Volume

4

FLEXCEL STUDIO FOR .NET

TMS Software

Exporting Excel files to PDF

Table of Contents

Introduction

FlexCel comes with a full PDF writer that allows you to natively export Excel files to PDF, without needing to have Acrobat or Excel installed. While the output will not be exactly the same as Excel, a lot of effort has been done to make it as similar as possible.

Creating PDF files

Introduction:

FlexCel provides two ways to create a PDF file. At higher level, you can use FlexCelPdfExport component to natively convert an xls file into pdf. At the lower level, you have the PdfWriter class, that provides a primitive API to create pdf files.

Using PdfWriter

PdfWriter is the low level option, and it was really not designed to be used directly, but to provide the methods FlexCelPdfExport needs to do its job.

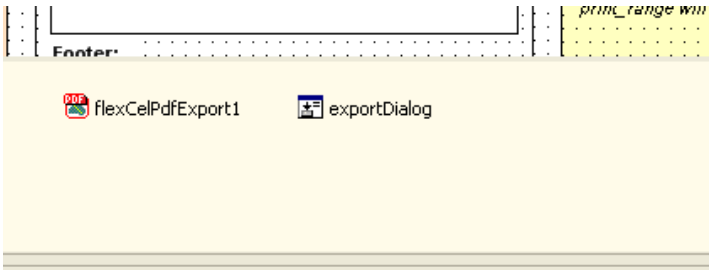
But, it can be used standalone to create a PDF file for scratch, or most likely to modify the output from FlexCelPdfExport using one of the Before/After CreatePage events.

We will not cover it in detail here since methods are documented in FlexCel.chm, but we will mention that you can find an example in how to use PdfWriter in the API Demos.

Using FlexCelPdfExport

This is the high level option, and the one you would normally use. To export an xls file to pdf, you need to:

1. Drop a FlexCelPdfExport component into a form. (Or you might create it directly by code)



2. Set the FlexCelPdfExport properties. You will find there are not a lot of properties (things like margins, printing gridlines or not, etc) and this is because all this information is read from the Excel file. If you need to change them, change the associated properties on the attached XlsFile.
3. Attach an XlsFile to the FlexCelPdfExport component, by assigning :

```
flexCelPdfExport1.Workbook = someXlsFile.
```

4. Export the file. You can use .Export to export only the active sheet, or BeginExport/ExportSheet/EndExport to export more than one.

Preparing a file for printing

Excel printing is "printer dependent", that is, it can print different things to different printers.

A rectangle printed to a standard laser printer can be 11 x 11 cm. and the same rectangle printed to a dot matrix printer might be 11.5 x 11.5 cm. As you can imagine, this might prove problematic when your file is designed to be printed in one page at 100% zoom. In some printers, it might actually print in one page, but in others it might print in two, or even in 4.

In the example above, if your file was 11x11 and you were printing in an 11x11 page, it will fit in one page. But if the printer prints a little bigger, 11.5, it will actually need 4 pages to fit.

Different from Excel, FlexCel printing is resolution independent (using GDI+) and it will print the same size everywhere. So when we calibrated the printing, we had to choose a mix of all possibilities. If Excel prints a rectangle 10cm. wide in one printer, 11 in other and 12 in other, FlexCel will print the same in all of them. We settled by emulating printing in a 600 dpi laser printer, but even this is not 100% exact. Laser printers don't print exactly the same either in Excel. In short, FlexCel results might not be the same size as Excel results for your particular printer.

So whenever you need to "fit" into a page, it is recommended that you use the "Print to Fit" setting in Excel. Look at the "Preparing for printing" section in the API guide for details on this. But if you don't want to use print to fit, make sure you leave a large enough margin at the right and at the bottom, so the page prints fine everywhere, including FlexCel.

Font Management

Due to some .NET limitations the font handling can be problematic, so here we discuss some concepts that might help with the issue.

Selecting which fonts to use

First of all, you need to know that there are two different kinds of fonts supported by FlexCel's PdfWriter.

