

Using the Excel Export Helper with SpreadsheetLight

Updated Wednesday, February 01, 2017

By Ricky Donalson



[SpreadsheetLight](https://www.spreadsheetlight.com/)

Table of Contents

Initializing the Exporter	4
Static Implementation Functions:.....	5
OutputWorkbook:.....	5
• (Overload 1) Use DataTable – Output to Web Page.	5
• (Overload 2) Use DataSet – Output to Web Page.....	6
• (Overload 3) Use DataSet – Output to Named Workbook in Specific Location.....	7
Setting up the DataSet	8
Entity Relationship – Four Related Tables:	8
Programmatic Construction:.....	9
Output:.....	10
Entity Relationship – Three Related Tables:	11
Programmatic Construction:.....	12
Output:.....	13
Entity Relationship – Two Related Tables:.....	14
Programmatic Construction:.....	15
Output:.....	16
Entity Relationship – Four Unrelated Tables:	17
Programmatic Construction:.....	18
Output:.....	19
Setting up User-Defined-Settings.....	20
The Settings Class.....	20
Properties.....	20
The ChildSetting Class	21
Properties.....	21
Setting up User-Defined-Columns (UDCs)	24
Properties.....	24
Examples	27
Configuring the Settings Class.....	27
Configuring the ChildSetting Class	27
Parameter Injection	27
Constructor Injection	28
Example 1.....	28
Example 2.....	28
Configuring the Column Class	28
Parameter Injection	28

Constructor Injection	29
Example 1	29
Example 2	29
Putting It All Together	30

Initializing the Exporter

The “Exporter” has static components make it very easy to use.

The helper is found in the “Ups.Toolkit. Ups.Toolkit.SpreadsheetLight” library. Add a reference to this Library or Project file to your project.

Location in TFS: ~\P01A\CS-CTG\WebDev\Project Base - .NET 4.5.2\Ups.Toolkit\Ups.Toolkit.SpreadsheetLight

The helper is flexible in that it will place separate DataTables with no parent-child relation on separate Sheets within the same Workbook, but if there is a relation then the tables will be grouped on the same Sheet. For example, if there are four tables in the DataSet, and two are related and two are not, then the two unrelated tables will appear on separate sheets, while the two related tables will grouped on the same Sheet.

The helper is also flexible in area of styling; you can go all out and really customize the output with User-Defined Columns, and Styling for all sections, or you can just call the base “OutputWorkbook” function and it will just export the data with default styling.

All of the static “OutputWorkbook” function overloads output bytes that will be used by webpage with a “Response” Header in this manner:

First create the Data: In this instance mock data is being used, but you will create a DataTable or DataSet from the output of procedures that are accessed from the in-house DAL

```
// Create the Data
var data = new CreateMockData();
var dataSet = data.CreateDataSet();

// Writing to HTTP Response output Excel content to download from client
Response.Clear ();
Response.ContentType = "application/excel";
Response.AddHeader ("Content-disposition", "filename=ExcelExport.xls");
Exporter.OutputWorkbook(
    Response.OutputStream,
    dataSet,
    "Basic Grouped DS",
    DefaultExcelExportStyles.SetupDefaultStyles(),
    true,
    null
);
Response.OutputStream.Flush();
Response.OutputStream.Close();
Response.Flush();
Response.Close();
```

Static Implementation Functions:

OutputWorkbook:

There are two basic Overload versions of "OutputWorkbook" on that inputs a single DataTable and the other a DataSet with either a single table or multiple related tables. All have the same required & optional parameters. If the DataSet has multiple tables they must be related primary key to foreign key. If there is just one table then it behaves like the DataTable Overload version. If you add a concatenated file name and path then it will out to a file directly.

