Foxit Reader SDK

For Windows and Linux

Programming Guide

Revision 1.4

2005.12

Overview
Features
Tutorials
Calling from Different Programming Languages
Reference
Redistribution

Overview

Foxit Reader SDK is developed by Foxit Software Company for high-quality rendering of PDF document pages integrated into a wide range of applications, with minimized resource demand and redistribution size.

Foxit Reader SDK is specialized in PDF document rendering, it doesn't provide PDF parsing, editing, or creation capabilities. Although it exports only a few functions, applications can use it to do all the rendering required for any type of PDF manipulation.

Foxit Reader SDK supports almost all PDF page description features, except those very rarely used feature. The "Features" section lists all major features supported by the SDK.

Foxit Reader SDK is carefully designed and implemented so that for most documents and pages, it achieves a rendering speed comparable or faster than Adobe Acrobat Reader, while maintaining almost same rendering quality.

Foxit Reader SDK runs on Windows 95/NT or later, and most of the functions also run on Linux (any recent releases). For detailed information on runtime configuration please refer to redistribution section.

Evaluation Copy: The downloaded version of Foxit Reader SDK is for evaluation use only. It displays some messages along with any PDF documents you render. If you purchased a license of Foxit Reader SDK, Foxit will send a licensed copy of the SDK to you, and you need to call FPDF_UnlockDLL function to get rid of those evaluation messages. The function is described in the header file. You don't need to call this function if you are just evaluating the SDK (it does nothing if you call it).

Features

Here is a list of major PDF features supported by FOXIT READER SDK:

1. PDF Specification: 1.3, 1.4, and 1.5
2. Automatic cross reference rebuild for documents damaged or generated by non-standard creator
3. Compress Decoders: LZWDecode, FlateDecode, CCITTFaxDecode, DCTDecode
4. Standard 40 bit and 128 bit encryption support
5. All graphic objects: Path, Text, Image, Form, and Shading
6. All color spaces: RGB, Gray, CMYK, ICCBased, Lab, Indexed, Separation, etc
7. Tiling patterns and shadings
8. Images: monochrome/colored image, JPEG images, masked and transparent images
9. All font types: Type1, TrueType, Type0, Type3
10. All CJK (Chinese/Japanese/Korean) character set support (additional redistribution file required)
11. All text rendering modes: fill, stroke, or clipping
12. Anti-aliasing of text rendering
13. Transparent groups (including text, path, and image)
14. Display annotations
15. Unlimited page zoom factor
16. Page rotation at 90, 180 and 270 degrees

Tutorials

In this section we provide easy to follow tutorials on how to use FOXIT READER SDK to render PDF pages as you may need. Note we use C language in all tutorials, if you are using other programming languages, please refer to “Calling from Different Programming Languages” section for information on how you can call the DLL. For C/C++ language, you can include the FPDFVIEW.H header file in your source code, and call the functions listed in the header file. You also need to link with the FPDFVIEW.LIB file (on Windows) or libfpdfview.so (on Linux).

TUTORIAL 1: Load PDF Document and Page

In this tutorial, we will load a PDF document named “testdoc.pdf”, then load the first page of the document. In order to render any page, the page must be loaded first.

```
#include “fpdfview.h”
…
FPDF_DOCUMENT pdf_doc;
FPDF_PAGE pdf_page;

// Before we can do anything, we need to unlock the DLL
// NOTE: If you are evaluating FOXIT READER SDK, you don’t need unlock the DLL,
// then evaluation marks will be shown with all rendered pages.
FPDF_UnlockDLL(“license_id”, “unlock_code”);
// first, load the document (no password specified)
pdf_doc = FPDF_LoadDocument(“testdoc.pdf”, NULL);
if (pdf_doc == NULL) … // error handling
// now load the first page (page index is 0)
pdf_page = FPDF_LoadPage(pdf_doc, 0);
if (pdf_page == NULL) …    // error handling
```

TUTORIAL 2: Basic Rendering on Windows

In this tutorial, we will render a loaded page onto a screen window, within a specified screen rectangle area. Note we don’t show you the whole program here, only the handling of WM_PAINT message is shown. You need put the code into the right place according to the development tool you choose to use.

