Flexcel Reports Developers Guide

Documentation: May, 2010

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

Table of contents

Introduction

This document is a part of a 2-part description in how to create Excel files by “reporting” instead of by code. In this part we will look at how to set up the coded needed to create the files, and in the next part (FlexCel Report Designers Guide) we will look at how to modify the Excel file needed to create the reports.

About Excel Reporting

FlexCel gives you two ways to create an Excel file, with XlsFile (API) or with FlexCelReport (Template).
Each method has its good and bad things, and it is good to know the advantages and drawbacks of each.

Creating a report using XlsFile is a low level approach; we might if you want compare it with programming on assembler. Assembler programs are fast and can use all the power on your machine. Also, assembler programs are difficult to maintain and difficult to understand to anyone who was not the author. And sometimes they are difficult to understand for the author too.

Similar to assembler, creating a report with XlsFile will result on a really fast report, and you have access to the entire API, so you can do whatever you can do with FlexCelReport and more. After all, FlexCelReport uses XlsFile internally.
And, again similar to assembler, XlsFile reports can be a nightmare to maintain. When you reach enough number of lines like:

```
xls.SetCellValue(3,4,”Title”);
XlsFormat.Font.Bold=true;
XlsFormat.Font.Size=14;
XF = xls.AddFormat(XlsFormat.);
xls.SetCellFotmat(3,4, XF);
```

Changing the report can become quite difficult. Imagine the user wants to insert a column with the expenses (And don't ask me why, users always want to insert a column). Now you should change the line:

```
xls.SetCellFotmat(3,4, XF);
```

to

```
xlsSetCellFotmat(3,5, XF);
```

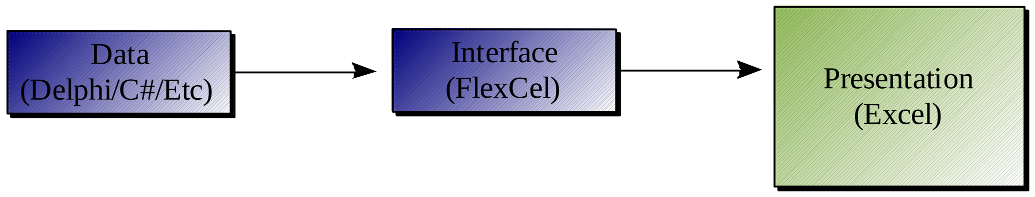
But wait! You need to change also all references to column 5 to 6, from 6 to 7... If the report is complex enough, (for example you have a master detail) this will be no fun at all.
But there is something much worse on using XlsFile directly. And this is that again, similar to assembler, only the author can change the report. If your user wants to change the logo to the new one, he needs to call you, and you need to recompile the application and send a new executable.

FlexCelReport is a higher level approach. The design is cleanly separated on three different layers, data layer, interface layer and presentation layer, and most of the work is done on the presentation layer, with Excel. You design the report visually on Excel, and you mess as little as possible with the data layer. If the user wants to change the logo, he can just open the template and change the logo. If he wants to insert a column, he can just open the template and insert a column. And the application does not even need to be recompiled.

As with any higher level approach, it is slower than using the API directly, and there are some things where it is more limited. But all in all, reports are really fast too and the things you cannot do are not probably worth doing anyway.
So, the option is yours. For normal reports we highly recommend that you use FlexCelReport. Even when it can take a little more time at the beginning that just hurry and start coding, it will pay on the long run. And much.

Organization of a FlexCel Report

A FlexCel report can be seen as three different modules working together to create the Excel file. Different from a "Coded" report where you mix data access with presentation, here each part has its own place, and can be developed separately by different people with different skills.



1. **Data Layer**
   This is the layer where we read our data from a database or memory and prepare it be read by the interface layer.

2. **Interface Layer**
   This layer's mission is to "answer" petitions from the Presentation layer, providing the data it needs to create a report.

3. **Presentation Layer**
   This is the most complex and changing layer of the three. Here is where you design all the visual aspects of the report, like data position, fonts, colors, etc.

The big advantage of this "layering" is that they are somehow independent, so you can work on them at the same time. Also, Data and Interface layers are small and do not change much on the lifetime of the application. Presentation does change a lot, but it is done completely on Excel, so there is no need to recompile the application each time a cell color or a position changes.

