# Part 1 - Using Modifiers and Ordinal Syntax

**Table of Contents**



## 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

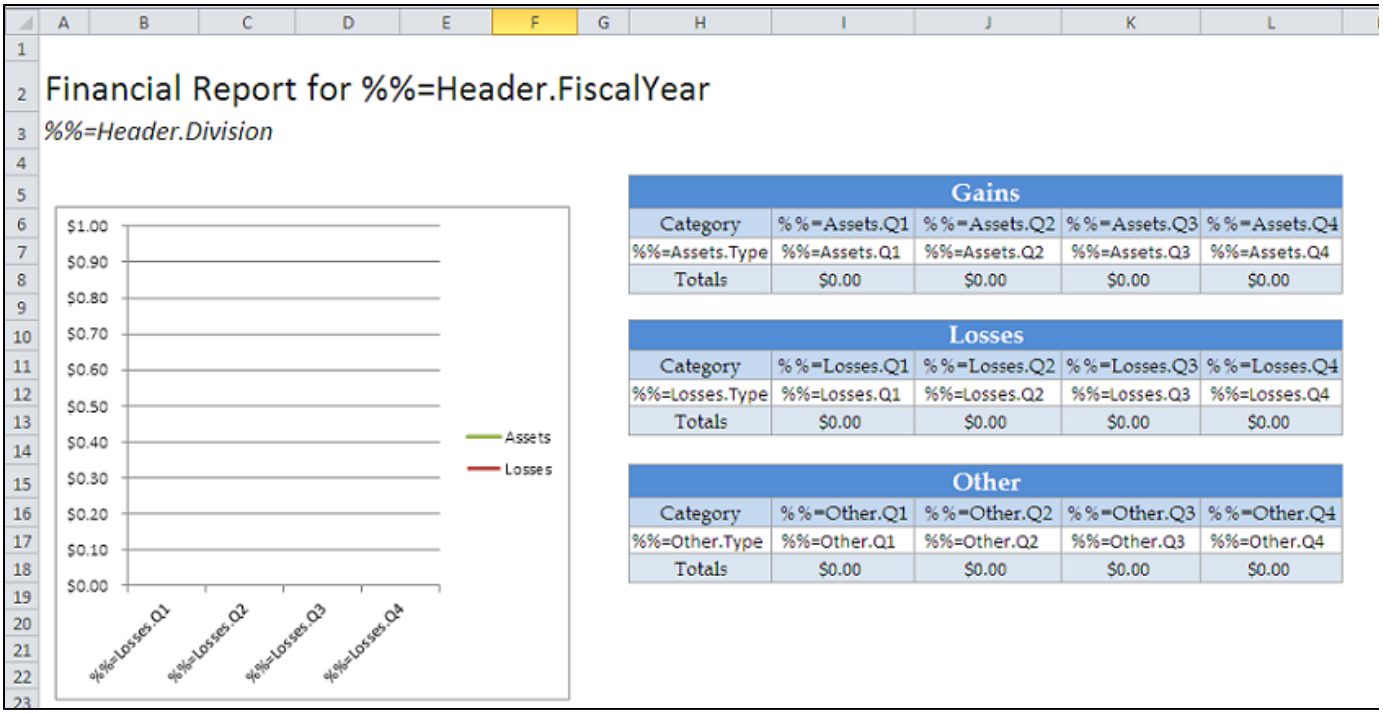
In this tutorial ExcelTemplate is being used to populate data and ExcelApplication is being used to apply some formatting and merge workbooks together. This part of the tutorial will demonstrate how to use of data marker modifiers and ordinal syntax in ExcelWriter templates.

