

FlexCelReport

FlexCelReport is a component for making reports in Excel from a Delphi application using a template. Uses Excel as the Report Designer, allowing you to use all of its power, from conditional formatting to pivot tables, including graphs, formulas, multiple sheets, multiple master-detail relationships and whatever you can do from Excel.

Reports can be generated by OLE automation or completely native, without need for any dll or having Excel installed.

In native mode templates can be linked into the exe, allowing you to create single exe applications.

This is a new version of TExcelReport component, renamed to get a more sillier and unique name. For old users, besides the native support, here are some new features:

- The Template property searches for the file, in the exe folder, windows folder and active folder. This allows you to specify relative paths instead of absolute.
- OnGetCellValue event: Allows you to change a field value before it goes to the report
- Multidimensional array of variant: Besides the `v[a][b][c]` array of variant formerly supported, now you can use a multidimensional array directly, `v[a,b,c]`
- Lazy Open: Datasets don't have to be opened before running the report. FlexCel automatically opens the datasets it needs and closes after it uses them. If a dataset was open before running the report, it remains open after.
- Comments/Images: Now you can replace comments and images also. (Images are not allowed in OLE mode)

INDEX

FlexCelReport.....	1
INDEX.....	2
Introduction.....	2
How to use.....	4
1- Creating the Excel Template.....	4
Now, It's time to add the ranges.....	5
2- Installing the Component.....	8
Installing FlexCelOLE.bpl.....	8
3- Creating the Delphi Datamodule.....	8
Properties.....	10
property Adapter: TExcelAdapter;.....	10
property Template: TFileName.....	10
property DataModule: TComponent.....	10
property FileName: TFileName.....	10
property AutoClose: Boolean.....	10
property CalcRecordCount: TCalcRecordCount default cr_Count;.....	11
property PagesDataSet: TDataSet.....	11
property PagesDataField: string;.....	12
Events.....	12
property OnRecordCount: TRecordCountEvent.....	12
property OnBeforeGenerateWorkbook: TOnGenerateEvent.....	12
property OnAfterGenerateWorkbook: TOnGenerateEvent.....	12
property OnBeforeGeneratePage: TOnGenerateEvent.....	14
property OnAfterGeneratePage: TOnGenerateEvent.....	14
property OnGetFileName: TOnGetFileNameEvent.....	14
property OnGetCellValue: TOnGetCellValue.....	14
Methods.....	15
procedure Run.....	15
Tips and Troubleshooting.....	16

Introduction

This component allows you to generate reports in Excel with data read from memory or from a database. It's been designed to give you all the power of an Excel spreadsheet, and I've tried hard to not just go with the 'most usual case', and give you the options when you need them.

At the moment I wrote the first version (quite a long ago...), I could find no other component that made the same, even when now I think you can find some alternatives. I didn't want to use QuickReports, because people normally didn't want just to print the reports, but to modify them, and mail them to their bosses. So I needed an Excel sheet, and the report builders were never good at exporting their data.

The mechanics are simple. You create a 'Template' in Excel where you define the layout, and put special codes where the cell should be filled with data. Then you create a Datamodule in Delphi and put the datasets that are going to access the database, along with some TFlexCelReport and one TOleAdapter or TXlsAdapter components. Then

you configure it, and from somewhere in your app call its “run” method. And that’s it. If you later want to change the template, you can do it without recompiling the application.


I’ve included a little Demo/tutorial app, with templates and the final result that should help you understand the way it works. Many times the best way to learn something is not reading the boring documentation, but just doing it...

A last thing. I did want to keep the interface as simple as possible, so I tried not to duplicate anything that Delphi or Excel can make easily. For example, I give no ways to sort the report, because sorting it is as easy as sorting the dataset. (And if you use SQL like me, you have infinite ways to combine, ascending, descending, etc). The idea is: do as much as you can in the Excel or standard Delphi side. So you get code that depends very little of this component in particular (normally, just one line of code: Report.run), making your life much easier if you want to replace it by another.