1. **PDF internal fonts**. PDF defines 14 standard fonts that must be available to any PDF viewer, so you don't need to embed the fonts on the PDF document. They will always be supported by the PDF viewer.

   Those fonts include a Serif (Times New Roman-like), a Monospace (Courier-like) and a Sans Serif (Arial-like) alternatives, on four variants each (regular, bold, italic and bold italic) and two Symbol fonts.

2. **True Type fonts**. Those are standard Windows fonts.

   When exporting to PDF, you can chose between three different ways to handle fonts, depending on the value you set on the FontMapping property:

   a. **ReplaceAllFonts**. This will replace all fonts on the xls file to the most similar ones on the 14 standard fonts. This way you get the minimum file size and the maximum portability, but the exported PDF file might not look exactly the same, as you will lose all fancy fonts.

   b. **ReplaceStandardFonts**. This is a compromise solution. It will only replace Arial, Times New Roman and Courier for the standard fonts, and use True type for all the others. You will get a bigger PDF file (if you embed the true type fonts), but it will look as the xls file.

   c. **DoNotReplaceFonts**. This will only use true type fonts. It will be the one that better matches the xls file, but it will be a lot larger (if you embed the true type fonts) or might not look good when the user does not have the fonts you used installed (if you don't embed them)

Also, you can choose whether to embed the True Type fonts or not. If you embed them, the file will be bigger, but also will render well when the user does not have the fonts on his machine. To avoid issues, it is normally recommended that you embed all fonts.

Note that if you use Unicode characters, the fonts will always be embedded no matter which embed option you choose, because this is needed to ensure the Unicode mapping will remain correct.

Accessing True Type data.

There is limitation .NET framework because there is no way to access the internal font data for a true type font. This data is needed to embed the font and also access font metrics not available on the framework.

There were two alternatives for us to take:

1. We could have used unmanaged code to call GetFontData on the Win32 API and get the actual data. This option was discarded because we want FlexCel to remain 100% managed. Staying 100% managed allows it to run on 64 bits, mono and everywhere, and it made no sense to go unmanaged just for one missing method.

2. We could read the font files (*.ttf) directly. This option is better, but is not perfect either. The problem here is that the .NET framework gives the standard folder for lots of system folders, but not for the fonts folder. Again, to access the system fonts folder (where *.ttf are) we would need to go unmanaged. (Note: In .NET 4.5 there is finally a way to find the "Fonts" folder and we use it. But you might still want to use your private font folder, so the rest of this section still applies).

As going unmanaged is out of question (for us), we adopted a compromise solution. We use solution 2) and assume fonts are on the

<System Folder>\..\Fonts (On Windows). This is true for all installations of Windows we are aware of, but it might not be true for some or in the future, or in mono and Unix.

If <System Folder> is empty, then we assume we are running on Linux, and we will try "**/usr/X11R6/lib/X11/fonts/truetype**".

If this folder does not exist, FlexCel will search on "**<folder where FlexCel.dll is>/Fonts**". If you are running Linux, you might create a ***symbolic link*** to the place where the fonts are installed. (see the MONO section below for more information)

As a last resort, FlexCel provides a **OnGetFontFolder** event that allows you to specify where fonts are stored on your system. Here you can tell FlexCel where the fonts are.

Avoid using this event if you can, since when you use it your code will not transparently run on MONO and Windows, you will have to have two different binaries. A Symbolic link from the FlexCel installation folder to the fonts folder should be a more elegant solution.

Of course, you will not face any of these problems when using only standard PDF fonts (and no Unicode)

Fonts in iOS and OSX

In MonoMac, Xamarin.Mac and Xamarin.iOS, FlexCel can access the true type directly from the Cocoa framework, so exporting to pdf should work without issues and without any extra step in those platforms. Note that in any case, the fonts available might be different from the fonts available in a Windows machine. You can get a list of fonts available in OSX here: http://en.wikipedia.org/wiki/List_of_typefaces_included_with_Mac_OS_X and in iOS here: http://iosfonts.com

Fonts in Android

At the time of this writing, in Android there are only 4 predefined fonts available for every app (Droid Sans, Droid Serif, Droid Mono and Roboto). This means that unless you want to use embedded pdf fonts, your application will have to provide its own fonts.