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:

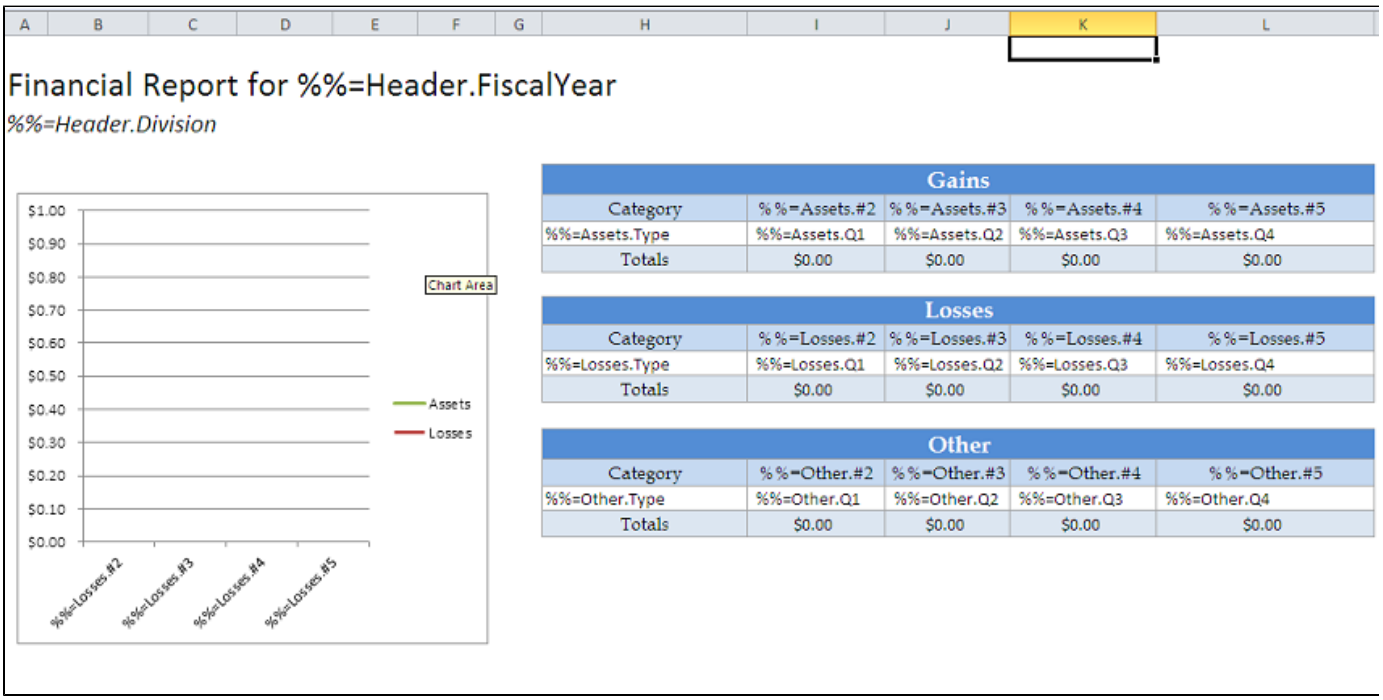



The next few steps will show the process of using ordinal syntax.

ExcelTemplate normally requires the data marker for a column of data to specify the data set and the column name (e.g. %%=DataSource.ColumnName). In the event that the column name or data source name is unknown, ExcelWriter supports the use of ordinal numbering based on the data source column.

Instead of binding to to a specific column name, "%%=Assets.Q1", the column name can be replaced with ordinal syntax, "%%=Assets.#2". This denotes the second column in the 'Assets' data set. The data set name can also be replaced, "%%=#3.#2", which denotes the second column from the third data source bound to the template.

However, in this example, we will only change the column name, as shown below:

