

Part 1 - Using Modifiers and Ordinal Syntax

Table of Contents

- Introduction
 - Getting Started
- Setting Up the Template
 - Using Ordinal Syntax
 - Using Data Marker Modifiers
- Adding an ExcelWriter Reference in Visual Studio
- Writing the Code
 - Data Binding
- Final Code
- Downloads
- Next Steps

Introduction



Following the Sample Code

In the downloadable [ExcelWriter_Basic_Tutorials.zip](#), there is a completed template file located in *CompleteFinancialReport/templates/Part1_Financial_Template.xlsx*.

Getting Started

In this tutorial [ExcelTemplate](#) is being used to populate data and [ExcelApplication](#) is being used to apply some formatting and merge workbooks together. This part of the tutorial will demonstrate how to use of [data marker modifiers](#) and ordinal syntax in ExcelWriter templates.



This example assumes an understanding of [ExcelTemplate](#). If you are not familiar with how to set up an Excel template with data markers, please go through the [Simple Expense Summary](#) first.

Setting Up the Template

Using Ordinal Syntax

Here is the starting template:

This template uses two different data marker modifiers - fieldname and optional. Modifiers are added in parentheses at the end of a data marker.

The *fieldname* modifier shows the field name of the column being bound. The syntax is as follows:

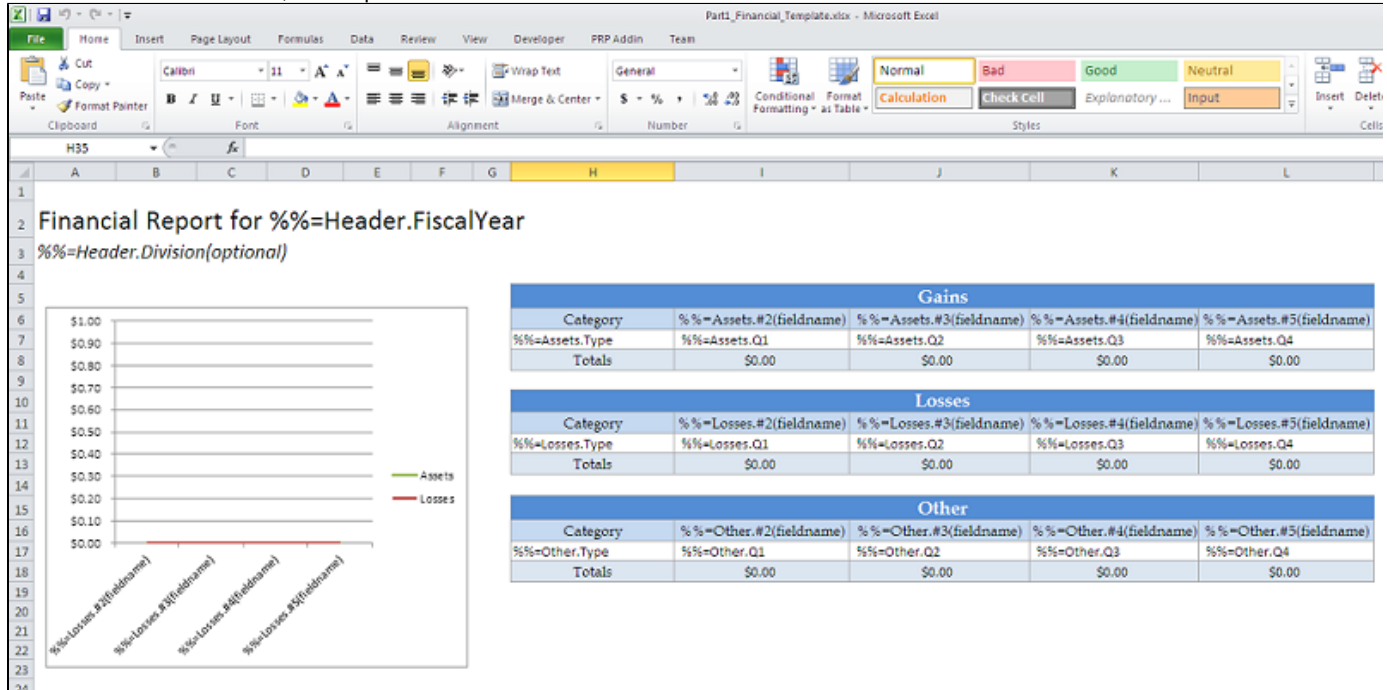
Part1_Financial_Template.xlsx - Microsoft					
File Home Insert Page Layout Formulas Data Review View Developer PRP Addin Team					
Clipboard Font Alignment Number Conditional Formatting as Table					
E30 fx					
	A	B	C	D	E
1	Assets				
2	Category	%%=Assets.#2(fieldname)	%%=Assets.#3(fieldname)	%%=Assets.#4(fieldname)	%%=Assets.#5(fieldname)
3	%%=Assets.Type	%%=Assets.Q1	%%=Assets.Q2	%%=Assets.Q3	%%=Assets.Q4
4	Totals	\$ -	\$ -	\$ -	\$ -
5					
6	Losses				
7	Category	%%=Losses.#2(fieldname)	%%=Losses.#3(fieldname)	%%=Losses.#4(fieldname)	%%=Losses.#5(fieldname)
8	%%=Losses.Type	%%=Losses.Q1	%%=Losses.Q2	%%=Losses.Q3	%%=Losses.Q4
9	Totals	\$ -	\$ -	\$ -	\$ -
10	Other				
11	Category	%%=Other.#2(fieldname)	%%=Other.#3(fieldname)	%%=Other.#4(fieldname)	%%=Other.#5(fieldname)
12	%%=Other.Type	%%=Other.Q1	%%=Other.Q2	%%=Other.Q3	%%=Other.Q4
13	Totals	\$ -	\$ -	\$ -	\$ -
14					
15					
16					
17					

