# TMS FlexCel for VCL and FireMonkey

## Introduction

Excel spreadsheets are used everywhere in the industry today. And if you are writing business applications, there is a very good chance that you will have to either import the data from Excel files your users created, or export the data from your application into Excel files your users can further manipulate.
As you would expect for such a common task, there are many solutions available; but interestingly enough, most of them have important flaws. Some of those solutions are:

1. Using OLE Automation. One could write a full article about the problems with OLE Automation, and I actually have, as you can read at http://www.tmssoftware.com/site/manuals/Thoughts%20about%20OLE%20Automation%20&%20Flexcel.pdf
But in short, OLE Automation just doesn't work in servers, because it hangs when more than one thread tries to access Excel at the same time. It will work in your tests, but fail when real users start stressing the system. Even worse than that, every user using your application for importing/exporting needs a valid Excel license.

2. Using OLEDB. While better than OLE Automation from a stability and performance point of view, it has almost zero formatting support.

3. Exporting or importing to/from CSV. Not only it also lacks formatting or formulas support, but also it has problems with locales. For example, in most European countries the comma is used as decimal separator instead of the dot, and as they can't use the comma also for separating fields in the CSV, the semicolon is used instead. A user with a localized version of Excel might not be able to read your csv.

4. "Educating" your users not to use Excel files, and force them to manipulate all the data inside your application.

While all of those solutions can be valid in some context, no one is really great and they all take a lot of compromises.

## Introducing TMS FlexCel Studio

FlexCel is designed to overcome the problems mentioned above. It gives your application the ability to read and write Excel files in a fast and reliable way, without needing to install Excel or any other external dll. You can forget about all of the headaches and just get your job done. In addition to that, a full and royalty-free FlexCel license costs less than a single user Excel license.

FlexCel runs in VCL and FireMonkey, Windows and OSX, and it supports fully formatted spreadsheets, both xls and xlsx.

In the rest of this paper, we will investigate how to do common things with FlexCel.

## Creating Excel files

### Creating a simple file

FlexCel provides an API for creating Excel files, which makes simple things simple, and complex things possible. For example, a minimal snippet to create an Excel file would be:

```
uses
  VCL.FlexCel.Core, FlexCel.XlsAdapter;

procedure CreateFile(const DestFileName: string);
var
  Xls: TExcelFile;
begin
  Xls := TXlsFile.Create(true);
  try
    Xls.NewFile(1, TExcelFileFormat.v2010);
    Xls.SetCellValue(1, 1, 'Hello');
    Xls.SetCellValue(1, 2,
          TFormula.Create('=A1 & " from FlexCel!"'));

    Xls.Save(DestFileName);
  finally
    FreeAndNil(Xls);
  end;
end;
```

We can note the following:

1. Uses: FlexCel has 7 "namespace units" which include all the functionality it supports. Those units are:

a. VCL.FlexCel.Core: You need to include this unit in all the VCL units using FlexCel. It provides basic types and functionality.

b. FMX.FlexCel.Core: This is similar to VCL.FlexCel.Core, but you include this unit for FireMonkey applications. When you use this unit, FlexCel will automatically use the FireMonkey classes like a FireMonkey TFont instead of VCL.

c. FlexCel.XlsAdapter: This is the Xls and Xlsx engine. You need to include this unit when reading and creating xls or xlsx files.

d. FlexCel.Render: This is the rendering engine, which FlexCel uses to "draw" a spreadsheet to PDF, HTML, or to a Printer. You need to use it if you are exporting or printing the xls/x files.

e. FlexCel.Pdf: This is a pdf engine, which can be used on its own to create PDF files. The rendering engine uses this unit to export xls/x files to PDF, but you could use it as a standalone PDF writer.

f. FlexCel.Preview: This includes the VCL FlexCel.Previewer component, which can show Excel files in the screen. You normally don't need to add this unit manually because it is added automatically when you drop a Previewer component in the form.

g. FMX.FlexCel.Preview: This is similar to FlexCel.Preview, but it contains the Previewer component for FireMonkey.

Those are all the units you need to use, so there is no need to guess where a class might be. It will always be in one of those.

2. The API is substantially different from OLE Automation. This is done on purpose, because the OLE Automation API can be very inefficient. It creates a lot of temporary objects, which need to be allocated and freed, and this adds a lot of overhead. In FlexCel performance isn't an afterthought, we have built it from the start to be as fast as possible, and this starts by the API. FlexCel API is "lower level" than OLE Automation, but this allows it to be much faster.