Remember: The data flow goes from the Presentation layer to the Data layer and back. It is the presentation that asks FlexCel for the data it needs, and FlexCel that in turn asks the data. It is not the application that tells FlexCel the data it needs (As in SetCell(xx)), but FlexCel that will ask for the data to the application when it needs it.

On this document we are going to speak about **Data** and **Interface** layers. The Presentation layer is discussed on a different document, "FlexCel Report Designers Guid" because it is complex enough to deserve it, and because it might be changed by someone who only understands Excel, not .NET. And we don't want to force him to read this document too.

Data Layer

The objective of the data Layer is to have all data ready for the report when it needs it. Currently, there are five ways to provide the data:

1. **Via Report Variables:** Report variables are added to FlexCel by using "FlexCelReport.SetValue". You can define as many as you want, and they are useful for passing constant data to the report, like the date, etc. On the presentation side you will write <#ReportVarName> each time you want to access a report variable.

2. **Via User defined functions:** User defined functions are classes you define on your code that allow for specific needs on the presentation layer that it can't handle alone. For example, you might define a user function named “NumberToString(int number) that will return “One” when number=1, “Two” when number =2 and so on. On the presentation layer you would write <#NumberToString(2)> to obtain the string “Two”

3. **Via DataSets:** .NET DataSets are a collection of memory tables that can be sorted, filtered or manipulated. Whatever format your data is, you can use it to fill a DataTable and then use that data on FlexCel. DataSets are the primary way to provide data to FlexCel.

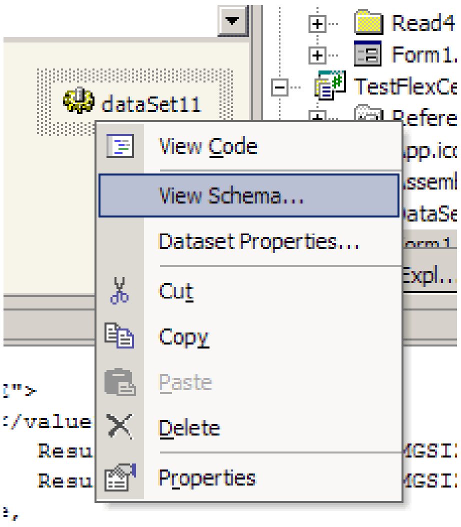
What if you have your own business objects and you do not want to use a dataset? Well, you can go for option number 5), Virtual Datasets. But before jumping there, please note the following: The presentation layer is allowed to further filter your data, sort it, etc. When using DataSets, FlexCel provides this functionality. But if you use your own objects, you will have to do it yourself, or live without the functionality.

So, if your data is for example on an array, you can just fill a Dataset with it. This will give the user the extra power to filter and sort the array. And remember that sometimes, a temporary buffer might not only not slow things down, but speed them up.

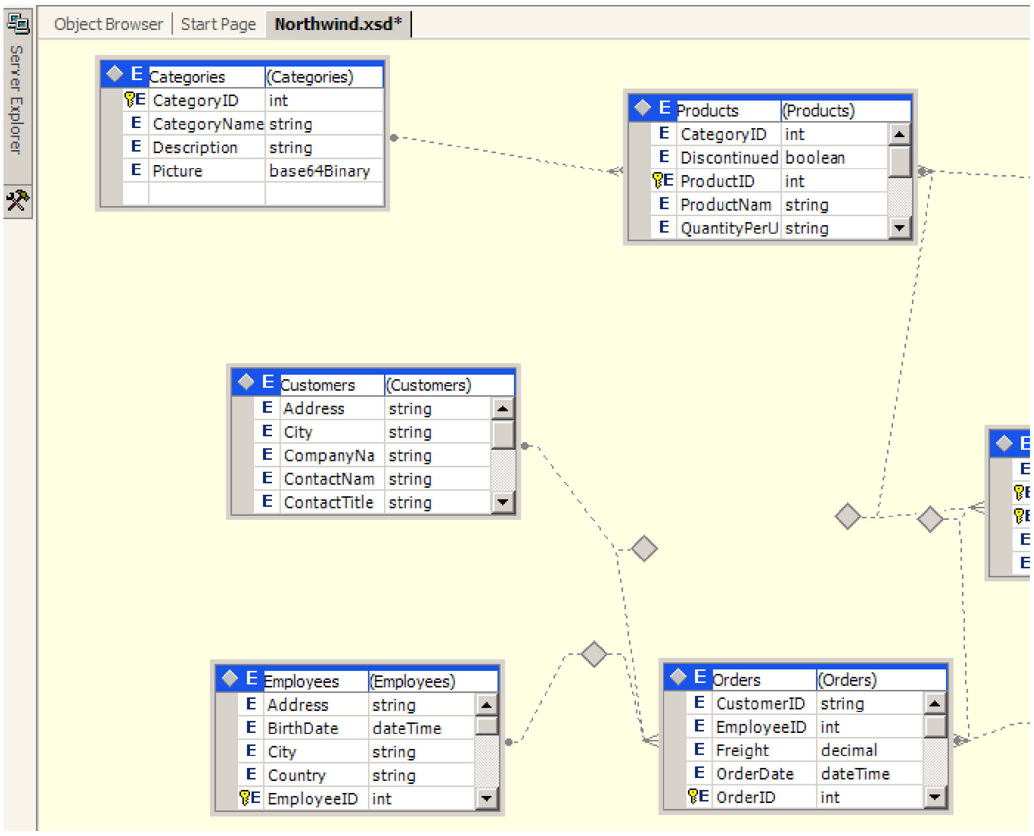
4. **Via Direct SQL in the template:** For maximum flexibility, you can set up the data layer on the template too. On this case, you need to add a database connection to FlexCel, and all the rest is done on the template. This allows the users to completely edit the reports, even the data layer from the template, but also it allows users to do thing they might not be intended to. Use with care.