As you will normally provide those fonts as Assets, FlexCel has built in support for using Assets when running in Android.

By default, if you copy your fonts to the "Assets/fonts" folder FlexCel will pick them automatically, and you don't need to do anything. If you want to specify a different folder with the font Assets, you can use the "GetFontFolder" event, supplying a FontPath that starts with "@" (like "@mydata/myfonts"). If the fontPath starts with @ and you are in android, FlexCel willuse it as a path to an Asset, not as a path in the folder. Of course you can still reference a real folder in the disk by not starting the name with "@"

Going Unmanaged

By the way, note that we said going unmanaged is out of question *for us*. If your application already uses unmanaged code, you can use the **GetFontData** event on FlexCelPdfExport to call GetFontData on the Win32 API, and return the font information to FlexCel. This way you will avoid scanning the font folder completely, and it can speed up a little PDF export.

While we do not recommend that you go unmanaged for this since scanning the font folder is fast anyway, you have all options. Do what you prefer.

For a demo on using the GetFontData event, see the ExportPdf demo.

Fonts and Excel 2007

Excel 2007 changed the default font in a new file to be "Calibri" instead of "Arial". This might bring you problems if you develop in a machine that has Excel 2007 installed, but you deploy in a server that doesn't. There are two solutions for this:

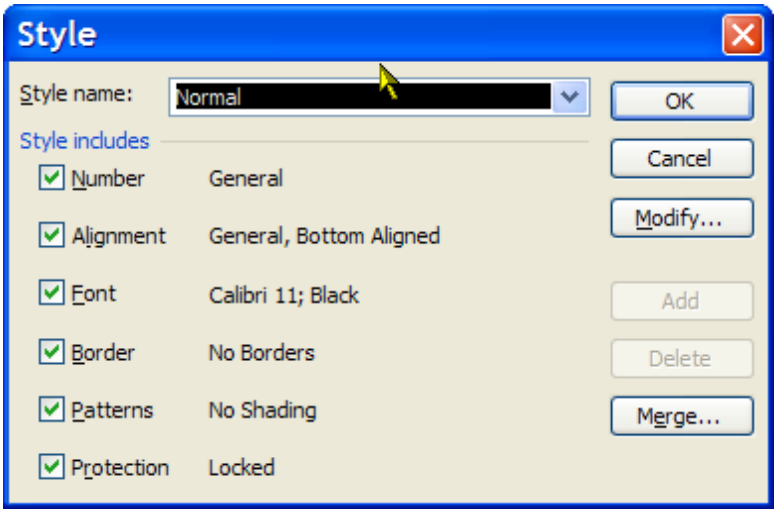
1. You can **copy the Excel 2007 fonts to the server**, and make sure you **embed the fonts in the PDF file**. Note that if you do not embed the fonts, any user who does not have Excel 2007 will not be able to see your PDF file correctly.

2. If you want maximum portability, make sure you change all fonts to Arial or Times new Roman in your template before exporting.

Setting the "Normal" font

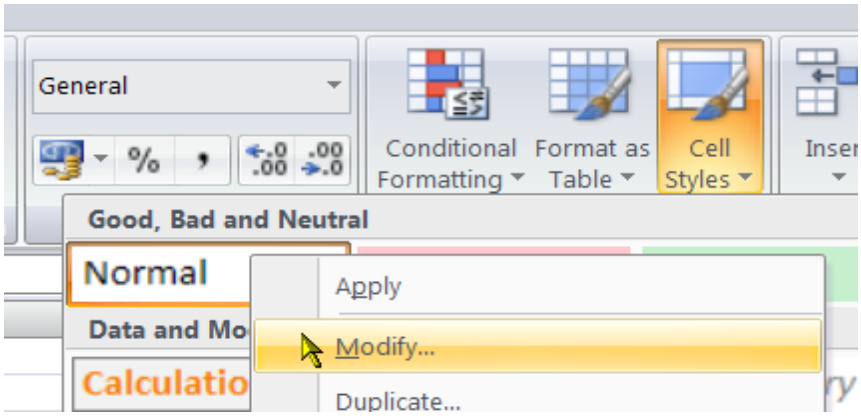
Especially important when changing the fonts is to make sure the "Normal" format uses a known font. You can see/change the normal font this way:

1. In **Excel 97-2003**: Go to "Menu->Format->Style..." You should see the following dialog:



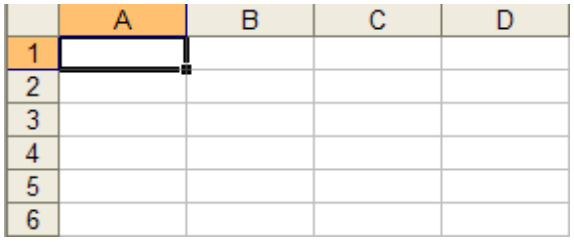
Verify that the "Normal" style has a known font like "Arial"

2. In **Excel 2007**: In the home tab in the Ribbon, select Cell Styles, right click in "Normal" and choose modify. Note that depending on your screen resolution, "Cell Styles" might show as a button or just display the "Normal" box directly in the ribbon.

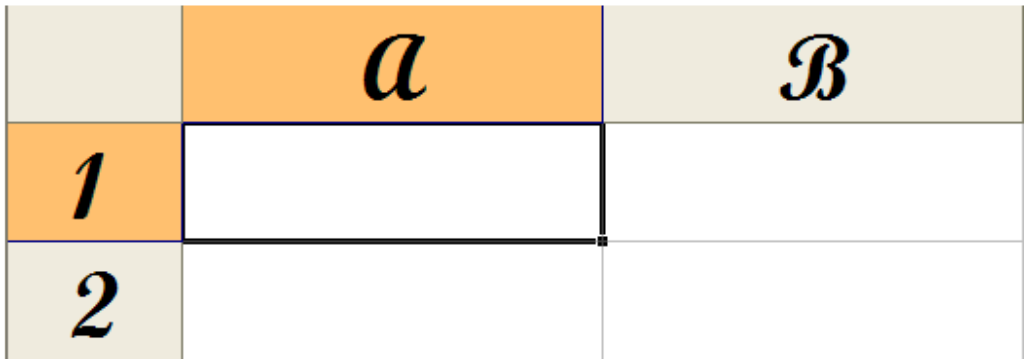


You will get a similar dialog to the one you got in Excel 2003. Again, make sure the normal style uses a font you have in your server.

The "Normal" style is used not only for empty cells, but to set the column widths. For example, this is how an empty sheet looks with "Normal" style using Arial 10:



And this is how it looks using Script MT 36:



As you can see, the font used in the "Normal" style is used to draw the headings "A", "B", "1", etc., and even more important, it is used to calculate the column width. Column with is measured as a percentage of the "0" character width in the normal font. If you change the normal font, column widths will change.

If you do not have the "Normal" font installed in your server, Windows will replace it with a substitute, and it will probably have a different width for the "0", leading to a wrong column width. So it is important that you have that font installed in your server.

Dealing with missing fonts and glyphs

There are three main font-related problems you might find when converting an xls file to PDF, and we are going to cover them in this section. The errors are non fatal, and that means that the file will be generated anyway, but it will not look as good as it could.

You can control what to do when any of these errors happen by hooking an event to the **FlexCelTrace** static class. From this event, you could write a log file when any of these errors happen, warn the user, or just raise an exception if you want to abort file generation.

Problem 1: Missing fonts

This is normally the easiest one to solve, and normally happens when deploying an application to a server. As explained in the section above, this often happens with "Calibri" font that gets installed by Excel 2007, and probably will not be installed in the server. As FlexCel needs the font to be present in order to create the pdf file, it will substitute it with a "similar" font, normally Arial or Microsoft sans serif.