- (Overload 1) Use DataTable – Output to Web Page.
 - (Required) "outputStream", The input Response.OutputStream;
 - (Required) "dataTable", The input DataTable
 - (Optional) "sheetNames", Your "User-Defined Sheet Names", enter an array of sheet names for number sheets that will be present in your output. If set to null then default names will pulled from; first the "SheetName" property in each of the "User-Defined" child settings, if that is not available then the table name will be used, and if that is not available then it will be set to "output" concatenated with the sheet number.
*** Note: Do not use any of these characters in the sheet names; greater-than or less-than signs (< >), asterisks (*), question marks (?), double quotes ("), vertical bars or pipes (|), colons (:), forward slashes (/) or brackets ([]).*
 - (Optional) "settings", Your "User-Defined Settings", otherwise in the "SalesDashboard" use the default settings generator by calling the "SetupDefaultStyles()" function in the [DefaultExcelExportStyles](#) class in the "App_Code" folder. This global class has the advantage letting you set the overall stylings for the entire web. If settings is set to null then the backup Default Style Generator function "SetupDefaultStyles()" in the [DefaultStyles](#) class will be used.
 - (Optional) "showColumnHeaders", if the parameter is not included then it default to "true". It is also an overall setting that will override the individual child settings for showing column headers.
 - (Optional) "fileNameAndPath", If output is being directed to a specific location and file name, enter the path and Workbook name, then set the "outputStream" to null.

Signature:

```
public static void OutputWorkbook(  
    Stream outputStream,  
    DataTable dataTable,  
    String[] sheetNames = null,  
    Settings settings = null,  
    bool showColumnHeaders = true,  
    string fileNameAndPath = null  
)
```

Sample Code:

```
// Create the Data  
var dataTable = MyDAL.MyDataTable();  
// Create Excel Workbook with Export Helper  
Exporter.OutputWorkbook(  
    Response.OutputStream,  
    dataTable,  
    new[]{"Directors", "Managers-TeamLeads", "Associates" },  
    DefaultExcelExportStyles.SetupDefaultStyles(),  
    true  
);
```

- (Overload 2) Use DataSet – Output to Web Page.
 - (Required) “outputStream” *
 - (Required) “dataSet”, The input DataSet with a single table or multiple related tables.
 - (Optional) “sheetNames” *
 - (Optional) “settings” *
 - (Optional) “showColumnHeaders” *
 - (Optional) “fileNameAndPath”, If output is being directed to a specific location and file name, enter the path and Workbook name, then set the “outputStream” to null.
- * Same as before.

Signature:

```
public static void OutputWorkbook(
    Stream outputStream,
    DataSet dataSet,
    String[] sheetNames = null,
    Settings settings = null,
    bool showColumnHeaders = true,
    string fileNameAndPath = null
)
```

Sample Code:

```
// Create the Data
var dataSet = MyDAL.MyDataSet();
// Create Excel Workbook with Export Helper
ExcelExportHelper.OutputWorkbook(
    Response.OutputStream,
    dataSet,
    new[] { "Custom Sheet Name" },
    DefaultExcelExportStyles.SetupDefaultStyles
);
```

- (Overload 3) Use DataSet – Output to Named Workbook in Specific Location
 - (Required) “outputStream”, In this instance set to null.
 - (Required) “dataSet” *
 - (Optional) “sheetName” *
 - (Optional) “settings” *
 - (Optional) “showColumnHeaders” *
 - (Optional) “fileNameAndPath”, If output is being directed to a specific location and file name, enter the path and Workbook name, then set the “outputStream” to null.
- * Same as before.

Signature:

```
public static void OutputWorkbook(
    Stream outputStream,
    DataSet dataSet,
    string sheetName = null,
    Settings settings = null,
    bool showColumnHeaders = true,
    string fileNameAndPath = null
)
```

Sample Code:

```
var fileNameAndPath = @"C:\\Workbook.xls";
// Create the Data
var dataSet = MyDAL.MyDataSet();
// Create Excel Workbook
ExcelExportHelper.OutputWorkbook(
    null,
    dataSet,
    new[] { "Custom Sheet Name" },
    CustomExcelExportStyles.SetupCustomStyles(),
    true,
    fileNameAndPath
);
```