The *optional* modifier allows the data marker to be ignored if there is no data corresponding with that column. This is just an ignore - the column itself will still exist, but the data marker will be skipped. Here is a usage example:

File Home Insert Page Layout Formulas Data Review View Developer							
Clipboard Font Alignment							
O22 fx							
	A	B	C	D	E	F	G
1							
2	Financial Report for %%=Header.FiscalYear						
3	%%=Header.Division(optional)						
4							
5	\$1.00						
6	\$0.90						
7	\$0.80					%%=A	
8	\$0.70						
9	\$0.60						
10	\$0.50						
11							

In the above, the "Division" column will be ignored if no division data is available.

After the modifiers are added, the template should resemble this:



Adding an ExcelWriter Reference in Visual Studio



Following the Sample Code

In the sample code, the reference to *SoftArtisans.OfficeWriter.ExcelWriter.dll* has already been added to the *CompleteFinancialReport* project.

Create a .NET project and add a reference to the ExcelWriter library.

1. Open Visual Studio and create a .NET project.
 - The sample code uses a web application.
2. Add a reference to *SoftArtisans.OfficeWriter.ExcelWriter.dll*
 - *SoftArtisans.OfficeWriter.ExcelWriter.dll* is located under **Program Files > SoftArtisans > OfficeWriter > dotnet > bin**

Writing the Code

1. Include the *SoftArtisans.OfficeWriter.ExcelWriter* namespace in the code behind

```
using SoftArtisans.OfficeWriter.ExcelWriter;
```

2. In the method that will run the report, instantiate the *ExcelTemplate* object.

```
ExcelTemplate XLT = new ExcelTemplate();
```

3. Open the template file with the *ExcelTemplate.Open* method.

```
XLT.Open(Page.MapPath("~/templates//Part1_Financial_Template.xlsx"));
```

4. Create a [DataBindingProperties](#) object. None of the binding properties will be changed for this tutorial, but `DataBindingProperties` is a required parameter in `ExcelTemplate` data binding methods.

```
DataBindingProperties dataProps = XLT.CreateDataBindingProperties();
```

Data Binding



Following the Sample

In the sample project, we are parsing CSV files with query results, rather than querying a live database. The CSV files are available under the `data` directory. There is a copy of the CSV parser, `GenericParsing.dll` in the `bin` directory of the project. `GetCSVData` is defined in `Part1.aspx.cs` in a region marked *Utility Methods*.

If you are following in your own project and would like to parse the CSV files as well, you will need to:

- Add a reference to `GenericParsing.dll`
- Include `GeneringParsing` at the top of your code.
- Add the `GetCSVData` method that can be found in the sample code.

1. Get the data for the *Assets*, *Losses*, and *Other* datasets

These calls are to a helper method `GetCSVData` that parses the CSV files and returns a `DataTable` with the values.

```
DataTable dtAssets = GetCSVData("//data//Assets.csv");  
DataTable dtLosses = GetCSVData("//data//Losses.csv");  
DataTable dtOther = GetCSVData("//data//Other.csv");
```

2. Create the datasets for the header row. Recall the optional modifier for the "Division" tag. This tutorial will not bind any data for that tag to demonstrate the function.

```
//Create the array of header values. This example only binds a single item  
string[] headerValues = { "2011" };  
  
//Create the array of header names.  
string[] headerNames = { "FiscalYear" };
```

3. Use [ExcelTemplate.BindData](#) to bind the data for the *Assets*, *Losses*, and *Other* data sets.

```
XLT.BindData(dtAssets, "Assets", bindingProps);  
XLT.BindData(dtLosses, "Losses", bindingProps);  
XLT.BindData(dtOther, "Other", bindingProps);
```

4. Use the [ExcelTemplate.BindRowData](#) method to bind the header data to the data markers in the template file (i.e. `%%=Header.FiscalYear`).

```
XLT.BindRowData(headerValues, headerNames, "Header", bindingProps);
```