Let us assume we have a handle to the window, which is stored in hWnd variable.

```
…
HWND hWnd;
FPDF_DOCUMENT pdf_doc; // you must load document before page can be loaded
FPDF_PAGE pdf_page; // you must load page before it can be rendered
…
// Get handle to Device Context (DC) for the window
HDC hDC = GetDC(hWnd);
// Call FPDF_RenderPage function to render the whole page
FPDF_RenderPage(hDC, pdf_page, 10, 10, 400, 500, 0, 0);
```

The codes above will render the whole page onto the window, with left-top corner of the page at point {10, 10} (coordination relative to the left-top corner of the window), and the page will be fit into a rectangle with width of 400 pixels and height of 500 pixels. The page will not be rotated.

TUTORIAL 3: Basic Rendering on Linux

Foxit Reader SDK doesn't directly support X-Windows output, however, you can use the Foxit Device Independent Bitmap (FXDIB) features provided in the SDK to render any PDF page into a bitmap device. You can then transfer the bitmap onto X-Windows surface, or image file, or other type of devices.

Before you can use a bitmap device, you need to create one, you can use FPDFBitmap_Create function to do that. Foxit Reader SDK supports two types of bitmaps: RGB bitmap and ARGB bitmap, you can choose which format you want to use when you create the bitmap.

After the bitmap created, you need to fill the buffer with background of you like. Normally it can be a white background or a non-colored background (if alpha channel is used).

Now you can use it in FPDF_RenderPageBitmap function. This function is pretty much the same as FPDF_RenderPage function, but instead of sending output to a Windows device, it outputs to the bitmap device.

Here is a sample for basic rendering on Linux:

```
FPDF_BITMAP bitmap;
FPDF_DOCUMENT pdf_doc; // you must load document before page can be loaded
FPDF_PAGE pdf_page; // you must load page before it can be rendered
…
// Create bitmap first. We don't use alpha channel here.
bitmap = FPDFBitmap_Create(400, 500, FALSE);

// Fill the bitmap with white background
```

```
FPDFBitmap_FillRect(bitmap, 0, 0, 400, 500, 255, 255, 255, 255);

// Call FPDF_RenderPageBitmap function to render the whole page
FPDF_RenderPageBitmap(bitmap, pdf_page, 10, 10, 400, 500, 0, 0);
```

The following tutorials introduce more features of rendering in Foxit Reader SDK. All the samples use FPDF_RenderPage function, however the same techniques apply to FPDF_RenderPageBitmap, in the same way.

TUTORIAL 4: Zoom the Page

In tutorial 2 we rendered the page into a fixed size rectangle. This is simple, however, it's not that useful. In most cases, you want to either show the PDF page in its original size, or zoom in/zoom out for larger/smaller display. All these operations can be achieved by adjustment of the size parameters (the fifth and sixth parameters) of FPDF_RenderPage function.

If you want to show the page in its original size, you first need to know the original page of the page, this is easy, you can call the following functions to get width and height of a page:

```
double page_width, page_height;
…
page_width = FPDF_GetPageWidth(pdf_page);
page_height = FPDF_GetPageHeight(pdf_page);
```

Note the page width and height returned from above two functions are measured in typographic POINTS. In typography, one point equals to 1/72 inch, which is about 0.35 milli-miter. You can not directly pass the width or height in points into FPDF_RenderPage function, because FPDF_RenderPage accepts only pixels for screen size. Fortunately you can convert the points to pixels if you know the size of one pixel on your computer screen:

```
int logpixelsx, logpixelsy, size_x, size_y;
HDC hDC;
…
// get number of pixels per inch (horizontally and vertically)
logpixelsx = GetDeviceCaps(hDC, LOGPIXELSX);
logpixelsy = GetDeviceCaps(hDC, LOGPIXELSY);
// convert points into pixels
Size_x = page_width / 72 * logpixelsx;
size_y = page_height / 72 * logpixelsy;
```

After we get the original page size in pixels, we can use it in the FPDF_RenderPage to display the page in

its original size:

```
…
FPDF_RenderPage(hDC, pdf_page, 10, 10, size_x, size_y, 0, 0);
```

Now, you may want to zoom in or out the page on the screen. Note you can zoom on both vertical and horizontal direction, with separate zoom factors. Again, all you need to do is to adjust the size_x and size_y parameter when you call FPDF_RenderPage function.

