</sub>}

1 **Description**

2 Append an arc, chord or a pie to the current path. The center point of ellipse is the middle point of the circumscribe rectangle.  
3 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  
4 ellipse is appended into the path. If the circumscribe rectangle is a square, a circle is appended into the path.  
5 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-  
6 top point of the circumscribe rectangle.

7 **Return Value**

8 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.



# 1 **Text Operations**

- 2 Text operations are used for obtain the information of device fonts,draw the device fonts,register the information of device fonts
- 3 and erase the information of device fonts.
- 4 All text operation functions are OPTIONAL.
- 5 Detailed operation is not determined yet.

# 1 Bitmap Image Operations

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

4 The typical draw of bitmap is

- 5 ● fsgpdStartDrawImage() – specified the image data
- 6 ● fsgpdTransferDrawImage() – transfer the image data(1 time or more)
- 7 ● fsgpdEndDrawImage() – end of transfer image data and draw it.

8 If called except fsgpdTransferDrawImage() function between fsgpdStartDrawImage() and fsgpdEndDrawImage() function ,it  
9 returns error and the error code is set as FSGPD\_BADREQUEST.

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

11 All bitmap operation functions are OPTIONAL.

## 12 fsgpdDrawImage

### 13 Name

14 fsgpdDrawImage – Draw bitmap image

### 15 Synopsis

```
16 FSGPD_RESULT fsgpdDrawImage (  
17     FSGPD_DC printerContext,  
18     FSGPD_INT sourceWidth  
19     FSGPD_INT sourceHeight;  
20     FSGPD_INT sourcePitch;  
21     FSGPD_INT colorDepth;  
22     FSGPD_IMAGEFORMAT imageFormat,  
23     FSGPD_RECTANGLE destinationSize,  
24     void *imageData);
```

### 25 Arguments

26 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

27 sourceWidth – Width of source image(pixels)

28 sourceHeight – Height of source image(pixels)

29 sourcePitch – Repetition pitch of source image (bytes)

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

31 imageFormat – Image format enum.

32 destinationSize – Size of destination image.

33 imageDate – The pointer of bitmap image data.

### 34 Description

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

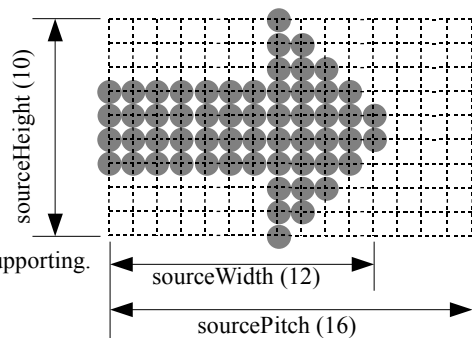
36 If current point is not integer, the reference point is revalued to the integer and  
37 the current point is not updated.

38 SourceWidth and sourceHeight are width and height of the bitmap image  
39 respectively. SourcePitch is the bytes of one raster data in \*imageData (see the  
40 figure right for an example). This argument is effective only when the  
41 imageFoirmat is Raw or RLE.

42 Image format can be used which the device obtained from device capability is supporting.  
43 (Raw, RLE, JPEG, PNG or the vendor supports uniquely can be specified.)

### 44 Return Value

45 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.



## 1 **fsgpdStartDrawImage**

### 2 **Name**

3 fsgpdStartDrawImage – start draw bitmap image

### 4 **Synopsis**

```
5 FSGPD_RESULT fsgpdStartDrawImage(  
6     FSGPD_DC printerContext,  
7     FSGPD_INT sourceWidth  
8     FSGPD_INT sourceHeight;  
9     FSGPD_INT sourcePitch;  
10    FSGPD_INT colorDepth;  
11    FSGPD_IMAGEFORMAT imageFormat,  
12    FSGPD_RECTANGLE destinationSize);
```

### 13 **Arguments**

14 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

15 sourceWidth – Width of source image(pixels)

16 sourceHeight – Height of source image(pixels)

17 sourcePitch – Repetition pitch of source image (bytes)

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

19 imageFormat– Image format enum.

20 destinationSize – Size of destination image.

### 21 **Description**

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

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

24 Image format can be used which the device obtained from device capability is supporting. (Raw, RLE, JPEG, PNG or the  
25 vendor supports uniquely can be specified.)