## Creating a complex Excel file

In the previous section we've seen the basics, and as you can notice, they are simple enough. But Excel files can be very complex, and guessing how to for example add a gradient to a cell, or protect a sheet, or add an autofilter can get difficult.

To make it easier, FlexCel provides a very useful tool named APIMate that can "convert" an Excel file into FlexCel code, for either Delphi or C++ Builder.

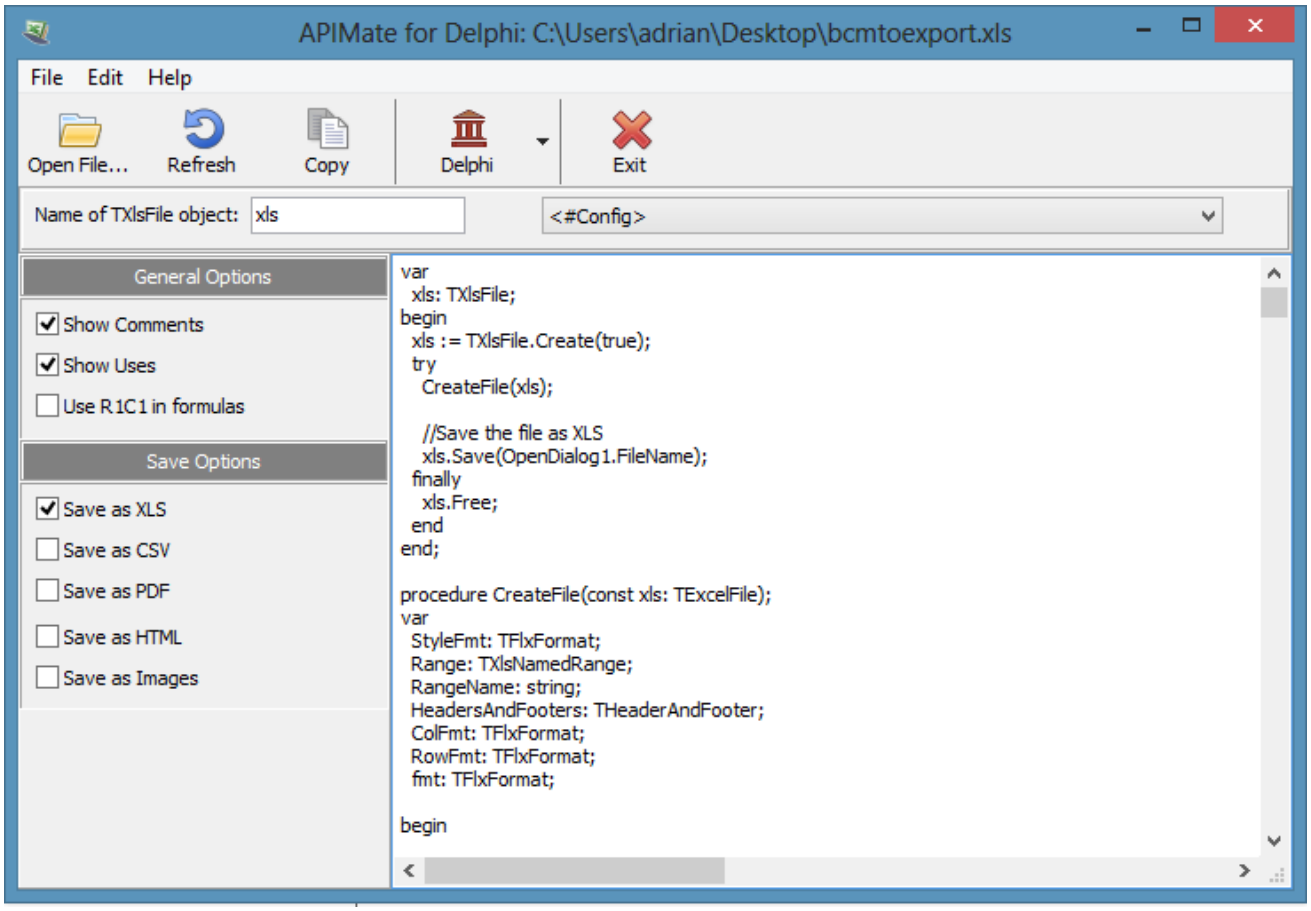


Figure 1 - Using ApiMate to create Excel files

It works like this: You create a file in Excel, and open it in APIMate (without closing Excel). It will tell you the Delphi or C++ code needed to create the file. Then you can make changes in Excel, save the file, and press "Refresh" in APIMate (all without closing Excel). It will tell you the code you need to modify the file from the old state to the new one.