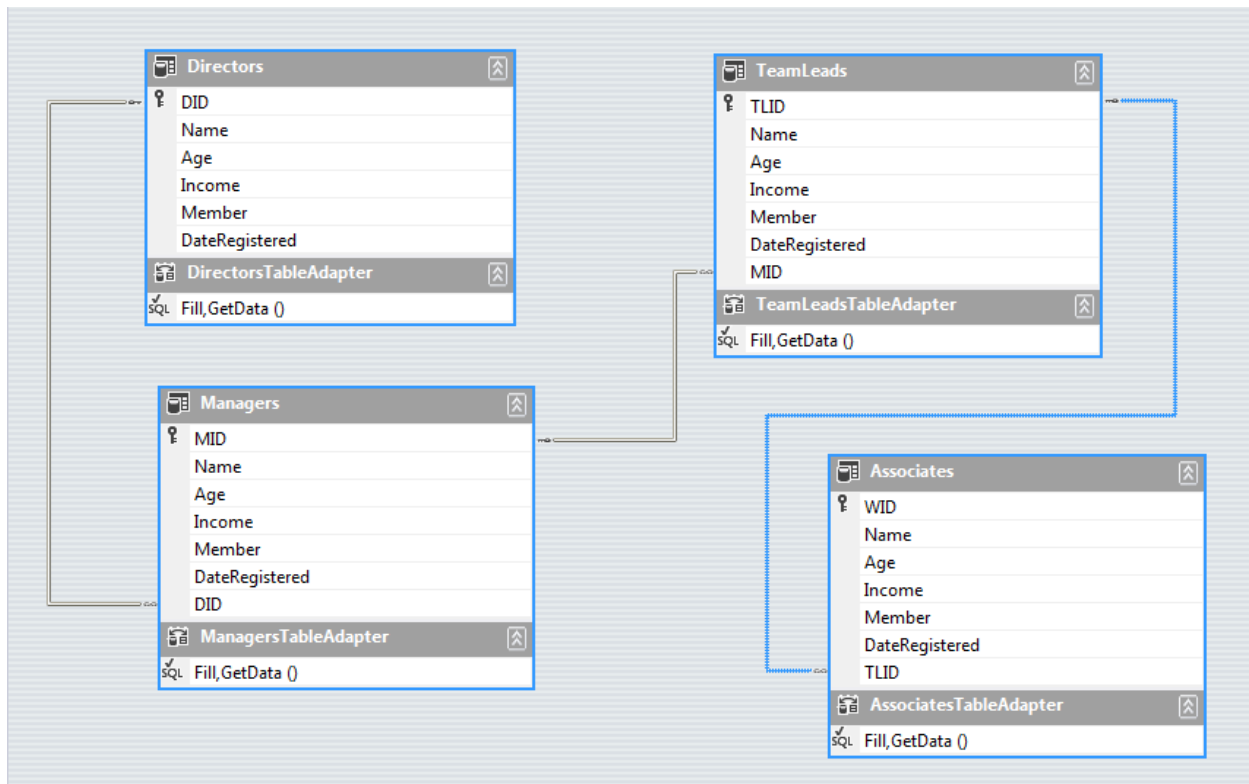
Setting up the DataSet

The DataSet can be created with one or more tables; with a single table the process is very straight forward, but with multiple tables if they have a parent-child association then they must have a relation with a primary key and a foreign key to be grouped. If there is no relation then they will appear on separate Sheets within the same Workbook. In the example used here there are four tables in the sample DataSet; Directors, Managers, Team Leads and Associates. All are related sequentially; the Associates report to the Team Leads, who report to the Managers, who report to the Directors, etc.

Entity Relationship – Four Related Tables:

This example has four related tables Directors, Managers, TeamLeads, & Associates. All four will be grouped on the same Sheet.

This ER diagram illustrates the data structure with the four related tables:



Programmatic Construction:

Here is an example of how the structure should be constructed programmatically:

```
public static DataSet GetMyMockDataSet()
{
    var ds = new DataSet();
    try
    {
        using (var sqlConnection = new
            SqlConnection(ConfigurationManager.ConnectionStrings["MyConnection"].ConnectionString))
        {
            sqlConnection.Open();

            var directors = new SqlDataAdapter("SELECT * FROM Directors", sqlConnection);
            var managers = new SqlDataAdapter("SELECT * FROM Managers", sqlConnection);
            var teamLeads = new SqlDataAdapter("SELECT * FROM TeamLeads", sqlConnection);
            var associates = new SqlDataAdapter("SELECT * FROM Associates", sqlConnection);

            directors.Fill(ds, "Directors");
            managers.Fill(ds, "Managers");
            teamLeads.Fill(ds, "TeamLeads");
            associates.Fill(ds, "Associates");

            ds.Relations.Add("FK_Directors_Managers",
                ds.Tables["Directors"].Columns["DID"],
                ds.Tables["Managers"].Columns["DID"]);

            ds.Relations.Add("FK_Managers_TeamLeads",
                ds.Tables["Managers"].Columns["MID"],
                ds.Tables["TeamLeads"].Columns["MID"]);

            ds.Relations.Add("FK_TeamLeads_Associates",
                ds.Tables["TeamLeads"].Columns["TLID"],
                ds.Tables["Associates"].Columns["TLID"]);
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return ds;
}
```