5. **Via Virtual Datasets:** If the overhead of copying your existing objects to datasets is too much, you can implement your own wrapper objects to provide this data to FlexCel without copying it. As said before, please take this option with care. DataSets are really fast and optimized, and if you create your own objects you will have to implement a similar level of performance. Please read the Appendix II for a more detailed description of virtual datasets.

Setting up Data Relationships
Once you have created a dataset and its datatables associated, you need to set up the relationships between datatables. You can do this by code, adding DataRelation objects, or use the schema editor on Visual Studio.



Once you are on the schema (or if you are doing it by code), set up all the master detail relationships between your tables.



Categories	(Categories)
CategoryID	int
CategoryName	string
Description	string
Picture	base64Binary

Products	(Products)
CategoryID	int
Discontinued	boolean
ProductID	int
ProductNam	string
QuantityPerU	string

Customers	(Customers)
Address	string
City	string
CompanyNa	string
ContactNam	string
ContactTitle	string

Employees	(Employees)
Address	string
BirthDate	dateTime
City	string
Country	string
EmployeeID	int

Orders	(Orders)
CustomerID	string
EmployeeID	int
Freight	decimal
OrderDate	dateTime
OrderID	int

This is all you need to do on the data layer. Once the correct relationships are created, you can create hundreds of reports without touching it anymore.

Imagine for example that you want to print the orders by customer. As Customer table is related with Orders table by OrderId, you can define two named ranges on the Excel template and it will automatically relate the orders with the corresponding customers. Now, if you want to know Orders by Employee, it is just defining an __Employees__ range and a __Orders__ range one inside the other on Excel. As they have a relationship too, orders will be automatically grouped by Employee. (And not by customer anymore, since there is no “__Customer__” range anymore).

Note: You might also define the relationships directly in the template instead of by code. This is useful when using Direct SQL, since the tables are defined directly in the Excel template and you can’t set up the relationships in code. This is described in detail in the designers guide, in the section about Direct SQL. Also, you can add custom relationships not related with datasets, when using virtual datasets. Again, this is described in the section about virtual datasets in this document.

Interface Layer

This is the simplest of the three, as all work is done internally. To setup the interface layer, you only need to tell FlexCel which datasets, tables and user functions it has available from the data layer. For this, you use “SetValue”, “SetUserFunction”, “AddTable” and “AddConnection” methods on FlexCelReport. Once you have told FlexCel what it can use, just call FlexCelReport.Run and this is all.

Note: As simple as it is, it is worth spending some time thinking what tables, functions and variables from the data layer you want to make available for the final user. Limiting it too much might mean cutting the power of the user to customize his report. And just adding everything might be too inefficient, since you are loading thousands of rows the user is not going to use, and might also have some security issues, as the user might get access to tables he is not expected to. You can also use SQL on the templates to allow maximum flexibility, but then you need to be extra careful about security.