How to use

1- Creating the Excel Template

To generate reports, the first thing you need is a template. Create a document like this:

	A	B	C	D
1			INVOICE	
2			#.CURRENT_DATE	
3				
4				
5				
6				
7		Marine Adventures		
8				
9		Send to:		
10		##Cust##Company		
11		##Cust##TotAddr1		
12		##Cust##TotAddr2		
13		##Cust##Country		
14				
15		Order No	Cust No	Sales Person
16		##Orders##OrderNo	##Orders##CustNo	##Orders##SalesPeri
17				
18		Part No	Description	
19		###Items##PartNo	###Items##Description	
20				
21				
22				
23				
24				
25				

(you can see it complete in the demo file Invoices.xls)

Here you put:

- 1) All the titles, images and formatting of the report. You can include graphs, filters, conditional format, images or anything you can think of. You can also fill as many sheets as you want, all of them will be filled with the data you request.
- 2) The fields to be filled from the database. Here you have to put a text on the form **##<dataset>##<field>**
For example if you are going to create a dataset called "Cust" with a field "Country" you will write **##Cust##Country** in the cell where you want the value
- 3) You can define some extra variables in the datamodule that are not tied to the database, by writing **#.<Variable>**
. For example, if you had defined a published variant property called "Current_Date" you could include its value by writing **#.Current_Date** in a cell
- 4) You can also include variant arrays here, just write **#.<Variable>#.<index1>#.<index2>#..... #.<indexN>**

For example, to write the value of the published variant property Price[1,3] into a cell, you could write something like “#.Price#.1#.3

Note that you can use the replaced field values inside a formula. For example, if you have in the cells:

A1: ##Client##FirstName
A2: ##Client##LastName
A3: (formula) =A1 &” “ & A2
(value) ##Client##FirstName ##Client##LastName

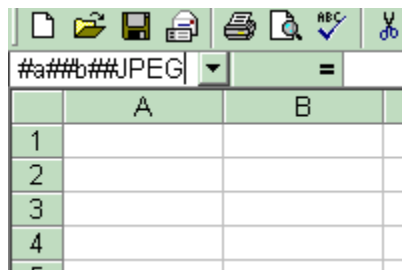
After the report is run, you will get:

A1: John
A2: Smith
A3: (formula) =A1 &” “ & A2
(Value) John Smith

But you can’t replace more than one value in the same cell. For example, if you have in A1: ##Client##FirstName ##Client##LastName, this will not be replaced correctly. To solve this case, you need to create a new calculated field in the dataset that contains the two others concatenated, and use this field in the template.

Another thing to take in count, date and time fields. They will be passed as a number to Excel (there is no variant to represent a date/time). So you **must** format the cell including the date (for example ##Client##SaleDate) with a **date** format. Note that when programming the component I’ve tried to pass Field.value to this cells (it passes a string), and it works without needing to format, but has problems with international representations.

Besides cells, you can also stream Images (only in native mode) or comments to the file. For comments, just write the usual ##Dataset##Field in the comment. For images, drop a Blank image, select it, and in the names combo, change its name to ##DataSet##Field##ImageType



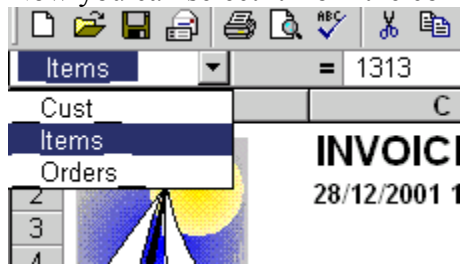
where ImageType is JPEG or PNG. Verify that the image format in the database is one of them, or convert them as needed. For more info, see the demo app.

Now, It’s time to add the ranges

You have to define one named range for each dataset you want to use, with the name __<DataSet>__ For example, if you want to populate the dataset “Items” in the range B19:E19 you should:

Go to Excel and select Insert->Name->Define from the menu.
Create a “__Items__” range for B19:E19

Now you can select it from the combo box.