Output:

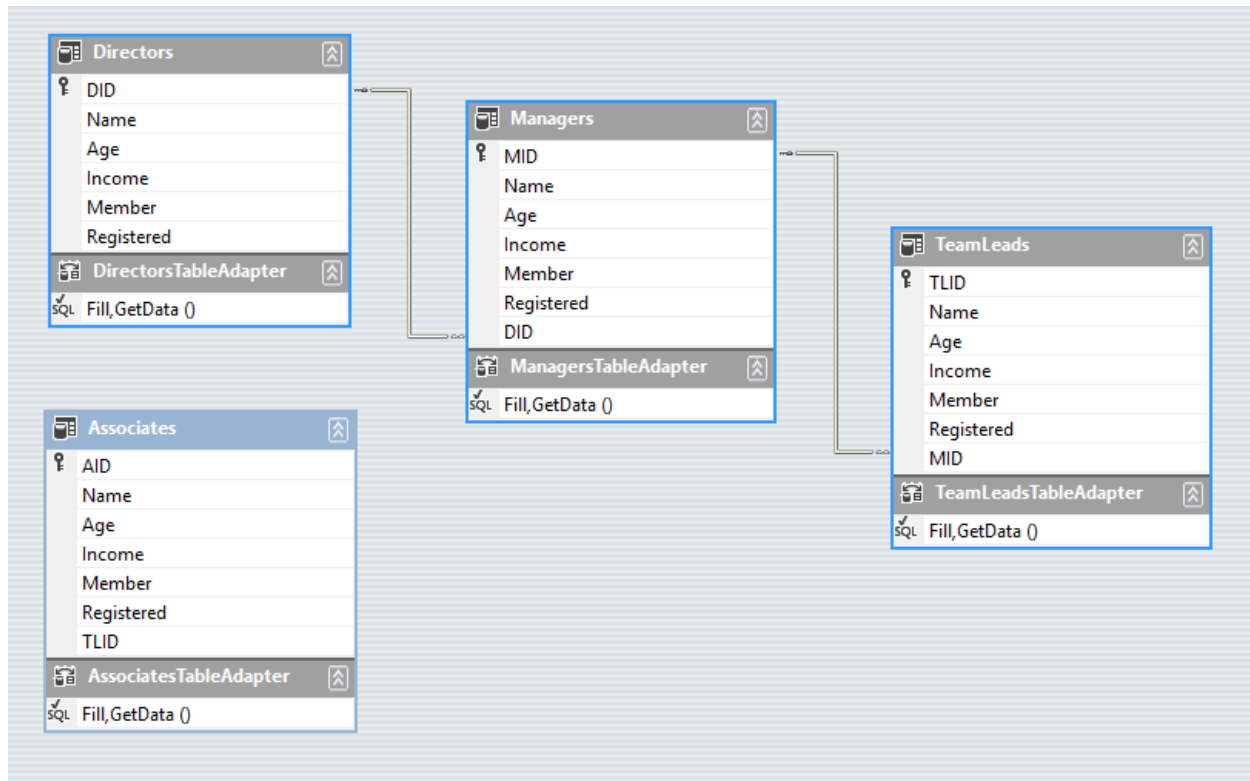
Here is what the output will look like:

Clipboard				Font		Alignment		Number		Formatting		Table	
A1													
										</			

Entity Relationship – Three Related Tables:

This example has four tables with three that are related Directors, Managers and TeamLeads. The unrelated table Associates gets its own sheet, the three related will be grouped on one sheet.

