Flexcel API
Developers Guide

Documentation: May, 2010

Web: http://www.tmssoftware.com
Email : info@tmssoftware.com

Table of contents

Introduction

The FlexCel API (Application Programmer Interface) is what you use to read or write Excel files on a low level way. To create Excel file with a template, use FlexCelReport.

Basic concepts

Before starting writing code, there are some basic concepts you should be familiar with. Mastering them will make things much easier in the future.

Arrays

To maintain our syntax compatible with Excel OLE automation, most FlexCel indexes/arrays are 1-based.

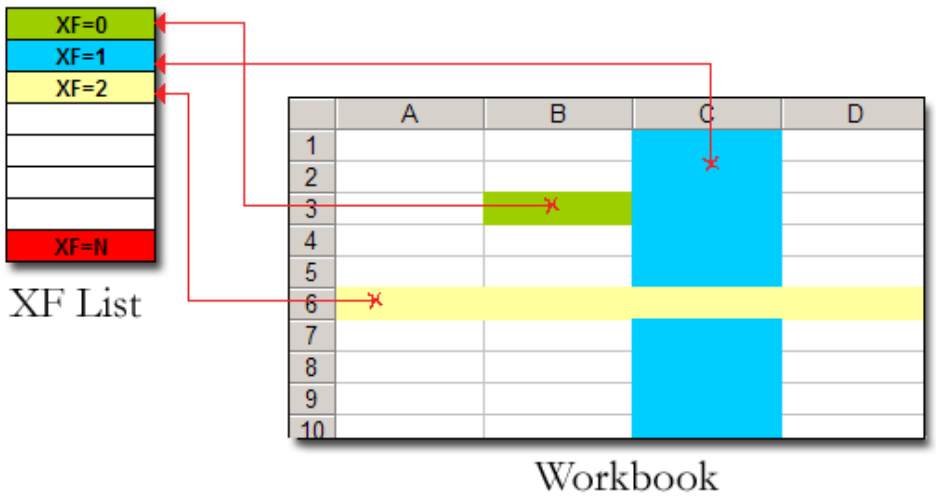
That is, cell A1 is (1,1) and not (0,0). To set the first sheet as ActiveSheet, you would write ActiveSheet:=1 and not ActiveSheet:=0.

So, in C++ loops should read: “for (int i=1;i<=Count;i++)” and in Delphi they should be like “for i:=1 to Count”

The two exceptions to this rule are XF and Font indexes, that are 0 based because they are so on Excel.

Cell Formats

All formats (colors, fonts, borders, etc) on an Excel workbook are stored into a list, and referred by number. This number is known as the XF (eXtended Format) index. A simple example follows:



Here Cell B3 has XF=0 and the XF definition for the background color is green. Row 6 has XF=2, so all the empty cells on row 6 are yellow. Column C has XF=1, so all the empty cells on column C that do not have a Row format are Blue.

Most methods at FlexCel import return a **XF** index, and then you have to look at the XF list (using the **GetFormat** method) to get a class encapsulating the real format. There are two helper methods, **GetCellFormatDef** and **GetCellVisibleFormatDef** that obtain the XF index and return the format class in one step.

To Create new formats, you have to use the AddFormat method. Once you get the Id of the new XF, you can use it as you wish.

Also, you don't have to worry also on inserting a format 2 times, if it already exists, AddFormat will return the existing id and not add a new XF entry.

Font Indexes

The same way we have an XF list were we store the formats for global use, there is a Font list were fonts are stored to be used by XFs. You normally don't need to worry about the FONT list because inserting on this list is automatically handled for you when you define an XF format. But, if you want to, you can for example change Font number 7 to be 15 points, and all XFs referencing Font 7 will automatically change to 15 points.

Palette colors

Colors in Excel are referred by a palette index. For example, you can have color 3 = rgb(123,134,188)

To get the real RGB color from a palette index, you have to use ColorPalette property. You can also modify the palette to fit your needs, by assigning this property.

There is a handy function for converting an RGB value to the nearest palette index, this is **MatchNearestColorIndex**.