### 26 **Return Value**

27 FSGPD\_OK when no error,or -1 when error and an error code is set into “fsgpdErrorNo”.

## 28 **fsgpdTransferDrawImage**

### 29 **Name**

30 fsgpdTransferDrawImage – transfer bitmap image data

### 31 **Synopsis**

```
32 FSGPD_RESULT fsgpdTransferDrawImage(  
33     FSGPD_DC printerContext,  
34     FSGPD_INT count,  
35     void *imageData);
```

### 36 **Arguments**

37 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

38 count – The number of bytes of imageData.

39 imageData – The pointer of bitmap image data.

### 40 **Description**

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

### 42 **Return Value**

43 FSGPD\_OK when no error,or -1 when error and an error code is set into “fsgpdErrorNo”.

## 1 **fsgpdEndDrawImage**

### 2 **Name**

3 fsgpdEndDrawImage – end draw bitmap image

### 4 **Synopsis**

```
5 FSGPD_RESULT fsgpdEndDrawImage(  
6     FSGPD_DC printerContext);
```

### 7 **Arguments**

8 printerContext – Printer Context obtained by the fsgpdOpenPrinter() function.

### 9 **Description**

10 Declare the end of drawing image data which is started by fsgpdStartDrawImage() or fsgpdDrawImage() functions.

### 11 **Return Value**

12 FSGPD\_OK when no error, or -1 when error and an error code is set into “fsgpdErrorNo”.

# 1 Scan Line Operations

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

4  
5 All scan line operations are OPTIONAL. However support for scan line operations is recommended in the case where path  
6 operations are not supported and you want to switch the color scheme from bitmap oriented to path oriented.

## 7 **fsgpdStartScanline**

### 8 **Name**

9 `fsgpdStartScanline` – initiates scan line drawing

### 10 **Synopsis**

```
11 FSGPD_RESULT fsgpdStartScanline(  
12     FSGPD_DC printerContext,  
13     FSGPD_INT yposition);
```

### 14 **Arguments**

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

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

### 17 **Description**

18 Initiates scan line drawing. Scan lines are drawn by `fsgpdStartScanline()` to declare the begging scan lines, `fsgpdScanline()`'s to  
19 set actual scan line data, then `fsgpdEndScanline()` to end the operations. Current point is not modified by `fsgpdStartScanline()`.

### 20 **Return Value**

21 Upon successful completion returns `FSGPD_OK`. Otherwise, -1 is returned and the variable `fsgpdErrorNo` is set to indicate  
22 the error.

## 23 **fsgpdScanline**

### 24 **Name**

25 `fsgpdScanline` – draws scan lines

### 26 **Synopsis**

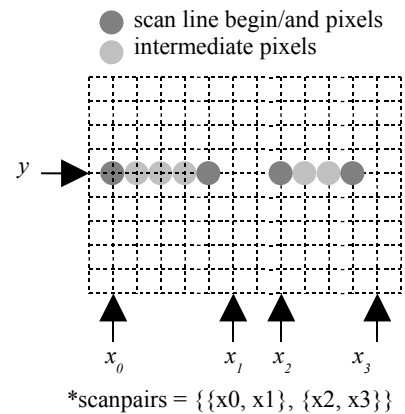
```
27 FSGPD_RESULT fsgpdScanline(  
28     FSGPD_DC printerContext,  
29     FSGPD_INT nscanpairs,  
30     FSGPD_INT *scanpairs);
```

### 31 **Arguments**

32 `printerContext` – a printer context returned by a call to  
33 `fsgpdOpenPrinter()`

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

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



### 36 **Description**

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

### 39 **Return Value**

40 Upon successful completion returns `FSGPD_OK`. Otherwise, -1 is returned and the variable `fsgpdErrorNo` is set to indicate  
41 the error.

## 1 **fsgpdEndScanline**

### 2 **Name**

3 fsgpdEndScanline – terminates scan line drawing

### 4 **Synopsis**

```
5 FSGPD_RESULT fsgpdEndScanline(  
6     FSGPD_DC printerContext);
```

### 7 **Arguments**

8 printerContext – a printer context returned by a call to fsgpdOpenPrinter()

### 9 **Description**

10 Terminates scan line drawing.

### 11 **Return Value**

12 Upon successful completion returns FSGPD\_OK. Otherwise, -1 is returned and the variable fsgpdErrorNo is set to indicate  
13 the error.