This might not be an issue if there are any fonts in the system that are similar to the one being replaced, but it can be a big issue with Calibri, since that font has very different metrics from the font it gets replaced (Arial). As an example, here you can see an Excel 2007 exported to PDF in a machine that has Calibri installed an in other that doesn't:

With Calibri installed in the fonts folder:

	A	B	C	D
1				
2		This text is in Calibri	Balance of Year	Balance Sheet
3		**This text is in Arial**	**Balance**	**Balance Sheet**
4		Assets		
5		Current assets:	2007	2008
6		Cash	-	-
7		Investments	-	-

Without Calibri installed (Replaced by Arial):

	A	B	C	D
1				
2		**Calibri**	**Year**	**Balance Sheet**
3		**This text is in Arial**	**Balance**	**Balance Sheet**
4		**Assets**		
5		**Current assets:**	**2007**	**2008**
6		Cash	-	-
7		Investments	-	-
8		Inventories	-	-

As you can see in the images, Calibri is much narrower than Arial, so the text in cell B2 "This Text is in Calibri" is cut and only "Calibri" shows in the second screenshot. If you are seeing

lots of cut text in the server while the files are exported fine in your development machines, this is probably the cause.

The solution for this problem is easy; make sure you have all the fonts you use installed in your system. If you want to get notified whenever this automatic font replace happens, you can catch the "FlexCelError.PdfFontNotFound" errors in FlexCelTrace, and use it to notify the user he should install the missing fonts.

Problem 2: Missing Glyphs

This problem happens when you are using a font that doesn't contain the character you want to display. If you for example write

```
"日本 に行きたい。"
```

inside a cell and keep the font "Arial", you will see the correct characters in Excel, but when exporting you might see blank squares like this:

```
□□ □□□□□□
```

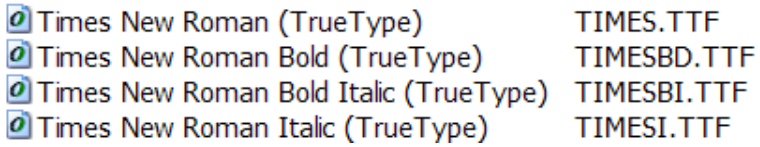
The reason for this is that "Arial" doesn't actually contain Japanese characters, and Excel is "under the hood" using other font (normally MS Mincho) to display the characters. To emulate this behavior, FlexCel provides a "FallbackFonts" property, where you can enter a list of fonts to try if the font that was supposed to be used doesn't have the character. If no font in the FallbackFont chain contains the glyph, you will see a blank square.

The solution in this case is to use fonts that actually have the characters you want to display, or ensure that some fonts in the FallbackFonts properties have them. By default FlexCel uses "Arial Unicode MT" as a fallback font, but you can add as many others as you need.

If you want to get notified when this happens so you can warn the user to change the fonts, you can catch the "FlexCelError.PdfGlyphNotInFont" and "FlexCelError.PdfUsedFallbackFont" errors in FlexCelTrace.

Problem 3: Faux Italics and Bolds

The last problem is related to fonts that don't have a specific "Italic" or "Bold" variant. Normally, a quality font comes with four files including four variants of the font: Bold, Italic and BoldItalic. If you look in your font folder, you will see things like this:



That is, a different file for each font variant. If the font comes with only one "Normal" file, the italics can be faked by the system by slanting the characters, and bold can be faked by adding some weight to the characters. But of course this leads to low quality results, and should be avoided whenever possible.

There is other problem with "fake" Italics and Bold, and that is that Acrobat will not show them when you embed the fonts.

The solution to this problem is to use fonts that include the variants you need.

Normally this is not a problem, since all quality fonts designed to be used with Italics and Bold already come with files for those variants. But if absolutely you need to use fonts that don't come with the variants, you might at least not embed the fonts, so the "fake" style will show.