A closing note

Once you finished reading this document and the “FlexCel Reports Designers Guide” about the presentation layer, it can be a good idea to look at the flash presentations available at www.tmssoftware.com/flexcelnet.htm
There you can see the three layers on a real example.

Appendix I: Translating FlexCel

FlexCel is designed with localization on mind. If you want to translate it to your language, here are the things you need to do.

Basic Concepts

There are three kinds of localization you might need to do to translate FlexCel: GUI localization, Message localization and Xls localization. Here we will cover what each one of them is, and in the next section which kinds you need to use to localize different parts of FlexCel.

GUI Localization:
To translate a FlexCel form to your language, open it with Visual Studio, select its language property to the corresponding language, and just change all the captions on the form. Nothing more needs to be done. When you select back default language, all the captions should be reset to English.

Message Localization:
FlexCel includes a lot of resx files with all the messages it generates. Not a single message is hardcoded as a string, so you can change those files to get the messages in your language. To translate messages, copy the corresponding resx file to your language and edit it with Visual Studio or an XML editor.
For example to translate "file.resx" to Spanish, you need to copy it as file.es.resx, add it to the project manager and change the file.

Xls Localization:
FlexCel also uses internally some xls files that you might need to translate too.

Localization Options

Following is the list of the different things you can localize, ordered by importance, the type of localization needed, and the files you need to change.

Translating FlexCel designer
If you plan to redistribute FlexCel designer to your final users so they can customize their reports, you should consider translating it.

- GUI Localization: tools\FlexCelDesigner\MainForm.cs
- Message Localization: tools\FlexCelDesigner\fldmsg.resx
- Xls Localization: tools\FlexCelDesigner\Config.xls (holds the configuration sheet)

Translating FlexCel Report messages.
- Message Localization: source\core\flxmsg.resx

Translating Xls Native Engine messages.
- Message Localization: source\XlsFile\xlsmsg.resx

Optionally, you could translate the tags and or function names to your language. (for example, translate <#delete row> to <#borrar linea>. But this is strongly not recommended, because the report will stop working when you switch languages. If you still want to do it, search for the other resx files on the source.

One good thing about using reports and not direct coding them is that localizing the files you create can be done very easy just by having differently localized templates. You can have a template in English and other in Spanish, and depending on what you choose, the report will be in one language or another.

Appendix II: Virtual DataSets

This is an advanced topic, targeted only to experienced developers. As explained earlier, the easiest way to provide data to FlexCel is via DataSets. DataSets are fast, optimized memory tables, with lots of added functionality, like data relationships, filtering or sorting, and if you are not sure you can make it better, you are probably better using them.

But you might to use your own objects directly, to avoid storing the data in memory twice.
You can do this by writing wrapper objects around your data, implementing the abstract classes **VirtualDataTable** and **VirtualDataTableState**. In fact, standard Datasets also inter operate with FlexCel by implementing those classes. If you have FlexCel sources, you can search for **TAdoDotNetDataTable** and **TadoDotNetDataTableState** to see how they are implemented.

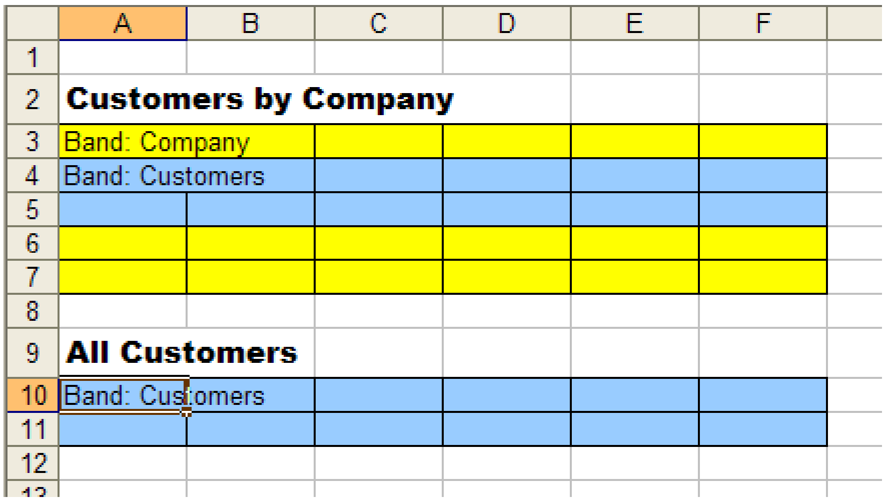
Then, we have two classes to implement:

1. On one side we have VirtualDataTable; that is a "stateless" container, much like a dataset. Each virtual dataset corresponds with a table on the data layer that you would add with "FlexCelReport.AddTable", or create by filtering existing datasets on the config sheet.