Automatic Colors

Besides the normal colors on the palette, Excel lets you set colors to Automatic. This will be returned as a <=0 index, or bigger than the max color palette entry, depending on Excel. You can use GetColorPalette(colorIndex, automaticColor) to get the real rgb color. Automatic colors depend on the case, but are normally black for the foregrounds and white for the backgrounds.

Date Cells

As you might already know, there is no DATE datatype in Excel.

Dates are saved as a double floating number where the integer part is the number of days that have passed from 1/1/1900, and the fractional part is corresponding fraction of the day. For example, the number 2.75 stands for "02/01/1900 06:00:00 p.m." You can see this easily at Excel by entering a number on a cell and then changing the cell format to a date, or changing the cell format of a date back to a number.

The good news is that you can normally convert directly from/to Delphi/Excel automatically since TDateTime is actually a double. That is, if you enter

XlsFile.CellValue[1,1] := now, and the cell (1,1) has date format, you will write the actual value of "now" to the sheet. Note that FlexCel will automatically take in account if you are using 1904 date mode (see below) and enter the correct date into the cell.

The bad news is that you have no way to know if a cell has a number or a date just by looking at its value. If you enter a date value into a cell and then read it back, you will get a double. So you have

to look at the format of the cell. There is a helper function, XlsFormatValue1904 that can help you on this issue.

There is also a **"1904"** date mode, where dates begin at 1904 and not 1900. This is used on **Mac Excel**, but you can change this option in Excel for Windows too. FlexCel completely supports 1900 and 1904 dates, but you need to be careful when converting dates to numbers and back.

When FlexCelImport.Option1904 is true, you can't cast a TDateTime to a double as they don't have the same values. You need to use the helper functions "ToOADate" and "FromOADate" in FlexCelImport to do the conversion.

Copying and pasting in BIFF8

XlsFile has a group of methods allowing you to copy/paste from/to FlexCel to/from Excel in native Excel format. All methods copy and paste the data on BIFF8 and Tabbed-Text format, to allow for copying/pasting from other sources besides Excel.

Copying and pasting in native BIFF8 format is a great advance over copying/pasting on plain text only. It allows you to keep cell formats/colors/rounding decimals/merged cells/etc. But it has its limitations too:

- It can't copy/paste images
- It can't copy/paste strings longer than 255 characters
- It can't copy the data on multiple sheets.

We would like to say that these limitations are not FlexCel's fault. The BIFF8 specification is correctly implemented; those are limitations on Excel's part.

Of course, Excel can copy and paste everything without problems, but this is because Excel doesn't use the clipboard to do the operation. If you close all instances of Excel, open a Worksheet, copy some cells to the clipboard, close Excel and open it again you will run into the same limitations. Copy/paste limitations on Excel don't show when it is kept in memory.

Reading And Writing Files

The native FlexCel engine is encapsulated on the class FlexCelImport. This class stores an Excel file in memory, and has methods allowing loading a new file into it, modifying it, or saving its contents to a file or a stream.

Important: Even when a FlexCel object is fully managed and you don't need to dispose it, keep in mind that it stores a full spreadsheet in memory. Do not leave global XlsFile objects hanging around when you don't need them because they can use a lot of memory. Just set them to null when you are done using them, or use local objects.

Opening and saving files

The easiest way to process a file is to drop a FlexCelImport and an XlsAdapter into a form or datamodule, set the Adapter property of FlexCelImport to the XlsAdapter component, and write the following code:

```
procedure TDataModule.ButtonOpenClick(Sender: TObject);
begin
  FlexCelImport.OpenFile('c:\test.xls');
  try
    DoSomething(FlexCelImport);
    FlexCelImport.Save('c:\result.xls');
  finally
    FlexCelImport.CloseFile;
  end;
end;
```

Here we can note:

1.On this example FlexCelImport is a global object, so we CloseFile the spreadsheet at the end to free resources. It is not really needed, you could not Close it and it would automatically free when you load other file, but closing it will ensure you do not keep the spreadsheet in memory if you don't need it.