This ER diagram illustrates the data structure with the three related tables with one unrelated:



Programmatic Construction:

Here is an example of how the structure should be constructed programmatically:

```
public static DataSet GetMyMockDataSet()
{
    var ds = new DataSet();
    try
    {
        using (var sqlConnection = new
            SqlConnection(ConfigurationManager.ConnectionStrings["MyConnection"].ConnectionString))
        {
            sqlConnection.Open();

            var directors = new SqlDataAdapter("SELECT * FROM Directors", sqlConnection);
            var managers = new SqlDataAdapter("SELECT * FROM Managers", sqlConnection);
            var teamLeads = new SqlDataAdapter("SELECT * FROM TeamLeads", sqlConnection);
            var associates = new SqlDataAdapter("SELECT * FROM Associates", sqlConnection);

            directors.Fill(ds, "Directors");
            managers.Fill(ds, "Managers");
            teamLeads.Fill(ds, "TeamLeads");
            associates.Fill(ds, "Associates");

            ds.Relations.Add("FK_Directors_Managers ",
                ds.Tables["Directors"].Columns["DID"],
                ds.Tables["Managers"].Columns["DID"]);

            ds.Relations.Add("FK_Managers_TeamLeads",
                ds.Tables["Managers"].Columns["MID"],
                ds.Tables["TeamLeads"].Columns["MID"]);
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return ds;
}
```

Output:

Here is what the output will look like:

A1

:

X

✓

fx

Director

1

2

3

A

B

C

D

E

F

G

H

1

Director

Chronology

Compensation

Member ?

Date Registered

2

Allen

30

\$

100,000.00

TRUE

30-Jan-17

3

Bill

30

\$

100,000.00

TRUE

30-Jan-17

4

Managers

Age

Compensation

Date Registered

5

Sam

30

\$100,000

1/30/2017

6

Team Leads

How Old?

Registration Date

Member?

