Converting Documents on a Windows Service or ASP.NET Application using PDFNet SDK

Background

This article discusses the issue on converting documents in a Windows Service or ASP.NET application. It will start by describing the nature of the conversion method of PDFNet SDK (pdftron.PDF.Convert.ToPdf). Next, the article will discuss about Microsoft Office document conversions, where support for OLE interop automation was added. Suggestions how to create a Windows Service or ASP.NET application using conversion APIs will be described afterwards.

The ToPdf Method

The pdftron.PDF.Convert.ToPdf (ToPdf for short) method (or, relatedly, ToXod or ToXps), converts multiple file formats to the requested file output format. It supports several formats like docx, xlsx, rtf, txt, html, pub, etc. In order to convert these file formats, ToPdf method uses PDFNet printer to request a print job and the print output (in XPS) will be used for the actual conversion process.

It is important to keep in mind that in order to use the PDFNet printer, there must be an application installed on the machine which will send print jobs to the PDFNet printer. The PDFNet printer works like any other printers. In order to print a document, an application must send printing jobs to the printer first. The PDFNet printer cannot open documents by itself. For example, printing a text file requires a text file viewer (in Windows, this is normally notepad.exe) to perform a printing task.

ToPdf automates this process by issuing the print command on a document which can be opened by an installed application. If the application does not support printing of the opened document, then the Convert.ToPdf method will fail.

Note: It is important to understand that in a 64-bit operating system, the 64-bit PDFNet printer driver must be installed. Installing the 32-bit PDFNet printer in a 64-bit operating system will not work. To install the 64-bit PDFNet printer driver, use the 64-bit version of PDFNet.dll and invoke the following method in a .NET application:

```
if (!pdftron.PDF.Convert.Printer.IsInstalled())
    pdftron.PDF.Convert.Printer.Install();
```

Microsoft Office Document Conversions

If Microsoft Office 2007 SP2 or later is installed, ToPdf method will take advantage of Microsoft Office's OLE interop automation library to convert Microsoft Office documents to PDF or XPS formats. Using Microsoft Office guarantees high quality PDF or XPS output files.

The PDFNet printer will not be used when a document will be converted using Microsoft Office's interop libraries. However; the converter will use the PDFNet printer if an earlier version of Microsoft Office is installed (any versions prior to 2007 SP2). This is because the SaveAsPDFandXPS extension is not available for these versions of Microsoft Office.

Windows Service

Windows Service applications are typically run unattended without any form of user interaction. By default, Windows Service applications are using the "LOCAL SYSTEM" or "LOCAL SERVICE" account. These accounts have very limited amount of permissions to perform tasks that requires registry access, and access to some system folders. Due to their nature of being unattended, it is understandable that these applications be limited in privileges in case of a system compromise or similar kinds of issues.

When using Microsoft Office Excel, before the application actually opens, the desktop folder of the user must be present. For example, when user "Bob" opens Microsoft Office Excel, Excel requires the folder "C:\Users\Bob\Desktop" to be both present and accessible before the actual Excel windows will show in Bob's screen. Similarly, for Windows Service, Excel expects "LOCAL SYSTEM's" or "LOCAL SERVICE's" desktop folder to be present and accessible before the process will continue.

Local System or Local Service's desktop folders can be found in the following locations:

1. C:\Windows\System32\config\systemprofile\Desktop
2. C:\Windows\SysWOW64\config\systemprofile\Desktop (only on 64-bit Windows)

In order for ToPdf to function with "LOCAL SYSTEM" or "LOCAL SERVICE", the desktop folders above must first be created. It is also important to make sure that these system accounts have access to the desktop folders. It is normally not necessary to set the permissions explicitly because the Desktop will inherit them from the systemprofile folder.

Another thing to keep in mind when using 64-bit Windows: 32-bit applications use the C:\Windows\SysWOW64 folders, while 64-bit applications use C:\Windows\System32. If a 64-bit Microsoft Office version is installed on a 64-bit Windows, the required desktop folder will be: C:\Windows\System32\config\systemprofile\Desktop. And if a 32-bit Microsoft Office version is installed on a 64-bit Windows, the C:\Windows\SysWOW64\config\systemprofile\Desktop will be required instead.

If using system accounts does not work with ToPdf even if the above desktop folders are created, an alternative would be to use a user account for the Windows Service instead. There may be some cases where the PDFTron PDFNet printer will not work with system accounts because the printer requires access to the registry. In such cases, user accounts normally worked better.