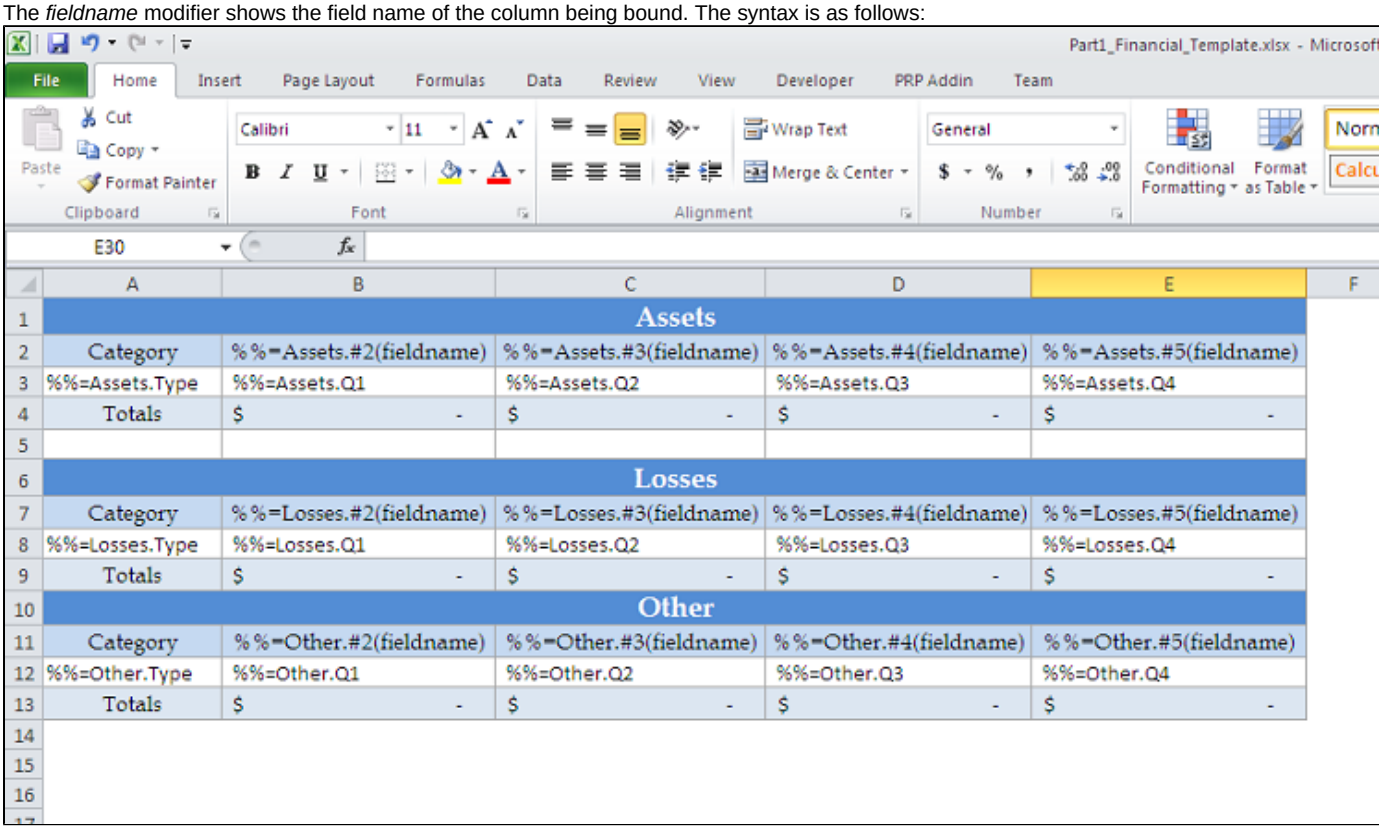



## Using Data Marker Modifiers

In addition to ordinal syntax, ExcelWriter supports *data marker modifiers*, which are tags that change how and what data is imported into the file.