The following call shows the page with 200% zoom factor:

```
…
FPDF_RenderPage(hDC, pdf_page, 10, 10, size_x * 2, size_y * 2, 0, 0);
```

While the following call shrink the page by 50%:

```
…
FPDF_RenderPage(hDC, pdf_page, 10, 10, size_x / 2, size_y / 2, 0, 0);
```

TUTORIAL 5: Scroll the Page

Sometimes your PDF viewing codes need to provide scroll capability, especially when the page is zoom into large size exceeding the displayable area in the screen window. To scroll the page to different position, horizontally or vertically, you can adjust the start_x and start_y parameters when you call FPDF_RenderPage function.

Because start_x and start_y parameters always specify the position of left-top corner for displaying the page, so when you scroll down, the left-top corner of the page will go up, therefore the start_y parameter should be decreased. Sometimes the parameter can go negative, it's OK, that's the way to tell FPDF_RenderPage function that the left-top corner of the page is not visible any more. Likewise, when you scroll up, the start_y will increase. When you scroll left/right, the start_x parameters will increase/decrease accordingly.

Here is an example how you can scroll the page vertically to half of its height:

```
int start_y;
…
// Calculate the new start_y, decreased by the scroll pixel size
start_y = 10 – page_height / 2;
FPDF_RenderPage(hDC, pdf_page, 10, start_y, size_x, size_y, 0, 0);
```

TUTORIAL 6: Rotate the Page

PDF pages can be displayed upright like it is, or, it can be rotated in both direction (clockwise or counter-clockwise), or it can be displayed upside-down. To achieve different rotation display, you just need to adjust the rotate parameter when you call FPDF_RenderPage function.

The rotate parameter can be 0, 1, 2, or 3. Use 0 for displaying not rotated, 1 for rotated (90 degrees) clockwise, 2 for upside-down, 3 for rotated (90 degrees) counter-clockwise.

The following example shows how you can rotate the page clockwise:

```
…
FPDF_RenderPage(hDC, pdf_page, 10, 10, size_x, size_y, 1, 0);
```

TUTORIAL 7: Render Part of the Page

Very often you don’t want to re-render the whole page because it might be time consuming, you just want to re-render the area that needs to be re-painted. FPDF_RenderPage supports partial rendering through the standard Windows clip box mechanism, that is, if you have clip box correctly set, FPDF_RenderPage automatically render only page objects (texts, graphics, or images) inside the clip box.

You can set the clip box implicitly using WIN32 function InvalidateRect, or, call one of the clipping WIN32 functions (like SelectClipPath) to set the clip box explicitly.

Here is an example showing how you can call InvalidateRect to set a clip box, so only partial page will be re-rendered. We assume you have handled WM_PAINT message correctly by calling FPDF_RenderPage function.

```
RECT rect;
int left, right, top, bottom;
…
rect.left = left;
rect.right = right;
rect.top = top;
rect.bottom = bottom;
InvalidateRect(hWnd, &rect, TRUE);
```

Note: for correctly re-render the page content, you MUST clear the drawing area first, as we indicated in the

last parameter of InvalidateRect function call. And you need to handle WM_ERASEBKGND function to fill the background with pure white.

TUTORIAL 8: Print Pages

Output PDF pages to printer is almost the same as displaying the pages on screen. Windows system treats both screen and printer as display devices. So you can get a DC (Device Context) for a printer, and then you can use the same FPDF_RenderPage function to output to the printer.

The only difference would be for printing you need to deal with documents and pages. We don't discuss how to get a printer DC here, please refer to WIN32 documents or documents about your development tool.

Here is an example on how you can print a loaded page:

```
HDC hDC;
DOCINFO doc_info;
…
// Get a printer DC
hDC = CreateDC("WINSPOOL", "Printer Name", NULL, NULL);
// Start a printer job
StartDoc(hDC, &doc_info);
// Start a new printing page
StartPage(hDC);
…  // we need to calculate the page size here
// Now we can render the page to the printer
FPDF_RenderPage(hDC, pdf_page, 0, 0, size_x, size_y, 0, 0);
…
```

TUTORIAL 9: Convert Page into Bitmap

Another application of FPDF_RenderPage function is converting page into a bitmap. Windows allows you can create a memory DC which holds a bitmap inside, when you draw onto that DC, nothing really got output, except the bitmap got changed. Therefore, after FPDF_RenderPage finished its rendering into a bitmap DC, you can take the bitmap anywhere, and save it into an image file, with any format you may like.