To be notified whenever FlexCel finds a "fake" style, you can use the "FlexCelError.PdfFauxBoldOrItalics" notifications in FlexCelTrace.

Accessibility of the generated files

Setting a natural language

It is important to set a natural language for the document if you know it. This way a screen reader or a text-to-speech engine will be able to correctly read the text out loud.

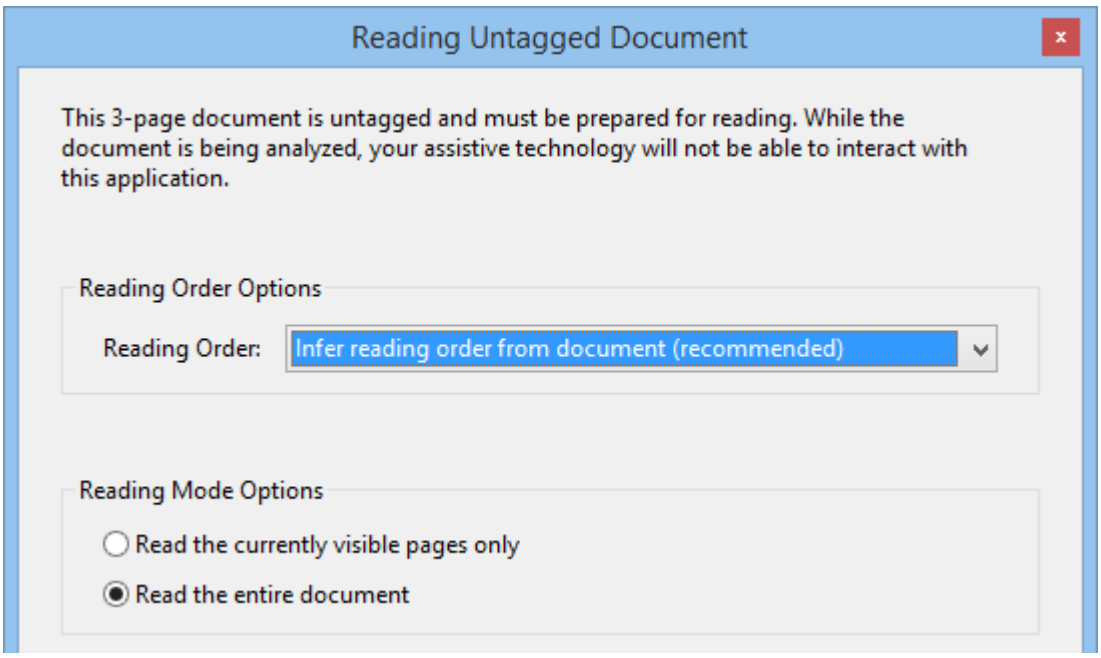
To set up the language in FlexCel, use code like this:

```
using (FlexCelPdfExport pdf = new FlexCelPdfExport(xls))
{
    pdf.Properties.Language = "es-es";
    …
```

Tagging the files

FlexCel allows to create Tagged PDFs, which contain extra information about the document structure (like for example what are the cells in the table). This information allows a screen reader to know the correct order to read the text.

As it is an important accessibility feature, since FlexCel 6.5 files are tagged by default. You must explicitly turn tagging off in order to get untagged pdfs. Note that tagged pdfs are bigger than normal ones, so this might be a reason to turn it off. But also note that newer Acrobat versions will display a dialog when opening untagged pdf files:



Creating PDF/A files

PDF/A files are files designed specifically for archiving. FlexCel has full support for the variations of the standard: PDF/A1a, PDF/A1b, PDF/A2a, PDF/A2a, PDF/A3a and PDF/A3b.

If you need to choose a version, we would recommend PDF/A2 or PDF/A3. PDF/A1 is a little too restrictive, and lacks some features that FlexCel could use to generate better files: It doesn't support transparency and it doesn't allow compressing the tags in the document. Due to the lack of transparency, if you have any transparent image in your file it might look wrong. Due to the lack of tag compression, files will be much bigger than PDF/A2.