We are now in position to answer the question at the top of this section: How to create a file with a gradient in a cell, a protected sheet and an autofilter? I created a simple file with those things in Excel, and APIMate gives back this code:

```
procedure CreateFile(const xls: TExcelFile);
var
  fmt: TFlxFormat;
  GradientStops: TArray<TGradientStop>;
  SheetProtectionOptions: TSheetProtectionOptions;

begin
  xls.NewFile(1, TExcelFileFormat.v2010);  //Create a new Excel
file with 3 sheets.

  //Set the names of the sheets
```

```
  xls.ActiveSheet := 1;
  xls.ActiveSheet := 1;  //Set the sheet we are working in.

  //Global Workbook Options
 xls.OptionsCheckCompatibility := false;

  //Printer Settings
  xls.PrintXResolution := 600;
  xls.PrintYResolution := 600;
  xls.PrintOptions := [TPrintOptions.Orientation];

  fmt := xls.GetCellVisibleFormatDef(3, 1);
  fmt.FillPattern.Pattern := TFlxPatternStyle.Gradient;
  SetLength(GradientStops, 2);
  GradientStops[0].Position := 0;
  GradientStops[0].Color :=
TExcelColor.FromTheme(TThemeColor.Background1);
  GradientStops[1].Position := 1;
  GradientStops[1].Color :=
TExcelColor.FromTheme(TThemeColor.Accent1);
  fmt.FillPattern.Gradient :=
TExcelLinearGradient_Create(GradientStops, 90);
  xls.SetCellFormat(3, 1, xls.AddFormat(fmt));

  //AutoFilter
 xls.SetAutoFilter(1, 3, 5);

  //Protection
  SheetProtectionOptions := TSheetProtectionOptions.Create(false);
    SheetProtectionOptions.Contents := true;
    SheetProtectionOptions.Objects := true;
    SheetProtectionOptions.Scenarios := true;
    SheetProtectionOptions.SelectLockedCells := true;
    SheetProtectionOptions.SelectUnlockedCells := true;
  xls.Protection.SetSheetProtection('******',
SheetProtectionOptions);
end;
```

APIMate is not the panacea for all the questions, but it helps a lot answering many of those "how do I?" doubts. Remember to use it.

## Creating Excel files by modifying existing ones

While you can get very far creating Excel files by hand, it can get tiring and also make your files more difficult to modify later. Besides, there are some things that you can't add with the API but that FlexCel will preserve if present in the original file, like macros or charts.

For those cases, a different approach might be worth. You can create a "template" file with all the formatting/macros/etc. in Excel and either deploy it

with your app or embed it as a resource (if you don't want to distribute extra files). Then your application reads this template and fills in the blanks.

FlexCel provides very powerful methods for manipulating existing files:

1. InsertAndCopyRange: This is a heavily overloaded method, which can do most of the things you might need to do. It can insert or copy or insert and copy a range of cells or columns or rows from one location to another, or from one sheet to another or from one workbook to another. It can also copy the full contents, or only the formulas, or only the formats.

2. InsertAndCopySheets: This will copy, insert or insert and copy sheets. When copying, you can copy a sheet from the same file or from a different one.

3. DeleteRange: It deletes a range of cells, or full columns, or full rows.

4. DeleteSheet: It deletes a full sheet.

5. MoveRange: It moves a range of cells or columns or rows from one place to another.

All those methods are heavily tested and optimized, so you can use them with trust. And all of them behave exactly like Excel, for example, if you copy the formula "=A1 + $A$1" in cell B1 to B2, it will be copied as "=A2 + $A$1"

## Reading Excel files

While reading Excel files with FlexCel API is straightforward and shown in the "Reading files" demo, there are a couple of unique features that I think are worth mentioning:

1. FlexCel can loop only on the cells that are actually used in a spreadsheet. This can make a big difference if for example you have your data in the range A1:D100, but also have a helper cell in IV1. In this case, the maximum used column would be IV, and you would have to process the range A1:IV100. If you loop in the existing cells, the number of cells will be much less.

2. FlexCel provides a "Virtual Mode" for reading files "as you go" without keeping the files in memory. If you are reading very big spreadsheets, Virtual Mode can make your application scale much better, because you can read files as big as needed without consuming any extra memory.

## Going beyond

Even when reading and writing Excel files is FlexCel's reason to be, it is far from the only things it can do.