# 1 Raster Image Operations

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

## 3 **fsgpdStartRaster**

### 4 **Name**

5 `fsgpdStartRaster` – initiates raster image drawing

### 6 **Synopsis**

```
7 FSGPD_RESULT fsgpdStartRaster(  
8     FSGPD_DC printerContext,  
9     FSGPD_INT rasterWidth);
```

### 10 **Arguments**

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

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

### 13 **Description**

14 Initiates raster data drawing.

15 The raster image will be drawn from current point.

16 The color space as determined by the current Graphics State Object will be used. Possible values include:

17 `FSGPD_CSPACE_BW`, `FSGPD_CSPACE_DEVICEGRAY`, `FSGPD_CSPACE_DEVICECMY`,  
18 `FSGPD_CSPACE_DEVICECMYK`, `FSGPD_CSPACE_STANDARDRGB`, `FSGPD_CSPACE_STANDARDRGB64`.

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

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

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

22

Color Space	# of planes	Bits per Plane	Bits per Pixel	Bytes per Pixel	Note
<code>FSGPD_CSPACE_BW</code>	1	1	1	1/8	bit order required padding bits are added to the rightmost byte if necessary
<code>FSGPD_CSPACE_DEVICEGRAY</code>	1	8	8	1	
<code>FSGPD_CSPACE_STANDARDRGB</code>	3	8	24	3	
<code>FSGPD_CSPACE_STANDARDRGB64</code>	3	16	48	6	byte order required
<code>FSGPD_CSPACE_DEVICECMY</code>	3	8	24	3	
<code>FSGPD_CSPACE_DEVICECMYK</code>	4	8	32	4	

### 23 **Return Value**

24 Upon successful completion returns `FSGPD_OK`. Otherwise, -1 is returned and the variable `fsgpdErrorNo` is set to indicate  
25 the error.

## 26 **fsgpdTransferRasterData**

### 27 **Name**

28 `fsgpdTransferRasterData` – transfers raster image data

### 29 **Synopsis**

```
30 FSGPD_RESULT fsgpdTransferRasterData(  
31     FSGPD_DC printerContext,  
32     FSGPD_INT count,  
33     FSGPD_BYTE *data);
```

1 **Arguments**

- 2 printerContext – a printer context returned by a call to fsgpdOpenPrinter ()
- 3 count – number of pixel data elements
- 4 data – pointer to an array holding the pixel data

5 **Description**

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

9 **Return Value**

10 Upon successful completion returns FSGPD\_OK. Otherwise, -1 is returned and the variable fsgpdErrorNo is set to indicate  
11 the error.

12 **fsgpdSkipRaster**

13 **Name**

14 fsgpdSkipRaster – skips lines during raster image drawing

15 **Synopsis**

```
16 FSGPD_RESULT fsgpdSkipRaster(  
17     FSGPD_DC printerContext,  
18     FSGPD_INT count);
```

19 **Arguments**

- 20 printerContext – a printer context returned by a call to fsgpdOpenPrinter ()
- 21 count – number of lines to be skipped

22 **Description**

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

25 **Return Value**

26 Upon successful completion returns FSGPD\_OK. Otherwise, -1 is returned and the variable fsgpdErrorNo is set to indicate  
27 the error.

28 **fsgpdEndRaster**

29 **Name**

30 fsgpdEndRaster – terminates raster image drawing

31 **Synopsis**

```
32 FSGPD_RESULT fsgpdEndRaster(  
33     FSGPD_DC printerContext);
```

34 **Arguments**

- 35 printerContext – a printer context returned by a call to fsgpdOpenPrinter ()

36 **Description**

37 Terminates raster image drawing.

38 **Return Value**

39 Upon successful completion returns FSGPD\_OK. Otherwise, -1 is returned and the variable fsgpdErrorNo is set to indicate  
40 the error.



# 1 Stream Data Operations

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

## 5 **fsgpdStartStream**

### 6 **Name**

7 fsgpdStartStream – start of streaming data transfer

### 8 **Synopsis**

```
9 FSGPD_RESULT fsgpdStartStream(  
10     FSGPD_DC printerContext);
```

### 11 **Arguments**

12 printerContext – Specifies a printer context obtained using fsgpdOpenPrinter()

### 13 **Description**

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

### 16 **Return Value**

17 The return value is FSGPD\_OK in case of normal finish. In case of error, the return value is –1 and error code is stored in  
18 “fsgpdErrorNo”.