ASP.NET Application Development

In an ASP.NET application, the IIS Web Server assigns an application pool under which this application will run on. This application pool has associated settings regarding the execution environment like user, .NET framework version, etc. If an ASP.NET application is not developed with these application pool settings in mind, then this ASP.NET application will fail to run. Basically, these application pool settings determine how an ASP.NET application will be executed.

By default, many ASP.NET applications run under the "NETWORK SERVICE" identity (in other Windows environment, applications run under "ASPNET" by default) as assigned through the application pool context. This default identity has very limited permissions. Some of which include, but not limited to, restricted access to file system, limited permissions to access or modify the Windows registry, cannot properly spawn a new process, etc. These limited permissions to Windows resources under this identity are important for ASP.NET applications in order to ensure security.

Having a very limited access to Windows resources, the ToPdf will fail on an ASP.NET application if the application pool's identity is set to "NETWORK SERVICE" or lower. It is a known issue that Microsoft Office Interop assemblies will not work under this identity. In most cases, when ToPdf is used under "NETWORK SERVICE" identity, the web application will show an error message about retrieving COM class factory failed. The simplest explanation for this error is that COM objects can only be activated by the following accounts:

- Administrator
- System
- Interactive

Using the PDFNet printer will also fail under the "NETWORK SERVICE" identity. This is because in order to print a document, the document will have to be opened first by an external application which will send the print job. Opening this document will spawn a new process. This new process will be the default application to open the document. As mentioned earlier, "NETWORK SERVICE" has limited permissions when spawning a new process. In most cases, many applications require a graphical interface to complete its initialization before execution continues. One example of which is notepad.exe. With these applications, if the graphical interface cannot be loaded, then it will look like the process appears to be stuck - even though there is an entry of this application in the process list where it seems like it is waiting for something. This is the case when attempting to spawn a new process under "NETWORK SERVICE".

Using the "LOCAL SYSTEM" identity

In most cases, elevating the application pool's identity to "LOCAL SYSTEM" will allow an ASP.NET application to use the ToPdf method with no issues. It is important to keep in mind, however, that using this identity presents security risks on the web server machine. "LOCAL SYSTEM" has wider ranges of privilege access to Windows resources. It allows many resources to be accessed by the ASP.NET application assigned to the application pool. If an ASP.NET application is exploited, running under the "LOCAL SYSTEM" identity may allow the web server machine to be exploited as well.

Similarly to a Windows Service application which may use the "LOCAL SYSTEM" account, there may be additional requirements in order to get the ToPdf method to function correctly. Please see the Windows Service section of this document for more details.

Using the ASP.NET impersonation feature

ASP.NET impersonation allows ASP.NET applications to execute code under the context of the impersonated user. Using impersonation in ASP.NET allows Microsoft Office Interop to be used under the "NETWORK SERVICE" identity (for the application pool).
In order to use ASP.NET impersonation, simply make the application's web.config contain a line similar to below:

```
<configuration>
<system.web>
<identity impersonate="true" username="computer\myuser" password="********"
/>
</system.web>
</configuration>
```

Using ASP.NET impersonation, however, will not allow an ASP.NET application use the PDFNet printer. As such, under an ASP.NET impersonation, ToPdf will only work with Microsoft Office documents provided that a minimum version of Microsoft Office Suite is installed on the machine.

According to Microsoft:
"… in ASP.NET, impersonation is performed at the thread level and not at the process level. Therefore, any process that you spawn from ASP.NET will run under the context of the ASP.NET worker process (NETWORK SERVICE) and not under the impersonated context."

As mentioned earlier, the PDFNet printer in ToPdf method requires spawning a new process in order to print documents. The article below describes how to spawn a process with ASP.NET impersonation:
http://support.microsoft.com/kb/889251/en-us

As long as the created process is run under accounts which have proper permissions, the PDFNet printer should be able to convert documents properly.

Isolating the Conversion Process

Perhaps the best solution to introduce a conversion feature in an ASP.NET project is to have an isolated application which will be responsible for doing the conversions. Having a separate application will ensure that the security model of the ASP.NET application remains intact. This will also provide the converter application all the permission it requires to successfully convert a document.

This separate application can either be a Windows service listening for conversion tasks or a console application which will be started by the ASP.NET application. As long this application's permission is limited to accessing specific registry keys used for printing and allowing process creation, all the conversion processes of ToPdf should work.

Related links:

http://msdn.microsoft.com/en-us/library/ms686005%28v=vs.85%29.aspx
http://msdn.microsoft.com/en-us/library/xh507fc5%28v=VS.100%29.aspx