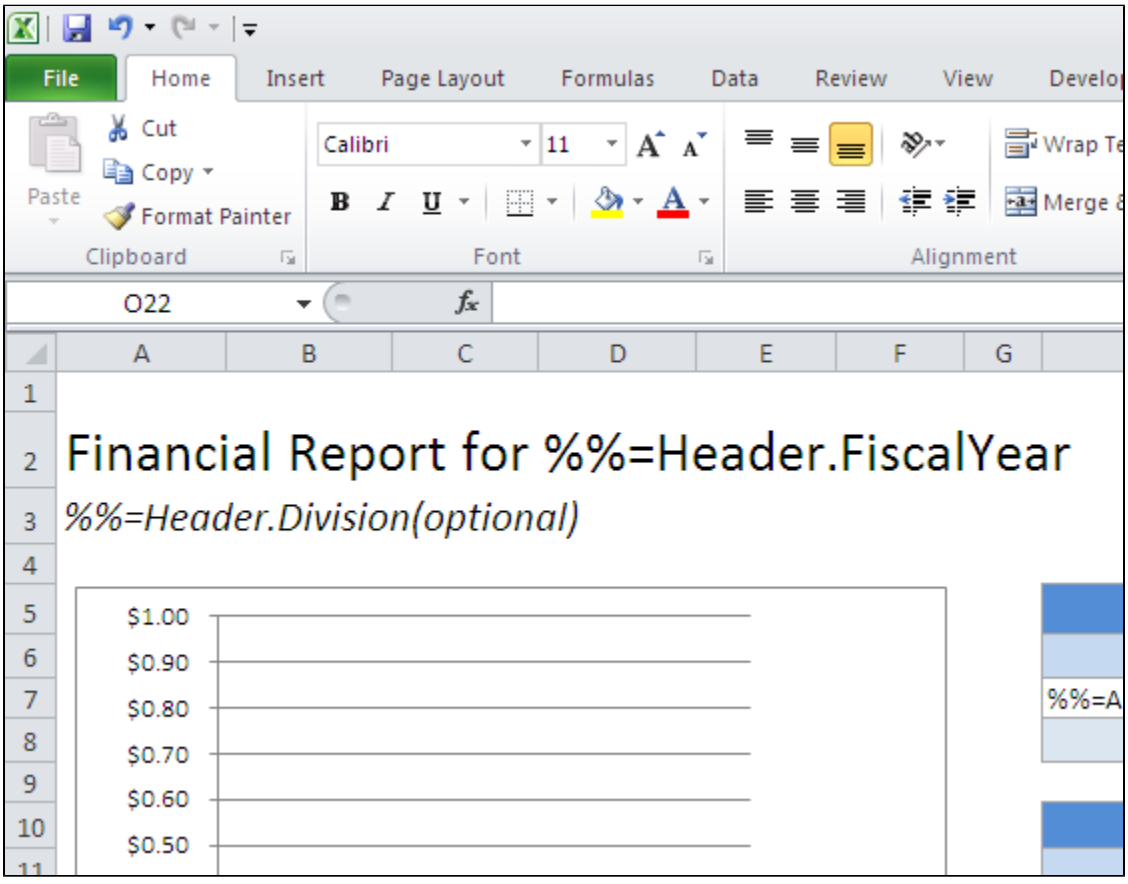
This template uses two different data marker modifiers - fieldname and optional. Modifiers are added in parentheses at the end of a data marker.