This range will be copied once for each entry of the Dataset, with the values replaced. It is possible to create as many ranges as you want inside others, to reflect master-detail relationships. (For example, you may have the __Cust__ range covering all used range of the sheet, and then a __Orders__ range inside and a __Items__ inside the __Orders__

When the report is run, the __Cust__ range will be copied as many times as records are in the Cust Dataset, and for each copy, the range __Orders__ will be copied as many times as records are in the Order dataset, for the corresponding value of the __Cust__ Dataset. Same way, the __Items__ range will be copied in each __Orders__ copy, as many time as entries are in the Items dataset, for the corresponding Orders and Cust entry.

Range notes

Note 1: Don't intersect ranges, or you are going to get strange results. Each range should be either completely inside or completely outside the others.

Note 2: The ranges affect the entire row that will be copied. Never put one range to the right or left of the other.





Note 3: It is possible to define some __XX__ ranges that don't correspond to any dataset. These ranges are not going to be copied, but values will be substituted

Note 4: Always define some Range that includes all the others, so the component knows where to make the replacements. FlexCelReport always will look for the biggest range to begin replacing values. I usually call this range __MAIN__, but you can call it as you want, as long as it begins with “__” and ends with “__”

Blank Rows





A problem that happens with Excel and that I have not been able to completely solve is related with the way you insert the cells.

Imagine you define the following Template:





TEXT0					=Sum(B2:B2)
	A	B			
1	Item	Price			
2	###Items###Name	###Items###Price			
3	Total	=Sum(B2:B2)			
4					
5					
6					

And let be the __Items__ range = A2:B2

When the report is run it will insert n-1 cells between rows 2 and 3, where n is the number of records in the Items dataset. But the formula in B3 (“=Sum(B2:B2)”) will not change to reflect the inserted rows. You will get something like this:

TEXT0					=Sum(B2:B2)
	A	B			
1	Item	Price			
2	Banana	100			
3	Apple	200			
4	Total	=Sum(B2:B2)			
5					
6					

As a rule of thumb, to solve this problem you should **always leave a blank line** below the ranges to let the formulas adapt. You should define a template like:

TEXT0						=Sum(B2:B3)
	A		B			
1	Item		Price			
2	###Items###Name		###Items###Price			
4	Total		=Sum(B2:B3)			
5						
6						

With the __Items__ range defined as A2:B2. Now, when the rows are inserted between rows 2 and 3, the formula now in B4 will be updated right.

You can leave the template this way (by making the row 3 very small), or if you want to get rid of the blank line, you could use the “...delete row...”, as in the image.

	A	B
1	Item	Price
2	###Items###Name	###Items###Price
3	...delete row...	
4	Total	=Sum(B2:B3)
5		
6		
7		

All the rows that have in the “A” column the text “...delete row...” (without any spaces an all in lowercase) will be deleted after the report is run. An use of this is if you want to create a Pivot Table or a Graph , and you can see it in the PivotDemo.xls template.

2- Installing the Component

This should be very straightforward, but it isn't... so.... here we go

There are 3 Packages to install:

- **FlexCel.bpl**: This is the main package and the first to install. Just open it with file->open, and click install.
- **FlexCelXLS.bpl**: This gives you the native support. You only have to install it if you are using Native adapters.
- **FlexCelOLE.bpl**: This is for the OLE support. Installing it is a bit tricky (see below). You only have to install it if you want OLE support.

Anyway, to run the demo, you should install them all.

Installing FlexCelOLE.bpl

Delphi has 2 **different** sets of Office server components, Excel97 (the one that comes with the D5 unpatched) and Excel2000 (this one comes with the update) In D6, you can choose when you install if you want one OR the other. The names in D5 are **dcloxserver50** for Office97 and **dcloffice2k50.bpl** for Office2000. In D6, replace the 50 by 60.

The problem is, as this set of components have the same name, you can only register ONE in the palette. In D5, you have Office97 registered (because it is the one by default), even when you have the units Excel2000 etc. So, when you want to register a component (TOleAdapter) using Excel2000, you will get an error message and it will not be installed.