In order to create PDFA files, you need to set PdfWriter or FlexCelPdfExport property **PdfType** to the correct version. For example:

```
pdf.PdfType = TPdfType.PDFA1
```

Then you need to choose if you want to generate "a" (PDF/A1**a**, PDF/A2**a**, PDF/A3**a**) or "b" (PDF/A1**b**, PDF/A2**b**, PDF/A3**b**) files. "a" files are the most complete, and they require you to tag the file.

When using FlexCelPdfExport, you would just set the correct option by changing the **TagMode** property:

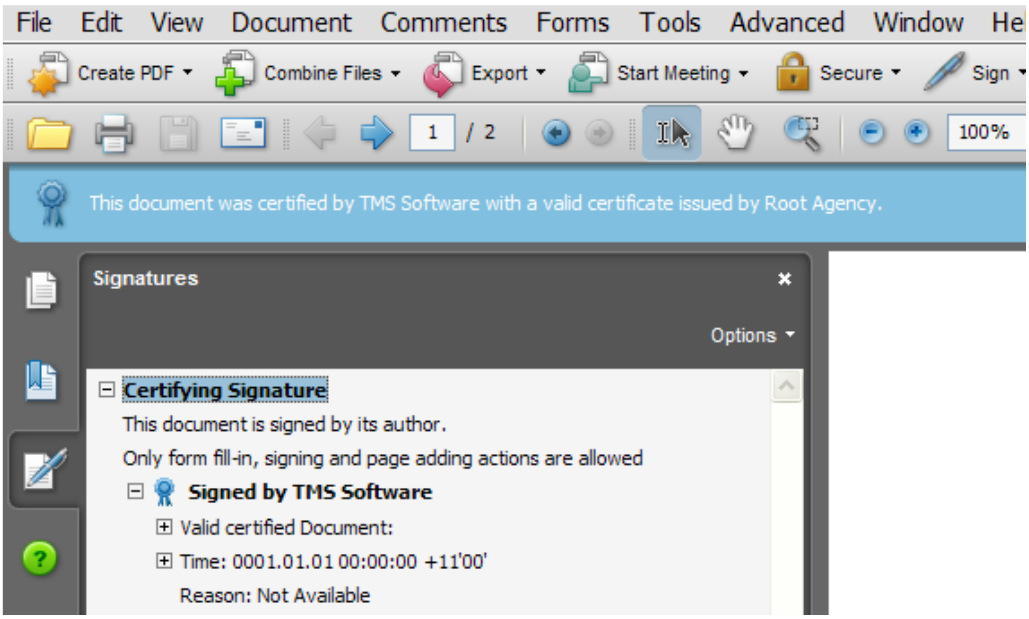
```
pdf.TagMode = TTagMode.None;  //Generates "b" files
```

As the TagMode is Full by default, FlexCel by default generates "a" files.

When using PdfWriter, you need to manually tag the files, as FlexCel can't know the structure from the drawing commands. You need to use the methods `TagContentBegin` / `TagContentEnd` to specify the blocks of text you want to tag, and then set the `TagActions` property to specify how that tagged content relates to the structure of the file. Tagging in PdfWriter is an advanced topic outside the scope of this document. Due to the way PdfWriter is designed, it won't keep tags in memory and you need to write them directly to the file as you are creating it.

Signing PDF Files

FlexCel allows you to sign your PDF files with a certificate, so any change to the file will invalidate it. This is how a signature looks like in Acrobat 8:



In Acrobat 7 you will see the same sidebar at the left, but there is no blue bar at the top.

Important Note: FlexCel signing algorithm is supported only in Acrobat 7 or newer, it will not work in Acrobat 6 or 5 since those versions do not have the required support. Because of this reason, the header in the generated PDF file will automatically switch to say that the file is "Adobe 7 or newer" compatible when you include a signature in your files. If there are no signatures, the default header specifies the file is "Acrobat 5 or newer" compatible.

It is also worth noting that users will still be able to see the generated files in Acrobat 5 or 6, but they will get a warning when opening them and the signature will not validate.