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



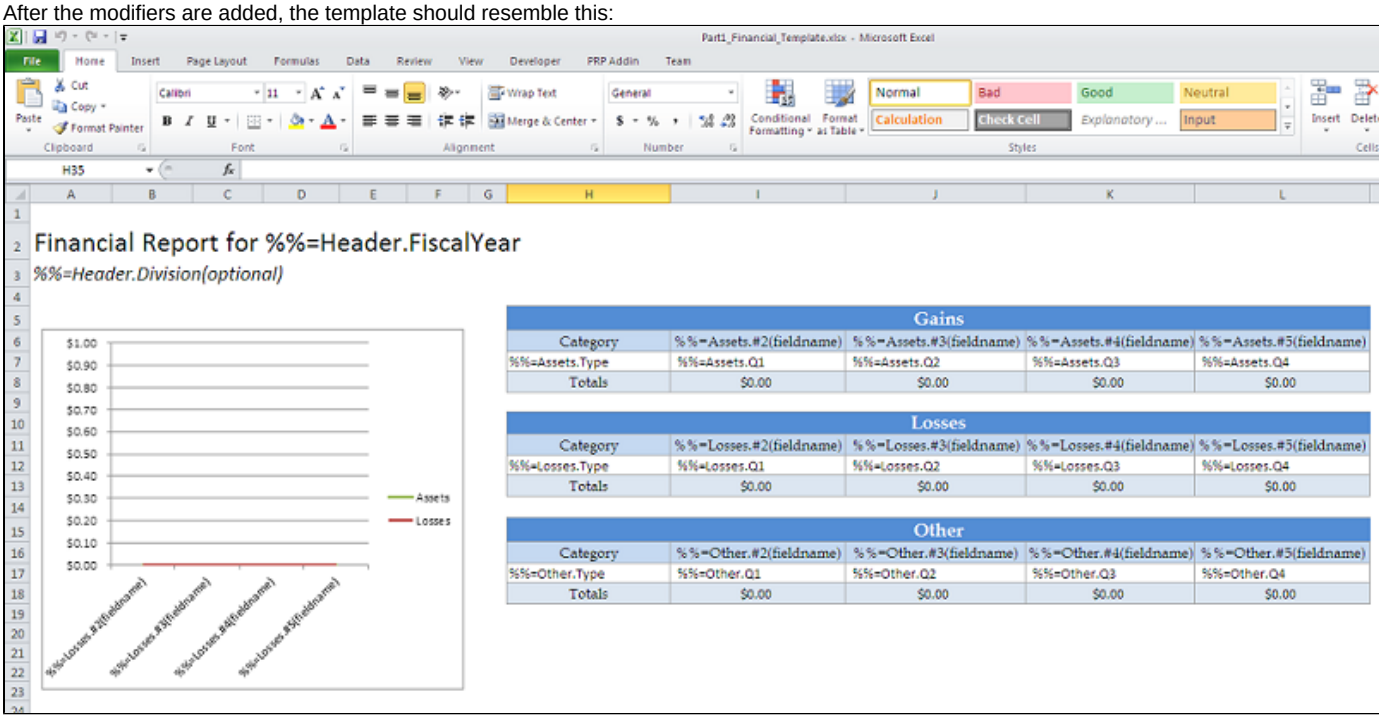
| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | **Assets** | | | | | |
| 2 | Category | %%=Assets.#2(fieldname) | %%=Assets.#3(fieldname) | %%=Assets.#4(fieldname) | %%=Assets.#5(fieldname) | |
| 3 | %%=Assets.Type | %%=Assets.Q1 | %%=Assets.Q2 | %%=Assets.Q3 | %%=Assets.Q4 | |
| 4 | 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 | | | | | | |

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:



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);
```

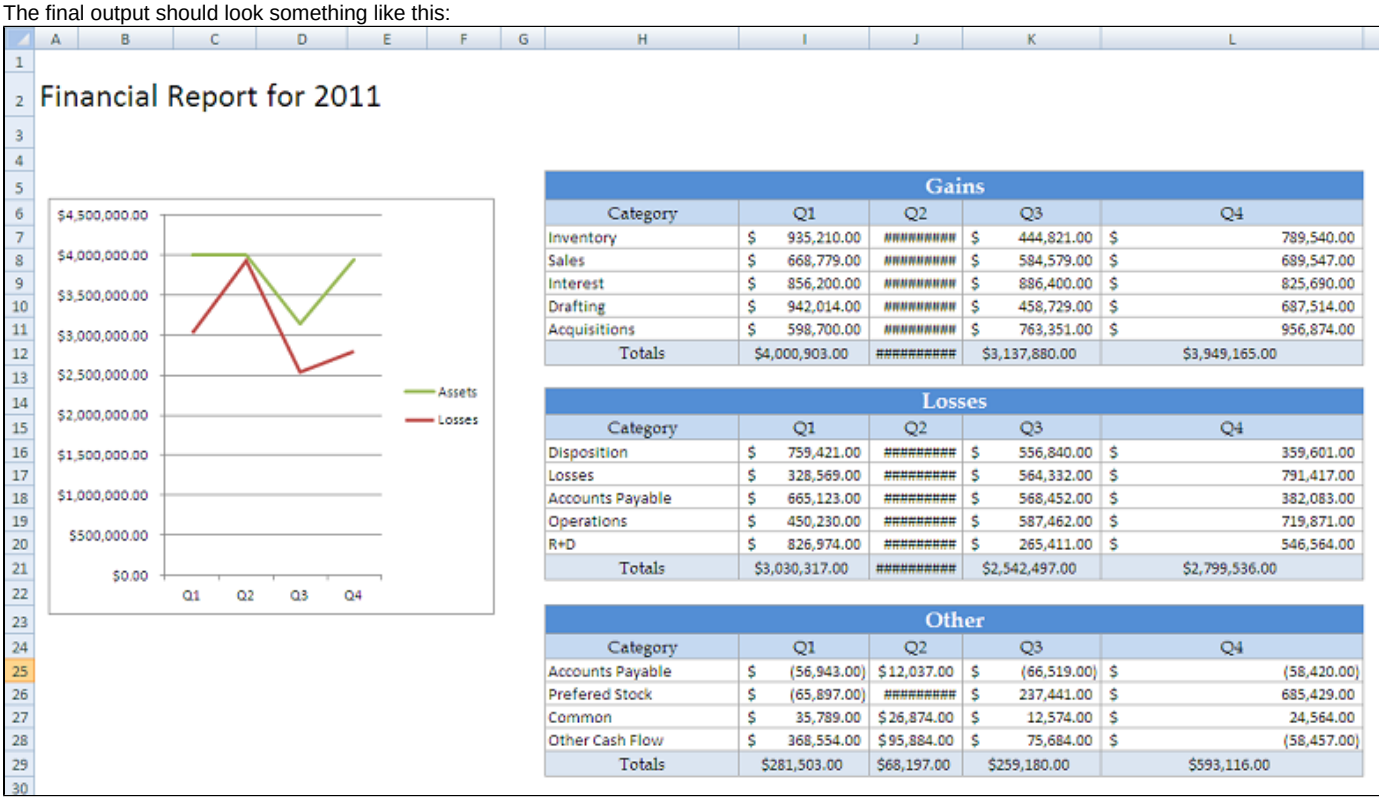
5. Call ExcelTemplate.Process() to import all data into the file.

```
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:



Financial Report for 2011




**Gains**

| Category | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| Inventory | $ 935,210.00 | ########## | $ 444,821.00 | $ 789,540.00 |
| Sales | $ 668,779.00 | ########## | $ 584,579.00 | $ 689,547.00 |
| Interest | $ 856,200.00 | ########## | $ 886,400.00 | $ 825,690.00 |
| Drafting | $ 942,014.00 | ########## | $ 458,729.00 | $ 687,514.00 |
| Acquisitions | $ 598,700.00 | ########## | $ 763,351.00 | $ 956,874.00 |
| Totals | $4,000,903.00 | ########## | $3,137,880.00 | $3,949,165.00 |

**Losses**

| Category | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| Disposition | $ 759,421.00 | ######### | $ 556,840.00 | $ 359,601.00 |
| Losses | $ 328,569.00 | ######### | $ 564,332.00 | $ 791,417.00 |
| Accounts Payable | $ 665,123.00 | ######### | $ 568,452.00 | $ 382,083.00 |
| Operations | $ 450,230.00 | ######### | $ 587,462.00 | $ 719,871.00 |
| R+D | $ 826,974.00 | ######### | $ 265,411.00 | $ 546,564.00 |
| Totals | $3,030,317.00 | ########## | $2,542,497.00 | $2,799,536.00 |

**Other**

| Category | Q1 | Q2 | Q3 | Q4 |
|---|---|---|---|---|
| Accounts Payable | $ (56,943.00) | $12,037.00 | $ (66,519.00) | $ (58,420.00) |
| Prefered Stock | $ (65,897.00) | ######### | $ 237,441.00 | $ 685,429.00 |
| Common | $ 35,789.00 | $26,874.00 | $ 12,574.00 | $ 24,564.00 |
| Other Cash Flow | $ 368,554.00 | $95,884.00 | $ 75,684.00 | $ (58,457.00) |
| Totals | $281,503.00 | $68,197.00 | $259,180.00 | $593,116.00 |

# 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