## 19 **fsgpdTransferStreamData**

### 20 **Name**

21 fsgpdTransferStreamData – send stream data

### 22 **Synopsis**

```
23 FSGPD_RESULT fsgpdTransferStreamData(  
24     FSGPD_DC printerContext,  
25     FSGPD_INT count,  
26     void *data);
```

### 27 **Arguments**

28 printerContext – Specifies a printer context obtained using fsgpdOpenPrinter()

29 count – number of data bytes transferred

30 data – pointer to stream data

### 31 **Description**

32 Send streaming data between fsgpdStartStream and fsgpdEndStream.

### 33 **Return Value**

34 The return value is FSGPD\_OK in case of normal finish. In case of error, the return value is –1 and error code is stored in  
35 “fsgpdErrorNo”.

## 36 **fsgpdEndStream**

### 37 **Name**

38 fsgpdEndStream – end of streaming data transfer

### 39 **Synopsis**

```
40 FSGPD_RESULT fsgpdEndStream(  
41     FSGPD_DC printerContext);
```

1 **Arguments**

2 printerContext – Specifies a printer context obtained using fsgpdOpenPrinter()

3 **Description**

4 End of streaming data transfer.

5 **Return Value**

6 The return value is FSGPD\_OK in case of normal finish. In case of error, the return value is –1 and error code is stored in  
7 “fsgpdErrorNo”.

# V.Graphic Operation Fallback

---

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

Normally, the printer driver declares the supported API through the `fsgpdOpenPrinter()` 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.

## 1 **Renderer Fallback**

2 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  
3 support”, the renderer, depending on the API, calls the driver after further breaking down the processing steps in accordance  
4 with the available functionality of the low level driver.