2. On the other side we have VirtualDataTableState.

   Each VirtualDataTableState corresponds with a band on the presentation layer.

   This is probably easier to visualize with an example. Let's imagine we have the following report:



	A	B	C	D	E	F
1						
2	Customers by Company					
3	Band: Company					
4	Band: Customers					
5						
6						
7						
8						
9	All Customers					
10	Band: Customers					
11						
12						

And this code:

```
FlexCelReport.AddTable(Customers);
FlexCelReport.AddTable(Company);
FlexCelReport.Run();
```

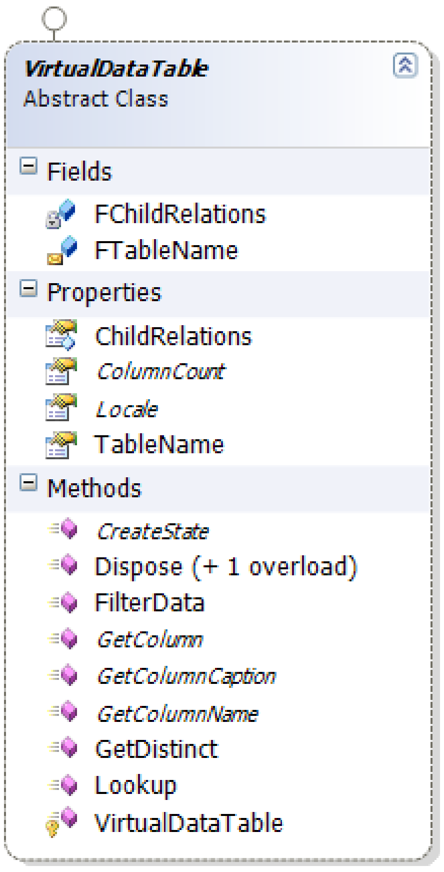
There are two VirtualDataTables here (Company and Customers), and three VirtualDataTableStates (One for the Company Band, one for the first Customers Band and one for the second Customers Band)

Take note that the same VirtualDataTable is used by two different VirtualDataTableStates, and might be used by other VirtualDataTableStates in other threads. This is why you cannot save any "state" information on the VirtualDataTable, and if you write to any private variable inside of it (for example to keep a cache) you should use locks to avoid threading issues.

Always assume that some other class might be reading your data.

VirtualDataTableState on the other hand is a simple class that will not be accessed by more than one class, and you can do whatever you want inside it without worries of other threads trying to access it.

Creating a VirtualDataTable descendant



At the left, you can see the class diagram of a VirtualDataTable class. Please note that you do not need to implement every method to make it work, just the ones that provide the functionality you want to give to your end users.

Required Methods:
On every DataTable you define, you need to implement at least the following methods:

GetColumn, GetColumnCaption, GetColumnName and **ColumnCount:**
Those methods define the "columns" of your dataset, and the fields you can write on the <#dataset.field> tags on the template.

CreateState: It allows FlexCel to create VirtualDataTableState instances of this DataTable for each band. You will not create VirtualDataTableState instances directly on you user code, FlexCel will create them using this method.

Optional Methods:
Now, depending on the functionality you want to provide to the end user, you might want to implement the following methods:

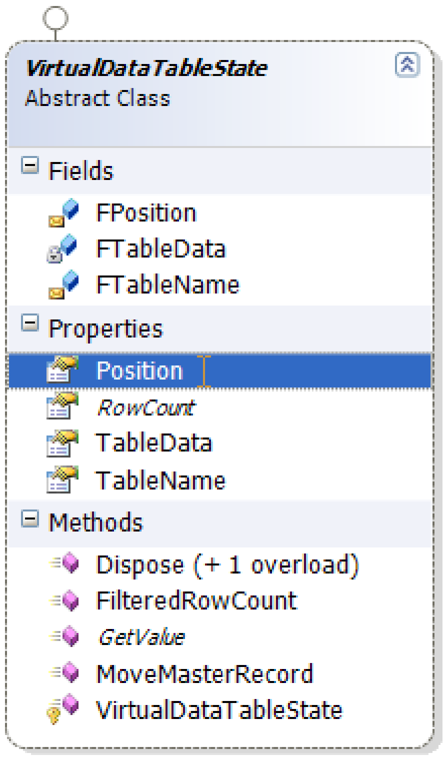
FilterData: Will return a new VirtualDataTable with the filtered data.

