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
- C# interface class provided for .NET Core and .NET Framework

How is Foxit Print Manager distributed?

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

lib\x86_vc15\fsdk_printmanager_dotnet.dll	The 32-bit build of the Foxit Print Manager .NET DLL
lib\x64_vc15\fsdk_printmanager_dotnet.dll	The 64-bit build of the Foxit Print Manager .NET DLL
lib\x86_vc15\fsdk_dotnet.dll	The 32-bit build of the Foxit PDF SDK .NET DLL
lib\x64_vc15\fsdk_dotnet.dll	The 64-bit build of the Foxit PDF SDK .NET DLL
lib\x86_vc15\fsdk.dll	The 32-bit build of the Foxit PDF SDK DLL
lib\x64_vc15\fsdk.dll	The 64-bit build of the Foxit PDF SDK DLL
doc\print_manager\Foxit PDF SDK Print Manager .NET API Reference.pdf	The function reference for the .NET interface
doc\print_manager\Foxit PDF SDK Print Manager Developer Guide.pdf	The PDF version of the developer guide
examples\extensions_demo\printmanager	Demo projects for Visual Studio

Deployment

All the DLLs must be deployed into the same directory.

Using the C# interface for .NET

Simply add a reference to the appropriate class library DLL **lib\x86_vc15\fsdk_printmanager_dotnet.dll** or **lib\x64_vc15\fsdk_printmanager_dotnet.dll** to your project and create a new instance of the **foxitprintmanager.PrintManager** class.

```
using foxit.printmanager;
var printManager = new PrintManager();
```

Initializing the print manager

Use the Init 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.

```
string sn = "... Foxit PDF SDK serial number here ...";
string key = "... Foxit PDF SDK license key here ...";
if (printManager.Init(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 **PrintManager.NewJob** function. This will return an instance of the PrintJob class to providing all the print job related functions.

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

When the job is ready to start, call the PrintJob.Begin function and printing will begin.

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

```
PrintJob job = printManager.NewJob();
job.AddPDFFromFile("C:\\docs\\file1.pdf", "");
job.AddPDFFromFile("C:\\docs\\file2.pdf", "");
job.AddPDFFromFile("C:\\docs\\file3.pdf", "");
int duplexMode = 1; // Vertical duplex
job.SetDuplex(duplexMode);
job.Begin();
```

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);
}
```