## 1 **Driver Fallback**

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

1

# VI. Macros, Types, Enums and Structures

Following program is a header file which contains macros, types, enums and structures defined in this specification.

```
1  /*
2  * OpenPrinting Vector Printer Driver API Definitions [fsgpd.h]
3  *
4  * Copyright (c) 2006 Free Standards Group
5  * Copyright (c) 2006 Fuji Xerox Printing Systems Co., Ltd.
6  * Copyright (c) 2006 Canon Inc.
7  * Copyright (c) 2003-2006 AXE Inc.
8  *
9  * All Rights Reserved.
10 *
11 * Permission to use, copy, modify, distribute, and sell this software
12 * and its documentation for any purpose is hereby granted without
13 * fee, provided that the above copyright notice appear in all copies
14 * and that both that copyright notice and this permission notice
15 * appear in supporting documentation.
16 *
17 * The above copyright notice and this permission notice shall be
18 * included in all copies or substantial portions of the Software.
19 *
20 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
21 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
22 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
23 * NONINFRINGEMENT. IN NO EVENT SHALL THE OPEN GROUP BE LIABLE FOR
24 * ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
25 * CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
26 * WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
27 */
28
29 #ifndef _FSGPD_H_
30 #define _FSGPD_H_
31
32 /* Return Values and Error Codes */
33 #define FSGPD_OK 0 /* -1 for errors */
34 #define FSGPD_FATALERROR -1 /* error: cannot be recovered */
35 #define FSGPD_BADREQUEST -2 /* error: called where it should not be called */
36 #define FSGPD_BADCONTEXT -3 /* error: invalid printer context */
37 #define FSGPD_NOTSUPPORTED -4 /* error: combination of parameters are set */
38 /* which cannot be handled by driver or printer */
39 #define FSGPD_JOBANCELED -5 /* error: job has been canceled by some cause */
40 #define FSGPD_PARAMERROR -6 /* error: invalid parameter */
41
42 /* Basic Types */
43 typedef int FSGPD_DC; /* driver/device context */
44 typedef int FSGPD_RESULT; /* return value */
45 typedef unsigned char FSGPD_BYTE; /* BYTE */
46 typedef unsigned char FSGPD_CHAR; /* character (string) */
47 typedef int FSGPD_INT; /* integer */
48 typedef int FSGPD_FIX; /* fixed integer */
49 typedef float FSGPD_FLOAT; /* float */
50 typedef unsigned int FSGPD_FLAG; /* flags */
51 typedef unsigned int FSGPD_ROP; /* raster operation */
52
53 /* graphic elements */
54 typedef struct _FSGPD_POINT {
55     FSGPD_FIX x, y;
56 } FSGPD_POINT;
57
58 typedef struct _FSGPD_RECTANGLE {
59     FSGPD_POINT p0; /* start point */
60     FSGPD_POINT p1; /* diagonal point */
61 } FSGPD_RECTANGLE;
62
63 typedef struct _FSGPD_ROUNDRECTANGLE {
64     FSGPD_POINT p0; /* start point */
65     FSGPD_POINT p1; /* diagonal point */
66     FSGPD_FIX xellipse, yellipse;
67 } FSGPD_ROUNDRECTANGLE;
68
```

```

1   } FSGPD_ROUNDRECTANGLE;
2
3   /* Image Formats */
4   typedef enum _FSGPD_IMAGEFORMAT {
5       FSGPD_IFORMAT_RAW           = 0,
6       FSGPD_IFORMAT_RLE           = 1,
7       FSGPD_IFORMAT_JPEG          = 2,
8       FSGPD_IFORMAT_PNG           = 3
9   } FSGPD_IMAGEFORMAT;
10
11  /* Color Presentation */
12  typedef enum _FSGPD_COLORMAPPING {
13      FSGPD_CMAP_DIRECT             = 0,
14      FSGPD_CMAP_INDEXED           = 1
15  } FSGPD_COLORMAPPING;
16
17  typedef enum _FSGPD_CSPACE {
18      FSGPD_CSPACE_BW               = 0,
19      FSGPD_CSPACE_DEVICEGRAY       = 1,
20      FSGPD_CSPACE_DEVICECMY        = 2,
21      FSGPD_CSPACE_DEVICECMYK       = 3,
22      FSGPD_CSPACE_DEVICERGB        = 4,
23      FSGPD_CSPACE_STANDARDRGB      = 5,
24      FSGPD_CSPACE_STANDARDRGB64    = 6
25  } FSGPD_CSPACE;
26
27  /* Fill, Paint, Clip */
28  typedef enum _FSGPD_FILLMODE {
29      FSGPD_FILLMODE_EVENODD        = 0,
30      FSGPD_FILLMODE_WINDING        = 1
31  } FSGPD_FILLMODE;
32
33  typedef enum _FSGPD_PAINTMODE {
34      FSGPD_PAINTMODE_OPAQUE         = 0,
35      FSGPD_PAINTMODE_TRANSPARENT    = 1
36  } FSGPD_PAINTMODE;
37
38  typedef enum _FSGPD_CLIPRULE {
39      FSGPD_CLIPRULE_EVENODD         = 0,
40      FSGPD_CLIPRULE_WINDING         = 1
41  } FSGPD_CLIPRULE;
42
43  /* Line */
44  typedef enum _FSGPD_LINESTYLE {
45      FSGPD_LINESTYLE_SOLID          = 0,
46      FSGPD_LINESTYLE_DASH           = 1
47  } FSGPD_LINESTYLE;
48
49  typedef enum _FSGPD_LINECAP {
50      FSGPD_LINECAP_BUTT              = 0,
51      FSGPD_LINECAP_ROUND             = 1,
52      FSGPD_LINECAP_SQUARE            = 2
53  } FSGPD_LINECAP;
54
55  typedef enum _FSGPD_LINEJOIN {
56      FSGPD_LINEJOIN_MITER            = 0,
57      FSGPD_LINEJOIN_ROUND            = 1,
58      FSGPD_LINEJOIN_BEVEL           = 2
59  } FSGPD_LINEJOIN;
60
61  /* Brush */
62  typedef enum _FSGPD_BDTYPE {
63      FSGPD_BDTYPE_NORMAL             = 0
64  } FSGPD_BDTYPE;
65
66  typedef struct _FSGPD_BRUSHDATA {
67      FSGPD_BDTYPE type;
68      FSGPD_INT width, height, pitch;
69      #if defined(__GNUC__) && __GNUC__ <= 2
70          FSGPD_BYTE data[1];
71      #else
72          FSGPD_BYTE data[];