Once you have the Excel file, FlexCel can also export it to pdf, print it, preview it or export it to HTML. You can even export it to MHTM, to create HTML formatted emails. And all of this with a lot of attention to details, so the exported files look exactly like Excel. Below, there is a screenshot of the native FireMonkey previewer showing an Excel file in OSX.

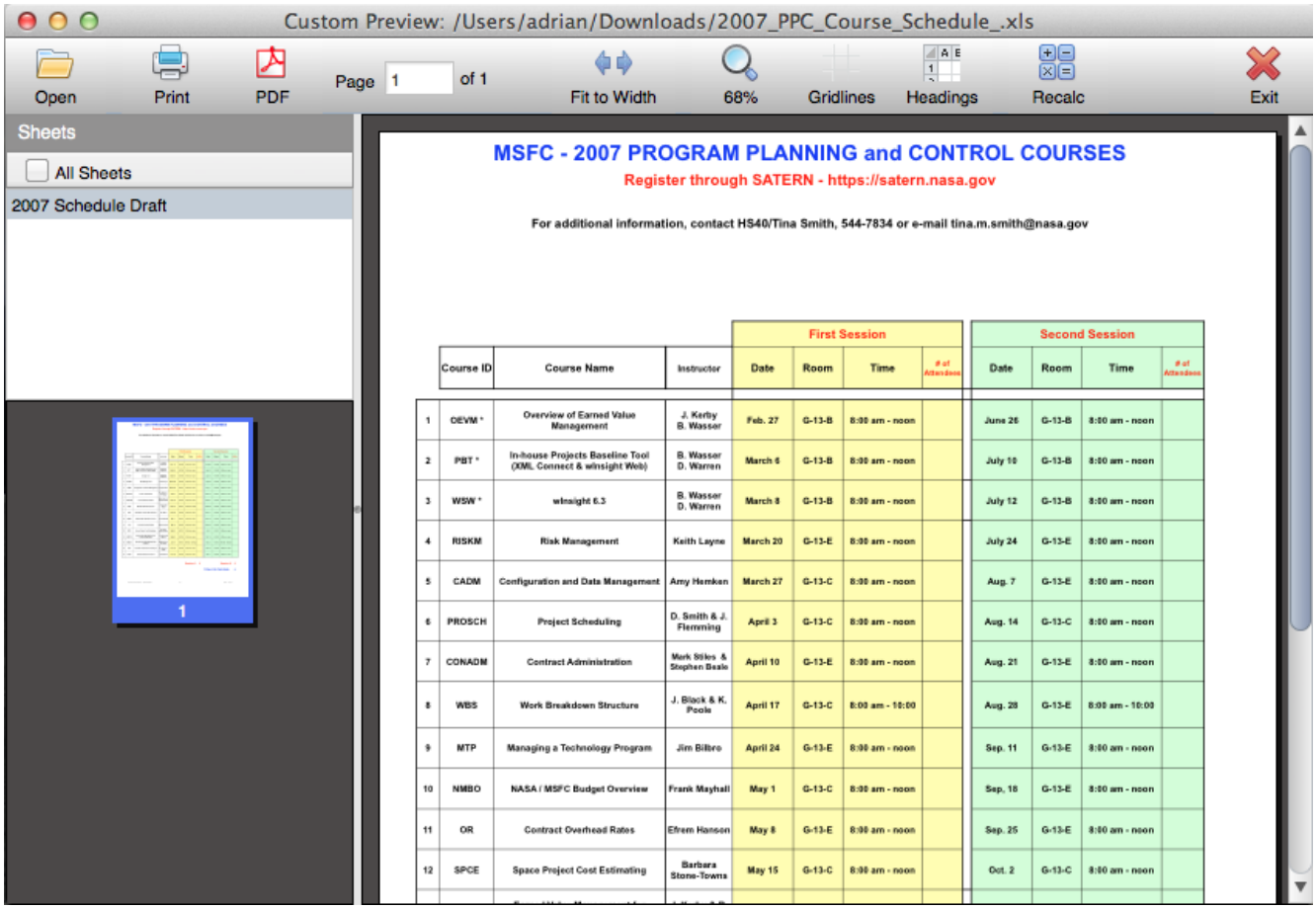


Figure 2 - The native FlexCel viewer running in OSX

## Closing words

In this paper I've tried to make a fast review of the core features available in FlexCel for VCL/FireMonkey, but there is a lot more to say, and you might get a better feeling by downloading it and running the demos available. You can get a free full functional trial at http://www.tmssoftware.com/site/flexcel.asp.