Here is an example:

```
HDC hDC, hMemDC;
HBITMAP hBitmap;
```

```
…
// Create a memory DC, which is compatible with the screen DC
hMemDC = CreateCompatibleDC(hDC);
// Create a bitmap for output
hBitmap = CreateCompatibleBitmap(hDC, size_x, size_y);
// Select the bitmap into the memory DC
hBitmap = (HBITMAP)SelectObject(hMemDC, hBitmap);
// Now render the page onto the memory DC, which in turn changes the bitmap
FPDF_RenderPage(hMemDC, pdf_page, 0, 0, size_x, size_y, 0, 0);
// Release the bitmap from the memory DC
hBitmap = (HBITMAP)SelectObject(hMemDC, hBitmap);
// Now the bitmap is filled with page contents.
// You can save it using WIN32 or other tools
…
```

Calling from Different Programming Languages

FOXIT READER SDK is a standalone dynamic libraray (DLL on Windows or shared library on Linux), it doesn't require any runtime language support (except basic Windows or Linux components), therefore, you can call the SDK from any programming language which supports Windows system function call or Linux shared library calls.

FOXIT READER SDK uses only basic data types including integer, double, zero-terminated string, and opaque pointers. You can find any of these types in any programming language. NOTE: for opaque pointer types defined in FPDFVIEW.H header file, including PDF_DOCUMENT and PDF_PAGE, can be treated simply as 32 bit integers.

Most programming languages support two types of DLL calling:

1. **Static loading**: you can declare the functions in your own language, at the compile time, so when the program starts, it will automatically look for FOXIT READER SDK and load all functions;
2. **Dynamic loading**: you can call Windows system functions to load the DLL at any time after your program starts, then you can load necessary functions from the DLL, and call them.

In this section, we discuss how to call FOXIT READER SDK from several popular programming languages, other than C/C++ language. We won't going to very details of the coding, instead we provide helpful link to web resources which provide more detailed information about calling DLL from different programming languages.

Calling from C# Managed Codes

In C# managed codes, you can call native DLL calls using so-called Platform Invoke (PInvoke). In the FPDFVIEW.ZIP package, you can find a SAMPLE.CS file which contains a wrapping class created by Foxit developer, you can use it directly. In that file there is also some sample codes showing how you can render a PDF page within C#.

Here is a complete tutorial online from Microsoft website on how to call DLLs from C#:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vcwlkplatforminvoketutorial.asp

Calling from Delphi

Delphi supports both dynamic loading and static loading of DLLs. For more information, please refer to the following page on Borland's website:

http://info.borland.com/techpubs/delphi/delphi5/oplg/dllpackg.html

Calling from Visual Basic

The “sample.frm” file within the SDK package includes VB declaration of all FOXIT READER SDK functions, and example codes for calling those functions to render a PDF page on a form control.

For detailed information on how you can call DLL functions from Visual Basic applications, please refer to the following website from Microsoft:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon98/html/vbconaccessingdllswindowsapi.asp

Function Reference

Details of functions in Foxit Reader SDK are removed from this section. For latest information on each function, please refer to the header file (fpdfview.h).

Redistribution

NOTE: You can only redistribute files listed below to customers of your application, under a written SDK license agreement with Foxit. You can not distribute any part of the SDK to general public, even with a SDK license agreement. Without SDK license agreement, you can not redistribute anything.

For better performance and rendering quality, some Windows system DLL may also be redistributed with FOXIT READER SDK, redistribution of such Windows DLLs is subject to redistribution license agreement with Microsoft. You can visit Microsoft website for more information.

Operating System	Foxit DLL	System DLL
Windows XP or later	FPDFVIEW.DLL	None
Windows 98/Me/2000	FPDFVIEW.DLL	GDIPLUS.DLL
Windows 95/NT	FPDFVIEW.DLL	IE 4.0 or later required
Linux	libfpdfview.so	None

For Windows DLLs, you can visit Microsoft website or get from a system running other OS. Or you can write to support@foxitsoftware.com for assistance.