So the first thing you have to know is what component set you have installed. Go to Menu->Component->Install Packages... and search for **Borland Sample Automation Server Components** or **Microsoft Office 2000 Automation Components**. Look at the name of the file and see if they are **dcloxserver50** or **dcloffice2k50.bpl**

Now, if you are using **dcloffice2k50.bpl** (Office2000) you don't have to do anything. If you are using **dcloxserver50** (Office97), right-click the FlexCelOLE package, select Options... and in the "Directories/Conditionals" tab define EXCEL97. Build and install it. Or, you can always uninstall it and install Office 2000 Components.

If you get an error saying "Component xxx can't be registered by package xxxx because it has already been registered by package xxxx", you are probably using the wrong define.

3- Creating the Delphi Datamodule

Now let's go with the other part, creating the app that will generate the reports. First, you define a Datamodule (not necessary, it could be a Form too) where all the data to the reports will be. In my experience, it's better to have all the queries for the reports different from the ones for the rest of the app.

Then you have to add the datasets, one for each in the template, and with the same names. Case is not important. After that, define all the calculated fields.

Then, it's time to define the properties (for the cells including “#.<Variable>”) They should be **published** and return a **variant**. Of course that variant may be a multidimensional array of variant. (for the cells including “#Variable#.index#.index #.index)

If you want to create a multisheet report, you can assign one dataset to the [PagesDataSet](#) property, and the active page will be copied (and filled) for each record of the dataset.

Warning: In OLE mode don't use a large dataset to populate the sheets, or it will take forever to fill (and it will be very difficult for the user to navigate between all the sheets!) I would advice that the dataset should have no much more than 10 records. If you are using Native mode, you can make it larger, but don't abuse.

The transactions should be Snapshot or equivalent if possible, because this will allow the data to remain the same during all the process.

Now drop a TFlexCelReport and a TXlsAdapter or a TOLEAdapter in the module. Fill in the properties, and that's all. Whenever you want to generate the report, call the “run” method of the component.

Properties

property Adapter: TExcelAdapter;

This property is connected with a TOLEAdapter or a TXLSAdapter you must drop into the Datamodule. If you connect it with a TOLEAdapter, all the operations will be made using Excel OLE Automation. If you connect it to a TXLSAdapter, everything will be made natively, without calling Excel.

property Template: TFileName

This is the excel file that's going to be used as the template to generate the report. Path can be relative. For example, if you set Template:= 'templ\t1.xls', and your exe is at c:\myApp, it will search for the template at 'c:\myApp\templ\t1.xls', and if it doesn't find it, will search in the windows folder\templ\t1.xls.

If you are using a TemplateStore, FlexCel will use it for the Template, and will not search the disks. You can still let the path info in the template property, but it will not be used. For example, if Template:= 'templ\t1.xls', FlexCel will search for t1.xls inside the TemplateStore.

property DataModule: TComponent

This is the datamodule where all the datasets for the report are located, and the published properties defined. Normally it will be the same datamodule that has the TFlxCelReport component, so it's assigned automatically and you don't have to care for it.

Really, the datamodule doesn't need to be a TDatamodule descendant, but any TComponent, as long as it holds the necessary DataSets. But for any medium-complex project, it will probably be one TDatamodule dedicated exclusive to the reports.

property FileName: TFileName

FileName is the name of the file to save when the AutoClose property is true. When AutoClose is false it has no meaning.

Don't put the extension (xls) in the filename, it is added automatically, and can be more than one if you save to multiple file formats.

This filename can be later modified for each one of the file formats used, see event [OnGetFileName](#)

property AutoClose: Boolean

This property only has meaning in OLE mode. In Native mode, the report is never opened, and you have to do this by hand. (Take a look at the demo)

When true, the Report is saved with the name specified by the Filename Property, and is never shown. This is useful for creating automatic non interactive reports for mailing, etc.

When false, the report stays open for the user to view.

property CalcRecordCount: TCalcRecordCount default cr_Count;

If you have spent some time with different databases, you probably know about recordcount and all the problems it brings. Some databases give the right number of records, others will give the unfiltered count (be careful with these, they may work right until someone filters the dataset), others will give always -1, and so on.