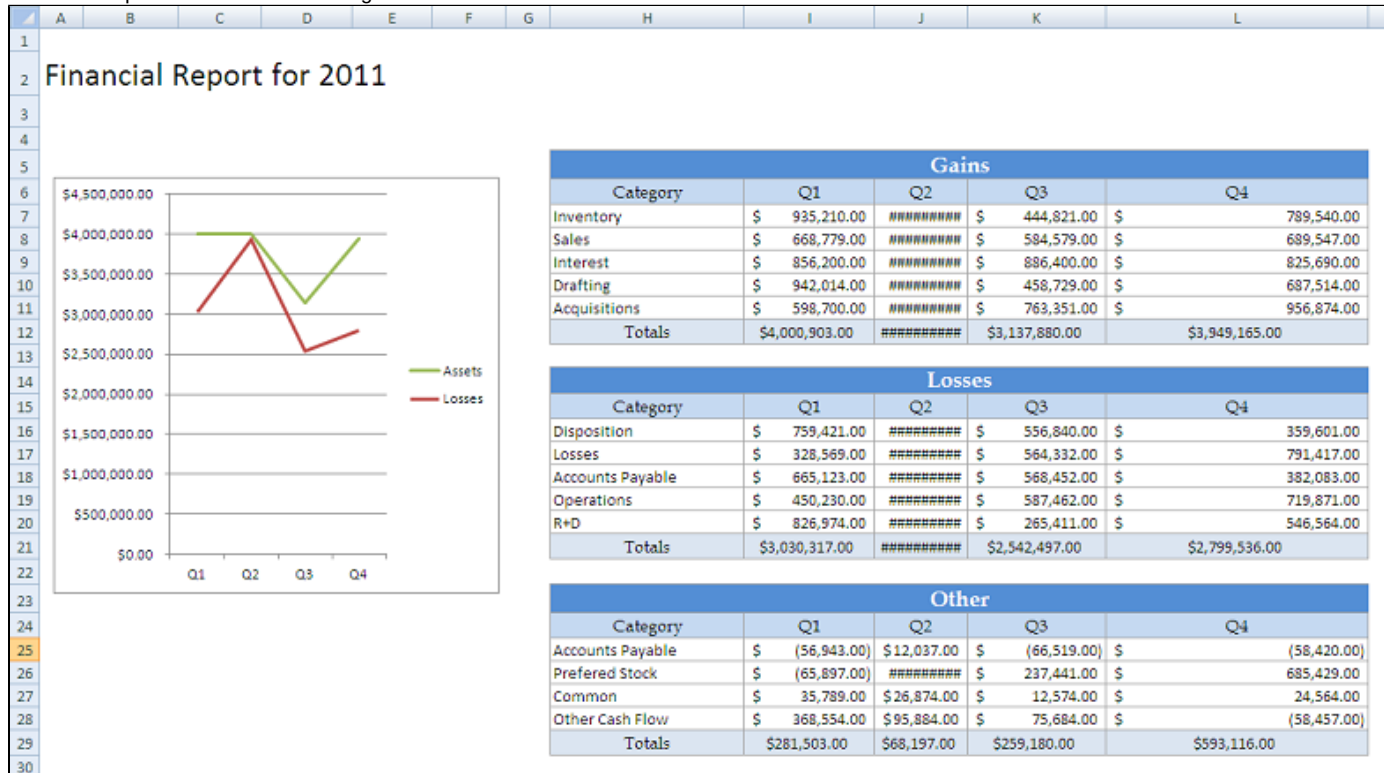
5. Call [ExcelTemplate.Process\(\)](#) to import all data into the file.

```
XLT.Process();
```

6. Call [ExcelTemplate.Save\(\)](#) to save the final output

```
XLT.Save(Page.Response, "temp.xlsx", false);
```

The final output should look something like this:



Final Code

```

using SoftArtisans.OfficeWriter.ExcelWriter;
using GenericParsing;
...

//Instantiate the template object
ExcelTemplate XLT = new ExcelTemplate();

//Open the file
XLT.Open(Page.MapPath("//templates//Part1_Financial_Template.xlsx"));

//Create data binding properties
DataBindingProperties bindingProps = XLT.CreateDataBindingProperties();

//Get the data from the CSVs. More info about the generic parser is available
//in the project and in the tutorial above.
DataTable dtAssets = GetCSVData("//data//Assets.csv");
DataTable dtLosses = GetCSVData("//data//Losses.csv");
DataTable dtOther = GetCSVData("//data//Other.csv");

//Declare the row data. This tutorial uses a single item array to demonstrate the
//optional modifier
string\[\] headerValues = { "2011" };
string\[\] headerNames = { "FiscalYear" };

//Bind each datatable
XLT.BindData(dtAssets, "Assets", bindingProps);
XLT.BindData(dtLosses, "Losses", bindingProps);
XLT.BindData(dtOther, "Other", bindingProps);

//Bind the single row data
XLT.BindRowData(headerValues, headerNames, "Header", bindingProps);

//Call process to import data to file
XLT.Process();

//Save the file
XLT.Save(Page.Response, "temp.xlsx", false);

```

Downloads

You can download the code for the Financial Report here.

- [ExcelWriter_Basic_Tutorials.zip](#)

Next Steps

[Continue to Part 2: Using Styles and Formatting](#)