2. You can look at the XlsAdapter component as the "engine" powering FlexCelImport and all "high level" FlexCel components. You only need one XlsAdapter component in your application, or you can have one per form. **But you don't need a separate XlsAdapter for every FlexCelImport.**

3. By default FlexCel never overwrites an existing file. So, before saving you always have to call File.Delete, or set XlsAdapter.AllowOverWriteFiles property = true.

If you prefer not to drop components in a form, or you just don't have a form or datamodule because it is a non visual application, you can always create them with code:

```
procedure TDataModule.ButtonOpenClick(Sender: TObject);
var
  FlexCelImport: TFlexCelImport;
  Adapter: TXLSAdapter;
begin
  Adapter := TXLSAdapter.Create(nil);
  try
    Adapter.AllowOverwritingFiles := true;
    FlexCelImport := TFlexCelImport.Create(nil);
    try
      FlexCelImport.Adapter := Adapter;
      FlexCelImport.OpenFile('c:\test.xls');
      DoSomething(FlexCelImport);
      FlexCelImport.Save('c:\result.xls');
    finally
      FreeAndNil(FlexCelImport);
    end;
  finally
    FreeAndNil(Adapter);
  end;
end;
```

Note that we didn't CloseFile in this example, as freeing FlexCelImport will close the file for us. Also note that we set AllowOverwritingFiles = true in the Adapter.

Modifying files

Once you have loaded a document with **FlexCelImport.OpenFile** or created a new empty file with **FlexCelImport.NewFile** you can proceed to modify it. Add cells, formats, images, insert sheets, delete ranges, merge cells or copy ranges from one place to another. It is not on the scope of this document to show how to do this, you should look first at the available demos for the different things you can do, and then on the reference (F1 from Delphi) to learn about specific methods.

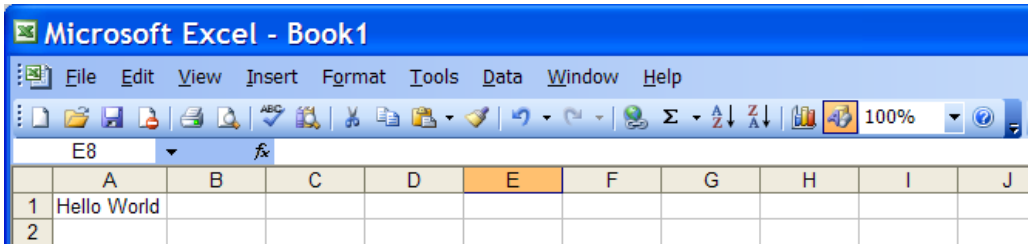
Autofitting Rows and Columns

FlexCel offers support for “autofitting” a row or a column, so it expands or shrinks depending on the data on the cells.

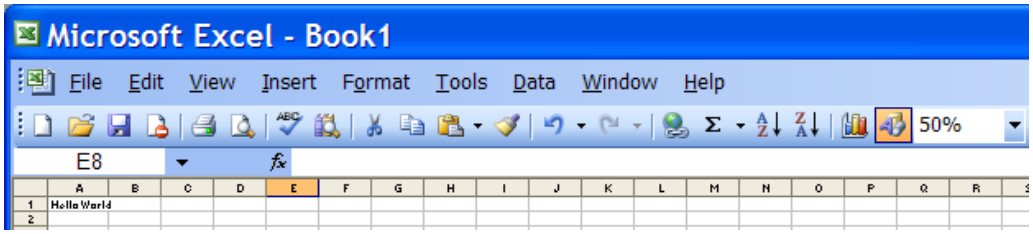
But **autofitting is not done automatically**, and we have good reasons for it to be this way.

So, to explain this better, let's try a simple example:

1) Imagine that we create a new Excel File, write “Hello world” on cell A1, and go to “Format->Column->AutoFit Selection”. We will get something like this:



2) As you see, column “A” was resized so “Hello World” fits inside. Easy, isn't it? Well, not as much as we would like it to be. Let's now change the zoom to 50%:



3) Now the text “Hello world” is using part of column “B”. We didn't changed anything except the zoom an now text does not fit anymore, in fact, you can autofit it again and column “A” will get bigger.