```

```

1  #endif
2
3  } FSGPD_BRUSHDATA;
4
5  typedef struct _FSGPD_BRUSH {
6      FSGPD_CSPACE colorSpace;
7      FSGPD_INT color[4];          /* aRGB quadruplet */
8      FSGPD_INT xorg, yorg;       /* brush origin */
9                                  /* ignored for fsgpdSetBgColor */
10     FSGPD_BRUSHDATA *pbrush;    /* pointer to brush data */
11                                  /* solid brush used, if NULL */
12 } FSGPD_BRUSH;
13
14 /* Misc. Flags */
15 typedef enum _FSGPD_ARCMODE {
16     FSGPD_ARC                = 0,
17     FSGPD_CHORD              = 1,
18     FSGPD_PIE                = 2
19 } FSGPD_ARCMODE;
20
21 typedef enum _FSGPD_ARCDIR {
22     FSGPD_CLOCKWISE          = 0,
23     FSGPD_COUNTERCLOCKWISE  = 1
24 } FSGPD_ARCDIR;
25
26 typedef enum _FSGPD_PATHMODE {
27     FSGPD_PATHCLOSE          = 0,
28     FSGPD_PATHOPEN          = 1
29 } FSGPD_PATHMODE;
30
31 /* CTM */
32 typedef struct _FSGPD_CTM {
33     FSGPD_FLOAT a, b, c, d, e, f;
34 } FSGPD_CTM;
35
36 /* Device Information and Capabilites */
37 typedef enum _FSGPD_QUERYINFOFLAGS {
38     FSGPD_QF_DEVICERESOLUTION    = 0x00000001,
39     FSGPD_QF_MEDIASIZE           = 0x00000002,
40     FSGPD_QF_PAGEROTATION        = 0x00000004,
41     FSGPD_QF_MEDIANUP            = 0x00000008,
42     FSGPD_QF_MEDIADUPLEX         = 0x00000010,
43     FSGPD_QF_MEDIASOURCE         = 0x00000020,
44     FSGPD_QF_MEDIADESTINATION    = 0x00000040,
45     FSGPD_QF_MEDIATYPE           = 0x00000080,
46     FSGPD_QF_MEDIASIZE           = 0x00010000,          /* only for fsgpdQueryDeviceInfo
47 */
48     FSGPD_QF_PRINTREGION         = 0x00020000 /* only for fsgpdQueryDeviceInfo */
49 } FSGPD_QUERYINFOFLAGS;
50
51 /* Function prototype */
52 FSGPD_DC fsgpdOpenPrinter(
53     FSGPD_INT outputFD,
54     FSGPD_CHAR *printerModel,
55     FSGPD_INT apiVersion[2],
56     FSGPD_INT *nApiEntry,
57     FSGPD_INT (**apiEntry[])());
58
59 /* API Procedure Entries */
60 typedef struct _FSGPD_API_PROCS {
61     FSGPD_DC
62     (*fsgpdOpenPrinter) (FSGPD_INT, FSGPD_CHAR*, FSGPD_INT*, FSGPD_INT*, struct
63     _FSGPD_API_PROCS**);
64     FSGPD_RESULT (*fsgpdClosePrinter) (FSGPD_DC);
65     FSGPD_RESULT (*fsgpdStartJob) (FSGPD_DC, FSGPD_CHAR*);
66     FSGPD_RESULT (*fsgpdEndJob) (FSGPD_DC);
67     FSGPD_RESULT (*fsgpdAbortJob) (FSGPD_DC);
68     FSGPD_RESULT (*fsgpdStartDoc) (FSGPD_DC, FSGPD_CHAR*);
69     FSGPD_RESULT (*fsgpdEndDoc) (FSGPD_DC);
70     FSGPD_RESULT (*fsgpdStartPage) (FSGPD_DC, FSGPD_CHAR*);
71     FSGPD_RESULT (*fsgpdEndPage) (FSGPD_DC);
72