The conclusion to it would be “don’t use recordcount at all”. But I really needed to. Passing data to Excel is by far the most time consuming task, much more than counting the records. So the code

```
(a) db.First; Rc:=0;
    while not db.eof do inc (Rc);
    ExcelApp.Insert(Rc) ;
```

is much faster than

```
(b) db.First;
    while not db.eof do ExcelApp.Insert
```

Then, depending of the database and the dataset you use, there are 4 ways to count the records: The first is using the [OnRecordCount](#) event (see below), and the others are setting CalcRecordCount to:

cr_None: Use it if you are sure the recordcount is always right, with the dataset filtered or not. This is the case if you are using a TClientDataSet

cr_Count: The default. It will make a db.Last before using RecordCount, to make sure all records have been fetched. This works with some BDE datasets, or InterbaseExpress.

cr_SlowCount: This will assume RecordCount is always wrong and count the records one by one using a procedure like the one in (a). Whenever you are not sure, or you are getting strange results use this.

property PagesDataSet: TDataSet

If you assign this property, the active page on the template will be duplicated once for each entry of the DataSet, and filled with the data corresponding to this entry. The name of the page will be given by the [PagesDataSetField](#) property.

property PagesDataField: string;

This is the name of the field that will be used to name the pages created. Use this together with [PagesDataSet](#) to create a report that spawns multiple sheets.

Events**property OnRecordCount: TRecordCountEvent**

Use this event to calc the number of records and tell the TFlexCelReport component by assigning the RecordCount variable, if you can't/don't want to use the [CalcRecordCount](#) property. The body for the event may be something like this:

```
procedure TTestReport.ExcelReport1RecordCount(Sender: TObject;
const DataSet: TDataSet; var RecordCount: Integer);
begin
    QCount.Open;
    try
        RecordCount:=QCountTotal.Value;
    finally
        QCount.Close;
    end; //finally
end;
```

and the SQL property of QCount may be something like this:
Select count(*) as Total from table

Be sure that QCount and Dataset use the same readonly, snapshot transaction, or someone might delete a record from table after calling QCount, but before the data is passed to Excel.

Note: Assigning this event leaves the value of [CalcRecordCount](#) without meaning.

property OnBeforeGenerateWorkbook: TOnGenerateEvent

This event fires after the Workbook has been created, but before putting anything in it. This allows you to perform custom actions on the worksheet. For more information on writing the custom macros, see [OnAfterGenerateWorkbook](#) event

property OnAfterGenerateWorkbook: TOnGenerateEvent

This event fires after the Report has been completed and allows you to write custom actions on it. This is especially useful if you are using OLE, because you can gain access to the Excel objects and automate them by your own.

For example, let's think you want to protect the report after it is generated (To protect individual sheets, you should use OnAfterGeneratePage)

First thing to do is open Excel and record a macro: Go to Menu->Tools->Macro->Record New Macro.

Write the name for the macro (we'll use "Macro1")

Goto Tools->Protect->Protect book

Stop the macro recording.

Press Alt-F11 to go to the Macro editor. (or goto Menu->Tools->Macro->Visual Basic Editor)

Open "Module1" in the Project Explorer

You should see something like this:

```
Sub Macro1()  
,  
  Macro1 Macro  
  Macro grabada el 20/03/2002 por adrian  
,  
  ActiveWorkbook.Protect  
End Sub
```

So, now you know the method is called "Protect" and applies to a Workbook.

Now open Delphi and double click in the OnAfterGenerateEvent

You get a handle to an ExcelFile var that (as we are using automation) gives you access to:

The ExcelApplication Object

The ExcelWorkbook Object

The LCID variable, this is just a parameter you might need to pass to the procedures you call. This parameter is never needed when recording macros in Excel, but Delphi might ask for it. If it does, just write it.