What happened here? The easy answer is that Excel is resolution dependent. Fonts scale in “steps”, and they look different at different resolutions. What is worse, printing also changes depending on the printer, and as a thumb rule, it is not similar at what you see on the screen.

So, what should a FlexCel autofit do? Make column A the width needed to show “Hello world” at 100% zoom, 96 dpi screen resolution? Resize column A so “Hello world” shows fine when printing? On a dot matrix printer? On a laser printer? Any answer we choose will lead us to a different column width, and there is no really “correct” answer.

As you can imagine, if we used all this space to describe the problem, is because there is not a real solution. Autofit on FlexCel will try to adapt row heights and column widths so the text prints fine from Excel on a 600 dpi laser printer, but text might not fit exactly on the screen. Autofit methods on FlexCel also provide a “Adjustment” parameter that you can use to make a bigger fit. For example, using 1.1 as adjustment, most text will display inside the cells in Excel at normal screen resolution, but when printing you might find whitespace at the border, since columns or rows are bigger than what they need to be.

And this was the reason we do not automatically autofit rows (as Excel does). Because of the mentioned differences between different resolutions, we cannot calculate exactly what Excel would

calculate. If we calculated the row height must be “149” and Excel calculated “155”, as all rows are by default autoheight, just opening an Excel file on FlexCel would change all row heights, and probably move page breaks. Due to the error accumulation, maybe on FlexCel you can enter one more row per page, and the header of the new page could land in the bottom of the previous.

The lesson, do the autofit yourself when you need to and on the rows that really need autofit(most don't). If you are using FlexCelImport, you have FlexCelImport.Autofit... methods that you can use for that.

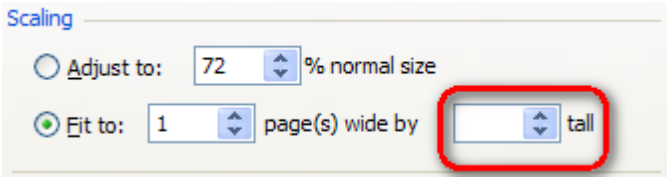
By default, Excel autofits all rows. So, when opening the file in Excel, it will re calculate row heights and show them fine. But when printing from FlexCel, make sure you autofit the rows you need, since FlexCel will not automatically do that.

Preparing for Printing

After creating a spreadsheet, one thing that can be problematic is to make it look good when printing or exporting to PDF.

Making the sheet fit in one page of width

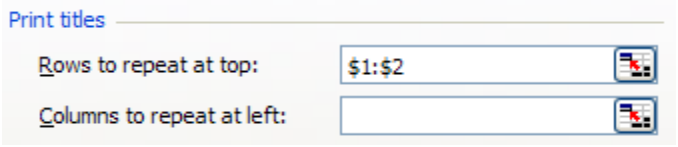
This is probably the first thing you should do when preparing most documents for printing. Go to the File Menu, select "Page Setup", and then in the "Scaling" section make it fit in one page wide:



Make the "tall" combobox empty by deleting the number on it, to allow your document have as many pages as it needs. You can do this directly in Excel when using Templates to create your documents, or you can do this in FlexCel API by setting FlexCelImport.PrintToFit = true, FlexCelImport.PrintNumberOfHorizontalPages = 1, and FlexCelImport.PrintNumberOfVerticalPages = 0.