```

```

1      FSGPD_RESULT
2      (*fsgpdQueryDeviceCapability) (FSGPD_DC, FSGPD_FLAG, FSGPD_INT, FSGPD_BYTE*);
3      FSGPD_RESULT
4      (*fsgpdQueryDeviceInfo) (FSGPD_DC, FSGPD_FLAG, FSGPD_INT, FSGPD_CHAR*);
5      FSGPD_RESULT (*fsgpdResetCTM) (FSGPD_DC);
6      FSGPD_RESULT (*fsgpdSetCTM) (FSGPD_DC, FSGPD_CTM*);
7      FSGPD_RESULT (*fsgpdGetCTM) (FSGPD_DC, FSGPD_CTM*);
8      FSGPD_RESULT (*fsgpdInitGS) (FSGPD_DC);
9      FSGPD_RESULT (*fsgpdSaveGS) (FSGPD_DC);
10     FSGPD_RESULT (*fsgpdRestoreGS) (FSGPD_DC);
11     FSGPD_RESULT (*fsgpdQueryColorSpace) (FSGPD_DC, FSGPD_CSPACE*, FSGPD_INT*);
12     FSGPD_RESULT (*fsgpdSetColorSpace) (FSGPD_DC, FSGPD_CSPACE);
13     FSGPD_RESULT (*fsgpdGetColorSpace) (FSGPD_DC, FSGPD_CSPACE*);
14     FSGPD_RESULT (*fsgpdQueryROP) (FSGPD_DC, FSGPD_INT*, FSGPD_ROP*);
15     FSGPD_RESULT (*fsgpdSetROP) (FSGPD_DC, FSGPD_ROP);
16     FSGPD_RESULT (*fsgpdGetROP) (FSGPD_DC, FSGPD_ROP*);
17     FSGPD_RESULT (*fsgpdSetFillMode) (FSGPD_DC, FSGPD_FILLMODE);
18     FSGPD_RESULT (*fsgpdGetFillMode) (FSGPD_DC, FSGPD_FILLMODE*);
19     FSGPD_RESULT (*fsgpdSetAlphaConstant) (FSGPD_DC, FSGPD_FLOAT);
20     FSGPD_RESULT (*fsgpdGetAlphaConstant) (FSGPD_DC, FSGPD_FLOAT*);
21     FSGPD_RESULT (*fsgpdSetLineWidth) (FSGPD_DC, FSGPD_FIX);
22     FSGPD_RESULT (*fsgpdGetLineWidth) (FSGPD_DC, FSGPD_FIX*);
23     FSGPD_RESULT (*fsgpdSetLineDash) (FSGPD_DC, FSGPD_FIX*, FSGPD_INT);
24     FSGPD_RESULT (*fsgpdGetLineDash) (FSGPD_DC, FSGPD_FIX*, FSGPD_INT*);
25     FSGPD_RESULT (*fsgpdSetLineDashOffset) (FSGPD_DC, FSGPD_FIX);
26     FSGPD_RESULT (*fsgpdGetLineDashOffset) (FSGPD_DC, FSGPD_FIX*);
27     FSGPD_RESULT (*fsgpdSetLineStyle) (FSGPD_DC, FSGPD_LINESTYLE);
28     FSGPD_RESULT (*fsgpdGetLineStyle) (FSGPD_DC, FSGPD_LINESTYLE*);
29     FSGPD_RESULT (*fsgpdSetLineCap) (FSGPD_DC, FSGPD_LINECAP);
30     FSGPD_RESULT (*fsgpdGetLineCap) (FSGPD_DC, FSGPD_LINECAP*);
31     FSGPD_RESULT (*fsgpdSetLineJoin) (FSGPD_DC, FSGPD_LINEJOIN);
32     FSGPD_RESULT (*fsgpdGetLineJoin) (FSGPD_DC, FSGPD_LINEJOIN*);
33     FSGPD_RESULT (*fsgpdSetMiterLimit) (FSGPD_DC, FSGPD_FIX);
34     FSGPD_RESULT (*fsgpdGetMiterLimit) (FSGPD_DC, FSGPD_FIX*);
35     FSGPD_RESULT (*fsgpdSetPaintMode) (FSGPD_DC, FSGPD_PAINTMODE);
36     FSGPD_RESULT (*fsgpdGetPaintMode) (FSGPD_DC, FSGPD_PAINTMODE*);
37     FSGPD_RESULT (*fsgpdSetStrokeColor) (FSGPD_DC, FSGPD_BRUSH*);
38     FSGPD_RESULT (*fsgpdSetFillColor) (FSGPD_DC, FSGPD_BRUSH*);
39     FSGPD_RESULT (*fsgpdSetBgColor) (FSGPD_DC, FSGPD_BRUSH*);
40     FSGPD_RESULT (*fsgpdNewPath) (FSGPD_DC);
41     FSGPD_RESULT (*fsgpdEndPath) (FSGPD_DC);
42     FSGPD_RESULT (*fsgpdStrokePath) (FSGPD_DC);
43     FSGPD_RESULT (*fsgpdFillPath) (FSGPD_DC);
44     FSGPD_RESULT (*fsgpdStrokeFillPath) (FSGPD_DC);
45     FSGPD_RESULT (*fsgpdSetClipPath) (FSGPD_DC, FSGPD_CLIPRULE);
46     FSGPD_RESULT (*fsgpdResetClipPath) (FSGPD_DC);
47     FSGPD_RESULT (*fsgpdSetCurrentPoint) (FSGPD_DC, FSGPD_FIX, FSGPD_FIX);
48     FSGPD_RESULT (*fsgpdLinePath) (FSGPD_DC, FSGPD_PATHMODE, FSGPD_INT, FSGPD_POINT*);
49     FSGPD_RESULT (*fsgpdPolygonPath) (FSGPD_DC, FSGPD_INT, FSGPD_INT*, FSGPD_POINT*);
50     FSGPD_RESULT (*fsgpdRectanglePath) (FSGPD_DC, FSGPD_INT, FSGPD_RECTANGLE*);
51     FSGPD_RESULT
52     (*fsgpdRoundRectanglePath) (FSGPD_DC, FSGPD_INT, FSGPD_ROUNDRECTANGLE*);
53     FSGPD_RESULT (*fsgpdBezierPath) (FSGPD_DC, FSGPD_INT, FSGPD_POINT*);
54     FSGPD_RESULT
55     (*fsgpdArcPath) (FSGPD_DC, FSGPD_ARCMODE, FSGPD_ARCDIR, FSGPD_FIX, FSGPD_FIX, FSGPD_FIX, FSGPD_FIX,
56     FSGPD_FIX, FSGPD_FIX, FSGPD_FIX, FSGPD_FIX, FSGPD_FIX);
57     FSGPD_RESULT
58     (*fsgpdDrawImage) (FSGPD_DC, FSGPD_INT, FSGPD_INT, FSGPD_INT, FSGPD_INT, FSGPD_IMAGEFORMAT,
59     FSGPD_RECTANGLE, void*);
60     FSGPD_RESULT
61     (*fsgpdStartDrawImage) (FSGPD_DC, FSGPD_INT, FSGPD_INT, FSGPD_INT, FSGPD_INT, FSGPD_INT, FSGPD_IMAGEFO
62     RMAT, FSGPD_RECTANGLE);
63     FSGPD_RESULT (*fsgpdTransferDrawImage) (FSGPD_DC, FSGPD_INT, void*);
64     FSGPD_RESULT (*fsgpdEndDrawImage) (FSGPD_DC);
65     FSGPD_RESULT (*fsgpdStartScanline) (FSGPD_DC, FSGPD_INT);
66     FSGPD_RESULT (*fsgpdScanline) (FSGPD_DC, FSGPD_INT, FSGPD_INT*);
67     FSGPD_RESULT (*fsgpdEndScanline) (FSGPD_DC);
68     FSGPD_RESULT (*fsgpdStartRaster) (FSGPD_DC, FSGPD_INT);
69     FSGPD_RESULT (*fsgpdTransferRasterData) (FSGPD_DC, FSGPD_INT, FSGPD_BYTE*);
70     FSGPD_RESULT (*fsgpdSkipRaster) (FSGPD_DC, FSGPD_INT);
71     FSGPD_RESULT (*fsgpdEndRaster) (FSGPD_DC);
72     FSGPD_RESULT (*fsgpdStartStream) (FSGPD_DC);