Now, type something like this

```
procedure TDemo.RepAfterGenerateWorkbook(Sender: TObject;  
  const ExcelApp: TExcelFile);  
var  
  Wb: TExcelWorkbook;  
  i: integer; v:variant ;  
begin  
  if ((Sender as TFlexCelReport).Adapter is TOLEAdapter) then  
  begin  
    Wb:=(ExcelApp as ToleFile ).ExcelWorkbook;  
    Wb.Protect;  
  end;  
end;
```

And that's all. Remember that when you call a method you always have to write all the parameters, just type "EmptyParam" for an empty one. For example, to save the file with another name you could write:

```
ExcelWorkbook.SaveAs('C:\test', EmptyParam, EmptyParam,  
EmptyParam, False, False, xlExclusive, EmptyParam, EmptyParam,  
EmptyParam, EmptyParam,LCID);
```

If you want to do something specific to each sheet, you should use `OnAfterGeneratePage` instead. This event gives you an additional variable, `ExcelWorkSheet`.

Note: Be careful what you do on these events. You get complete access to Excel with them, and you could easily hang it.

property `OnBeforeGeneratePage`: `TOnGenerateEvent`

This event fires after each page is selected, but before putting anything in it. This allows you to perform custom actions on the worksheet. For more information on writing the custom macros, see [OnAfterGenerateWorkbook](#) event

property `OnAfterGeneratePage`: `TOnGenerateEvent`

This event fires after each page is created. This allows you to perform custom actions on the worksheet. For more information on writing the custom macros, see [OnAfterGenerateWorkbook](#) event

property `OnGetFileName`: `TOnGetFileNameEvent`

This event fires before saving the file, once for each one of the formats you are saving and allows you to modify the filename for that format. If you don't assign this event, the file will be saved with the [Filename](#) property, plus an extension relative to the file format.

For example, if you had

```
ExcelReport.FileName:='c:\Hello';  
OleAdapter.SaveFormatBasic:=[saCSV, saExcel9795];
```

You will get the files: 'c:\hello.csv' and 'c:\hello.xls'

Now, if you assign the `OnGetFileNameEvent`:

```
ExcelReport.FileName:='Hi';  
  
procedure TFFTestReport.ExcelReportGetFilename (Sender:  
TObject; const FileFormat: integer; var Filename:  
TFileName);  
begin  
  case FileFormat of  
    xlCSV: Filename:='c:\csv\' + FileName;  
    xlExcel9795: Filename:='c:\xls\' + FileName;  
  end; //case  
end;
```

You will get the files: 'c:\csv\hi.csv' and 'c:\xls\hi.xls'.

property `OnGetCellValue`: `TOnGetCellValue`

This event is fired before each value is written to the Excel sheet. You can modify it here.

Methods

procedure Run

This is the only procedure you have to care about. Call it when you need to generate the report, after having set all of the properties.

Tips and Troubleshooting

Here are some tips taken from the mistakes I make more often. Feel free to add your own

- Properties referenced in the template must be of **variant** type and **published**.
- Do not create data ranges that intersect, like __d1__=a1:a5 and __d2__=a3:a7. You will be asking for trouble. See [Range notes](#)
- Be careful when creating formula that refers to inserted rows, always leave a blank row so the formula is updated. See [Blank rows](#).
- If you see 'garbage' data at the end of the report, or if not all the data is present, it may be a problem of the recordcount property. See [CalcRecordCount](#)
- Beware with the **dates**! They will be passed as a **number** to Excel, so the cells in the template must be formatted with a date/time format.
- For reports in general, be sure to use a 'snapshot' or similar transaction mode, or the data may be changed between the first record that goes to Excel and the last.
- When running the demo, you might end up with some files *.mb. These are temporary files from the BDE, not from the component!!! (The component doesn't create any temporary file)
- **Always** define a __MAIN__ range ! See [Range notes](#)
- When you don't know which command or parameter to use, record a macro in Excel and see what it does. And don't forget you have the help on Visual Basic for Excel also.
- In native mode, when you use formulas, Excel will ask for saving a Worksheet saved with an earlier version. This really means that it will save the worksheet with the formulas recalculated. The only way to avoid this message is not to use formulas.
- You can use Spanish messages. All FlexCel messages are saved in the files XlsMessages.pas and UFlxMessages.pas as resourcestrings. If you define "SPANISH" on these units, messages will be translated. Also, if you want to translate them to your own language, you only need to look there. (if you do this, please send me a copy...I will include them in the next version)