Foxit Print Manager

Introduction

Foxit Print Manager is a versatile tool for adding PDF printing functionality to your software on Windows.

Features

- Uses the Foxit PDF SDK for page rendering
- Auto page scaling and rotation
- Advanced job control
- DLL with a C interface can be controlled from any programming language
- C# interface class provided for .NET Core and .NET Framework

How is Foxit Print Manager distributed?

Foxit Print Manager contains a DLL which in turn uses the Foxit PDF SDK DLL. The following files are included with the distribution:

lib\fsdk_printmanager_win32.dll	The 32-bit build of the Foxit Print Manager DLL
lib\fsdk_printmanager_win64.dll	The 64-bit build of the Foxit Print Manager DLL
lib\fsdk_win32.dll	The 32-bit build of the Foxit PDF SDK DLL
lib\fsdk_win64.dll	The 64-bit build of the Foxit PDF SDK DLL
lib*.lib	C/C++ import libs for the DLLs
doc\Foxit PDF SDK Print Manager C API Reference.pdf	The function reference for the C interface
doc\Foxit PDF SDK Print Manager .NET API Reference.pdf	The function reference for the .NET interface (C#)
doc\Foxit PDF SDK Print Manager Developer Guide.pdf	The PDF version of the developer guide
examples\ExtensionsDemo\printmanager\	Demo projects for .NET and C++

Deployment

All the DLLs must be deployed into the same directory.

Using the C# interface for .NET

Simply add the **FoxitPrintManager.cs** file to your project and create a new instance of the **FoxitPrintManager.PrintManagerDLL** class.

```
// Use the correct path/filename for the DLL
string dllName = "..\\..\\lib\\fsdk_printmanager_win32.dll";
var printManager = new FoxitPrintManager.PrintManagerDLL(dllName);
```

Note: If you are using the .NET Framework 4.6 or earlier, rather use the **FoxitPrintManager4.6.cs** file which contains custom UTF-8 string marshalling code.

Initializing the print manager

Use the **InitPrintManager** function to initialize the print manager to unlock all the functionality. A valid serial number and license key for Foxit PDF SDK should be provided.When there is no need to use print manager any more, please call the ReleasePrintManager function of PrintManage to release it.

```
string sn = "... Foxit PDF SDK serial number here ...";
string key = "... Foxit PDF SDK license key here ...";
if (printManager.InitPrintManager(sn, key) == 1)
{
    // Other Print Manager function calls can now be made
}
```

Quickly printing a PDF

The easiest way to print a PDF from a file on disk is to use the **PrintPDFFromFile** function. You just need the file name and the password, if any, and a single function call:

```
string fileToPrint = "C:\\MyFiles\\MyDocument.pdf";
string filePassword = "";
if (File.Exists(fileToPrint))
{
    printManager.PrintPDFFromFile(fileToPrint, filePassword);
}
```

The document will be printed to the default printer.

Quickly printing a PDF from memory

If the PDF exists in memory as a byte array, it can be printed easily using the **PrintPDFFromMemory** function.

```
byte[] fileDataToPrint = File.ReadAllBytes("...");
string filePassword = "";
printManager.PrintPDFFromMemory(fileToPrint, filePassword);
```

Creating a print job

For any advanced printing, it's best to first create a print job. Once a print job has been created, multiple documents can be added to it and there are many different settings that can be controlled. It's also possible to monitor the print job while printing is in progress.

The first step is to call the **NewPrintJob** function. This will return a value that can be used as the jobID parameter for all the subsequent calls to print job related functions.

One or more files can be added to the print job either from disk (using **AddPDFFromFileToJob**) or from memory (using **AddPDFFromMemoryToJob**).

When the job is ready to start, call the **BeginPrintJob** function and printing will begin.

This example shows how to create a job, add three files, set the duplex and begin printing:

```
int jobID = printManager.NewPrintJob();
printManager.AddPDFFromFileToJob(jobID, "C:\\docs\\file1.pdf");
printManager.AddPDFFromFileToJob(jobID, "C:\\docs\\file2.pdf");
printManager.AddPDFFromFileToJob(jobID, "C:\\docs\\file3.pdf");
int duplexMode = 2; // Vertical duplex
printManager.SetDuplex(jobID, duplexMode);
printManager.BeginPrintJob(jobID);
```

Getting a list of printers

The printer names for the current system can be enumerated using the **GetPrinterCount** and **GetPrinterName** functions.

```
int printerCount = printManager.GetPrinterCount();
List<string> printers = new List<string>();
for (int printerIndex = 0; printerIndex < printerCount; printerIndex++)
{
    printers.Add(printManager.GetPrinterName(printerIndex));
}
```

Getting a list of paper sources

The paper sources (trays) for a specific printer can be enumerated using the **GetPaperSourceCount** and **GetPaperSourceName** functions.

This example retrieves the paper source list for the default printer:

```
int paperSourceCount = printManager.GetPaperSourceCount(printerName);
List<string> paperSources = new List<string>();
string printerName = printManager.GetDefaultPrinterName();
for (int paperSourceIndex = 0; paperSourceIndex < paperSourceCount; paperSourceIndex++)
{
    paperSources.Add(printManager.GetPaperSourceName(printerName), paperSourceIndex));
}
```

Getting a list of paper sizes

The paper sizes for a specific printer can be enumerated using the **GetPaperSizeCount**, **GetPaperSizeWidth** and **GetPaperSizeHeight** functions.

This example retrieves the page size list for the default printer:

```
int paperSizeCount = printManager.GetPaperSizeCount(printerName);
List<string> paperSizes = new List<string>();
string printerName = printManager.GetDefaultPrinterName();
for (int paperSizeIndex = 0; paperSizeIndex < paperSizeCount; paperSizeIndex++)
{
    double paperWidth = printManager.GetPaperSizeWidth(printerName), paperSizeIndex);
    double paperHeight = printManager.GetPaperHeight(printerName), paperSizeIndex);
    paperSizes.Add(paperWidth + " x " + paperHeight);
}
```

Directly using the C interface

If you are using a programming language such as C++ or Delphi, you can connect to the DLL's C interface. This is a little bit more involved compared to the C# interface class where all the complexity is handled for you.

If you are using dynamic linking, use the **LoadLibraryEx** WinAPI function with the **LOAD_LIBRARY_SEARCH_DEFAULT_DIRS** and **LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR** flags. Make sure that an absolute path is used, not a relative path, using a function such as **PathCanonicalize** if necessary.

If you are using C++ with Visual Studio, you can use the **fsdk_printmanager_win32.lib** or **fsdk_printmanager_win64.lib** to link the appropriate DLL to your project.

All functions are exported from the DLL with the **FXPM_** prefix.

Strings are in UTF-8 format. A copy of all data retrieved from the library should be made before the next function call.

All strings and data sent to the library will be copied to internal storage.

The first step would be to create an instance of the print manager, using the **NewPrintManager** function:

```
int managerID = FXPM_NewPrintManager();
```

This managerID value must then be used as the first parameter for all the following function calls. For example, to initialize the library:

```
int managerID = FXPM_NewPrintManager();
std::string sn = "... Foxit PDF SDK serial number here ...";
std::string key = "... Foxit PDF SDK license key here ...";
int initResult = FXPM_InitPrintManager(managerID, sn.c_str(), key.c_str());
```

When there is no need to use print manager any more, please call the FXPM_ReleasePrintManager function to release it.