Concepts of signing are outside the scope of this document, but you can find a lot of information in signing in the Acrobat documentation or just in Internet. A good place to start might be:

http://msdn.microsoft.com/msdnmag/issues/07/03/NETSecurity

Customizing the Signing Engine

FlexCel comes with a built-in signing implementation, but it allows you to change it by your own. There is mainly one reason you might want to do that:

> The built-in engine is not optimal. It uses the standard .NET PKCS classes, and those classes do not allow for incremental signing. This means that the whole PDF document must be kept in memory so we can pass all the contents as a byte array to the signing method in .NET. If you have a library that supports incremental signing, that is, each time a small group of bytes is written you recalculate the hash instead of calculating the whole hash at the end, you can save memory by creating your own engine. **Just make sure you need the extra performance** before doing so, because normally the built in implementation is good enough, and has the advantage of being 100% managed and built-in in the framework.

If you decide to create your own Signer class, you need to implement two simple abstract classes:

1. TPdfSigner:

This is the class that implements the signing. You need to override three methods:

Write, **GetSignature** and **EstimateLength**. The first method is called each time some bytes are written to the file. You should use them to calculate a PKCS7 signature with them. If you can't calculate the signature incrementally you will need to buffer them and calculate it when GetSignature is called. The second method, GetSignature, is called only once at the end of the pdf file and it should return the PKCS encoded signature as an array of bytes. The third method must return the length of the byte array that will be returned by GetSignature or a bigger number, but never smaller. Note that this third method will be called before the signature is computed so you might need to estimate the length.

2. TPdfSignerFactory:

This class is really simple, and it just should return an instance of the particular TPdfSigner child you created in 1).

When creating your own classes, it might be helpful to look at the TBuiltInSigner implementation on file Signatures.cs. Also look at PdfExport demo.

Export to PDF and FlexCel recalculation

When you create a report with FlexCel most formulas are recalculated, but some are not. This is not an issue when opening the xls file on Excel, as Excel will recalculate the whole file again, but will be an issue when exporting the xls file directly to PDF.

FlexCel implements over 200 Excel functions, and most used formulas are all there so you should not experience big issues. But you need to make sure you do not use any not implemented function to get a properly recalculated sheet. You can find a list of currently implemented functions on the file SupportedFunctions.xls. All functions mentioned there are safe to use.

FlexCel comes with a little utility, the demo "**Validate Recalc**" that will allow you to check if all the formulas on an xls file are ok to use. And of course you can use the code on this demo inside your own application to tell your users when they use a not supported formula.

Using FlexCelPdfExport on MONO (linux)

Mono is an incredible platform to deploy your .NET applications, and as FlexCel.NET is 100% managed code, most of the time running a FlexCel.NET application on MONO means just to copying the files to a linux server.

But in the case of Pdf export there is one issue that you need to consider:

> As explained before on this document, in Linux the True-Type fonts are not installed on "\Windows\Fonts", and might be on "/usr/X11R6/lib/X11/fonts/truetype" (this folder might change with the Linux distribution) . You can find where the ttf files are by writing
>
> ```
> find / -name *.ttf
> ```
>
> on a terminal window.

Note that FlexCel will search for files with a ttf extension in lowercase. Fonts in Windows might be uppercase (i.e. SYSTEM.TTF), and you need to rename them (i.e. SYSTEM.ttf).

If your ttf fonts are not on "/usr/X11R6/lib/X11/fonts/truetype", you should make a symbolic link between the place where flexcel.dll is and the Font folder. For example,

```
ln -s /font_folder Path_Where_FlexCel.dll_is/Fonts
```

If you prefer, instead of creating a symbolic link you can create an OnGetFontFolder event. The drawback of this is that the folder will be hardcoded on the application, and it will not be so easy to move it from a machine to other.

By the way, you will probably want to install the core Windows true type fonts on Linux:

http://corefonts.sourceforge.net/

Your Linux distribution might have those core fonts available as RPM too.