Repeating Rows and Columns at the top

Other useful thing you can do in the "Page Setup" dialog is to setup some rows and columns to be repeated in every page. Select the "Sheet" tab and set the "Print Titles":



This way your tables can keep their headers in every page. By the way, while you are in the "Sheet" tab, You might want to look at the option to print the gridlines or the column and row headings (The "A", "B", etc. at the top and "1", "2", etc. numbers at the left of the sheet)

You can do this directly in Excel when using Templates to create your documents, or you can do this in FlexCel API by doing:

```
InitializeNamedRange(NamedRange);
NamedRange.Name := InternalNameRange_Print_Titles;
NamedRange.RangeFormula := '=1:2,A:B';

//While this normally should be 0, In the case of //Print_Titles, the
sheetindex must be specified.
NamedRange.NameSheetIndex := 1;

FlexCelImport.SetNamedRange(NamedRange);
```

To set up the rows and columns to repeat, or set XlsFile.PrintGridLines = true and XlsFile.PrintHeadings = true to set up printing of gridlines and headings.

Using Page Headers/Footers

Besides repeating rows and columns, you can also add headers and footers from the page setup dialog. One interesting feature in Excel XP or newer is the ability to include images in the page headers or footers. As those images can be transparent, you can have a lot of creative ways to repeat information in every sheet.

From FlexCel API, use the XlsFile.PageHeader and PageFooter properties to set the header and footer text. If using a template, you can just set those things in the template.

Miscellanea

Finding out what format string to use in FlxFormat.Format

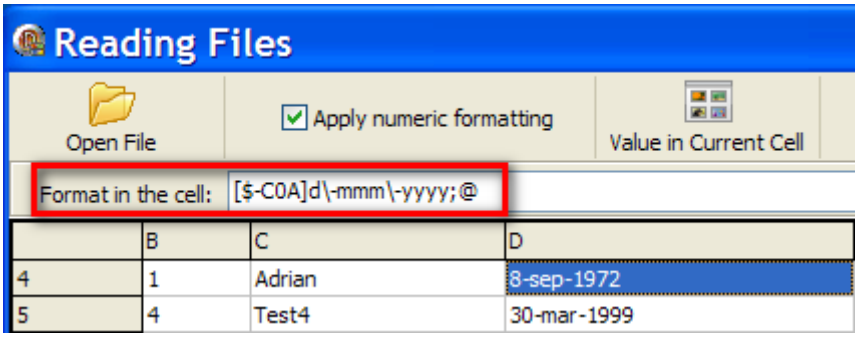
When you want to format a cell with a specific numeric format, you need to specify it in the TFlxFormat.Format property.

For example, to format a cell as Currency with two decimal places, you could use:

```
var
  fmt: TFlxFormat;
begin
  FlexCelImport.GetDefaultFormat(fmt);
  fmt.Format := '#,##0.00';
end;
```

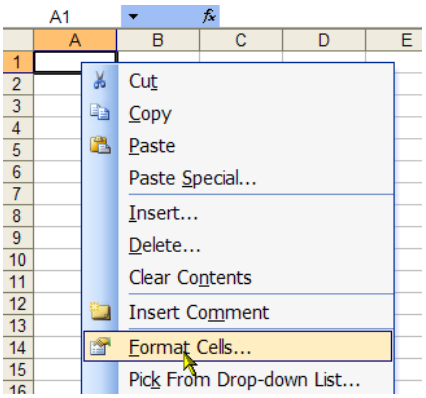
Now, how do you find out which format string to use? Format strings in FlexCel are the same used in Excel, and you can find documentation about them in Excel help, if you have doubts about them. But there are two easy ways to find out which format to use for most normal cases, and this is what we will explain now:

A. Simply use FlexCel to find out. Create a new file in Excel, format the cells as you want, open the file with FlexCel and look at the formats. You can also use the "Reading Files" demo for this, open the file with it and look at the format in the cell:

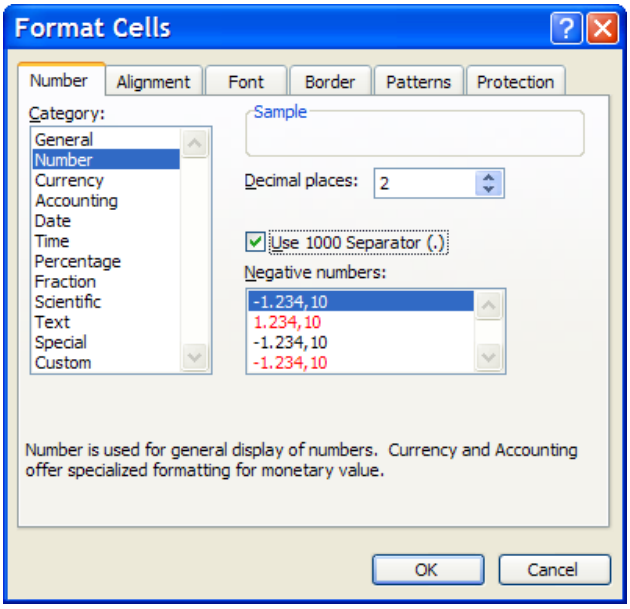


B. While method A) is normally easier, you can also find out the format directly from Excel by following the steps below:

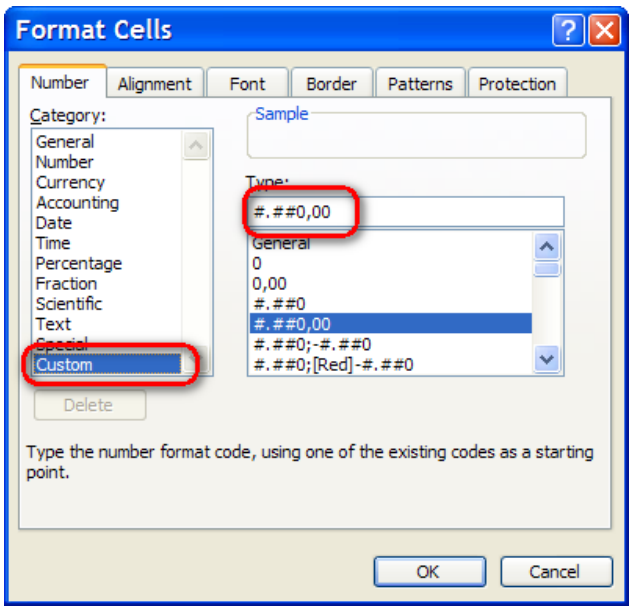
1.Open a new Excel sheet, right click a cell and select “Format Cells...”:



2. In the dialog that appears, select the format you want. In this example we will choose currency with two decimals and thousands separator, but it applies to any format.



3.Once you selected the format you want, Select “Custom” at the Category Listbox. There is no need to press OK in the dialog.



4. The string that appears in the “Type” box is the one you need to use, converted to English locale.

Important: You always need to enter the strings in ENGLISH format even if your machine is not in English. In this example we used on purpose an Spanish locale, where the thousands separator is "." and decimal "," so Excel shows "#.##0,00"

But as we need to enter the string as it would read in English, in FlexCel code we use "#,##0.00".

Other than the localization problem if your machine is not on an English locale, the string in the Type box is the one you need. And by the way, when using the “Reading Files” demo you do not need to worry about localization, it will always show the correct string.

Closing Words

We hope that after reading this document you got a better idea on the basic concepts of using the FlexCel API. It was kept short on purpose, so you can read it easily and remember it better. Concepts mentioned here (like XF format indexes) are basic to use FlexCel, so it is important that you get them right.

And one last thing. Remember that FlexCel API's main strength is that it **modifies** existing files; it doesn't use the traditional approach of one API for reading the file and another for writing. In fact, FlexCel doesn't even know how to create an empty file. When you call FlexCelImport.NewFile, you are really reading an empty xls file embedded as a resource on flexcel.dll. **You are always modifying things.**

Take advantage of this. For example, let's say you want to save a macro on your final file. There is no support on FlexCel for writing macros. But you can create the macro on Excel and save it to a template file, and then open the template with FlexCel instead of creating a new file with FlexCelImport.NewFile().

Use Excel and not FlexCel to create the basic skeleton for your file. Once you have all of this on place, modify it with FlexCel to add and delete the things you need. This is what FlexCel does best. And of course, whenever you can, use a FlexCelReport for creating files.