```

```
1         FSGPD_RESULT (*fsgpdTransferStreamData) (FSGPD_DC, FSGPD_INT, void*);
2         FSGPD_RESULT (*fsgpdEndStream) (FSGPD_DC);
3     } FSGPD_API_PROCS;
4
5 #endif /* _FSGPD_H_ */
```



# VII. Authors and Contributors

---

1

## 1 Editors

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

## 1 Authors

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

## 1 Contributors

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

- 1
- 2 Version 0.1 (Japanese) Mihara/Toratani
- 3 Version 0.2 (Japanese) Mihara
- 4     Openprinting-0.1.1 by opfc have implemented based on this version
- 5 Version 0.2-en 2004-3-14 Mihara/Toratani/Kitayama/Yamagishi/Kanjo
- 6     Translation to English
- 7     Some feedback from opfc implementation
- 8 Version 1.0 RC1 2005-7-20 Mihara
- 9     Change copyright notice from FDL to MIT style copyright
- 10     Delete temporary font operations.
- 11 Version 1.0 RC2 2006-6-9/2006-6-17/2006-6-20/2006-6-26/2006-6-29
- 12     Add API version number to OpenPrinter() parameter
- 13     Add pitch and delete count to/from DrawImage()/StartDrawImage()
- 14     Add a figure for SetMiterLimit.
- 15     Change function names/constants/enum/structures names to add FSG prefixes.
- 16     Change copyright year and dates
- 17     Delete “Printer Driver Database”
- 18     Delete \*cmap\* from CSPASE chart.
- 19     Fix parameter colorSpace (wrong) to colorDepth (correct) for DrawImage()/StartDrawImage
- 20     Change description for fsgpdStartJob/fsgpdEndJob/fsgpdAbortJob
- 21     Add type FSGPD\_CHAR for character type
- 22 Version 1.0 RC2 2006-10-20
- 23     Add FSGPD\_CSPACE\_DEVICEKRGB
- 24     Remove “current implementation” descriptions from Color Scheme and Color Space descriptions.
- 25     Remove “described later” descriptions
- 26     Make support of updf for scheme “MUST.”