You need to implement this method if you want to provide the user the ability to create new datasets by filtering existing ones on the config sheet. If you do not implement it, any attempt to create a filtered dataset on the config sheet will raise an exception.
Note that this only applies to standard filters. For <#Distinct()> or <#Split> filters you do not need to implement this.

GetDistinct: Will return a new VirtualDataTable with only unique records.
Implement this method if you want to let your user write <#Distinct()> filters on the config sheet.

LookUp: Will look for a record on the dataset, and return the corresponding value. This method allows the user to use <#Lookup()> tags on their reports.

Creating a VirtualDataTableState descendant



Now let's study the VirtualDataTable class: Again, you do not need to implement every method on this class, and the non implemented methods will just reduce functionality.

Required Methods:

RowCount: Here you will tell FlexCel how many records this dataset has on its current state. Remember that this might not be the total record count. If for example you are on a master detail relationship, and the master is on record 'A', you need to return the count of records that correspond with 'A'. Make sure this method is fast, since it is called a lot of times.

GetValue: Here you will finally tell FlexCel what is the value of the data at a given row and column. You can know the row by reading the “Position” property, and the column is a parameter to the method.

As with RowCount, this method should return the records on a specific state, not all the records on the datatable. If you are in a master-detail relationship and only two records of detail correspond the the master position, GetValue(position = 0) should return the first record, and GetValue(Position = 1) should return the second. It doesn't matter if the total number of records is 5000.

Optional Methods:

MoveMasterRecord: This method is called each time the master changes its position when you are on a master-detail relationship. You should use it to “filter” the data and cache the records that you will need to return on RowCount and GetValue. For example, when the master table moves to record “B”, you should find out here all the records that apply to “B”, and cache them somewhere where RowCount and GetValue can read them.
You should probably create indexes on the data on the constructor of this class, so MoveMasterRecord can be fast finding the information it needs.

You do not need to implement this method if the VirtualDataTable is not going to be used on Master-Detail relationships or Split relationships.

FilteredRowCount: This method returns the total count of records for the current state (similar to RowCount), but, without considering Split conditions.
If the dataset has 200 records, of which only 20 apply for the current master record, and you have a Split of 5 records, RowCount will return 5 and FilteredRowCount will return 20.
This method is used by the Master Split table to know how much records it should have. In the last example, with a FilteredRowCount of 20 and a split every 5 records, you need 4 master records. You do not need to implement this method if you do not want to provide “Split” functionality.

Setting up Relationships

Once the classes have been implemented, you need to consider how to handle the relationships (if you are using your tables on master-detail relationship).

If you do nothing after implementing the tables, the only way to create relationships will be on the template (via Relationship tags on the config sheet).

This will be fine in most cases, but sometimes you will want to create the data model in the code, as you could with normal DataSets.

The reason why you cannot user normal DataRelationships to do so is simple, DataRelationships are part of ADO.NET, they use DataColumn, DataColumns use DataTables, and so on. Since we just created brand new objects without DataTables or DataColumns, you just can't set up DataRelationships on them.

To allow 'design time' relationships, VirtualDataTable offers a ChildRelations property. This is a list of “Master Table/Master Field/Detail Table/Detail Field” entries, where each entry represents a relationship within two tables.

ChildRelations is protected by default, but you can expose it on your classes, or expose methods that internally will add to the ChildRelations list. Whatever relation you add to this list on the code (for example via “MyVirtualDataTable.ChildRelations.Add(MyRelation)”) will be used by FlexCelReport as if it were normal data relation.

Finally

When you have defined both classes, you need to create instances of your VirtualDataTable, and add them to FlexCel with FlexCelReport.AddTable. And that's all folks. For examples of how the whole process is done, please take a look at the Virtual DataSet demo, and also remember to look at the methods reference on FlexCel.chm.