Income

Team Lead ID

7

Mary

30

30-Jan-17

TRUE

\$ 100,000.00

1

8

Peter

30

30-Jan-17

TRUE

\$ 100,000.00

2

9

Andrew

30

\$100,000

1/30/2017

13

Markus

30

\$

100,000.00

TRUE

30-Jan-17

21

Thomas

30

\$

100,000.00

TRUE

30-Jan-17

24

25

Directors

Associates

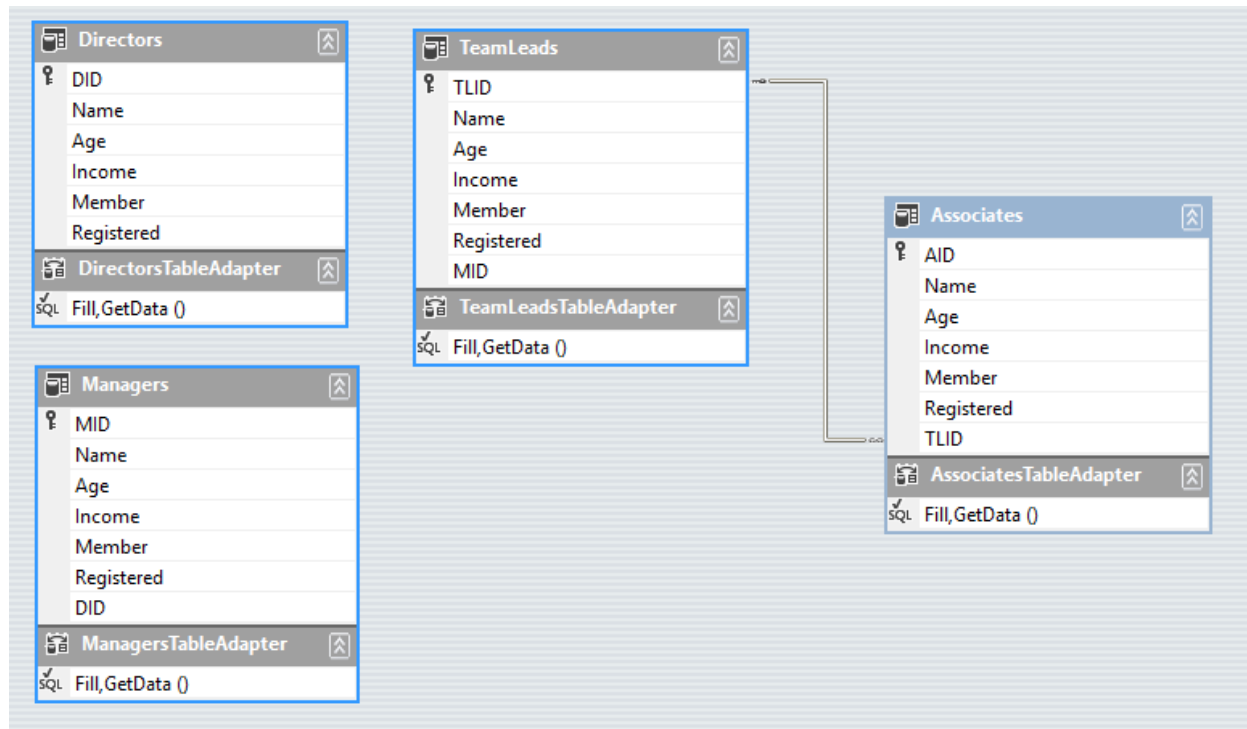
+

A1												:			Associate											
	A	B	C	D	E	F	G	H	I	J																
1	Associate	Member?	Date																							
2	Dan	TRUE	2017/01/30 11:48:50																							
3	Samuel L	TRUE	2017/01/30 11:48:50																							
4	Samuel P	TRUE	2017/01/30 11:48:50																							
5	Samuel D	TRUE	2017/01/30 11:48:50																							
6	Kyle A	TRUE	2017/01/30 11:48:50																							
7	Kyle B	TRUE	2017/01/30 11:48:50																							
8	Kyle C	TRUE	2017/01/30 11:48:50																							
9	Kyle D	TRUE	2017/01/30 11:48:50																							
10	Kyle E	TRUE	2017/01/30 11:48:50																							
11	Kyle F	TRUE	2017/01/30 11:48:50																							
12	Kyle G	TRUE	2017/01/30 11:48:50																							
13	Kyle H	TRUE	2017/01/30 11:48:50																							
14																										
												Directors Associates														

Entity Relationship – Two Related Tables:

This example has four tables with two that are related TeamLeads & Associates. The unrelated tables Directors and Managers get their own sheets, the two related will be grouped on one sheet.

This ER diagram illustrates the data structure:



Programmatic Construction:

Here is an example of how the structure should be constructed programmatically:

```
public static DataSet GetMyMockDataSet()
{
    var ds = new DataSet();
    try
    {
        using (var sqlConnection = new
            SqlConnection(ConfigurationManager.ConnectionStrings["MyConnection"].ConnectionString))
        {
            sqlConnection.Open();

            var directors = new SqlDataAdapter("SELECT * FROM Directors", sqlConnection);
            var managers = new SqlDataAdapter("SELECT * FROM Managers", sqlConnection);
            var teamLeads = new SqlDataAdapter("SELECT * FROM TeamLeads", sqlConnection);
            var associates = new SqlDataAdapter("SELECT * FROM Associates", sqlConnection);

            directors.Fill(ds, "Directors");
            managers.Fill(ds, "Managers");
            teamLeads.Fill(ds, "TeamLeads");
            associates.Fill(ds, "Associates");

            ds.Relations.Add("FK_TeamLeads_Associates ",
                ds.Tables["TeamLeads"].Columns["TLID"],
                ds.Tables["Associates"].Columns["ALID"]);
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return ds;
}
```

Output:

Here is what the output will look like:

	A	B	C	D	E	F	G
1	Director	Chronology	Compensation	Member ?	Date Registered		
2	Allen	30	\$ 100,000.00	TRUE	30-Jan-17		
3	Bill	30	\$ 100,000.00	TRUE	30-Jan-17		
4	Markus	30	\$ 100,000.00	TRUE	30-Jan-17		
5	Thomas	30	\$ 100,000.00	TRUE	30-Jan-17		
6							
7							
8							
9							
10							
11							
12							
13							
14							

Directors Managers TeamLeads +

	A	B	C	D	E	F	G
1	Managers	Age	Compensation	Date Registered			
2	Sam	30	\$100,000	1/30/2017			
3	Andrew	30	\$100,000	1/30/2017			
4	Martha	30	\$100,000	1/30/2017			
5	Sonja	30	\$100,000	1/30/2017			
6	Joe	30	\$100,000	1/30/2017			
7							
8							
9							
10							
11							
12							
13							
14							

Directors Managers TeamLeads +

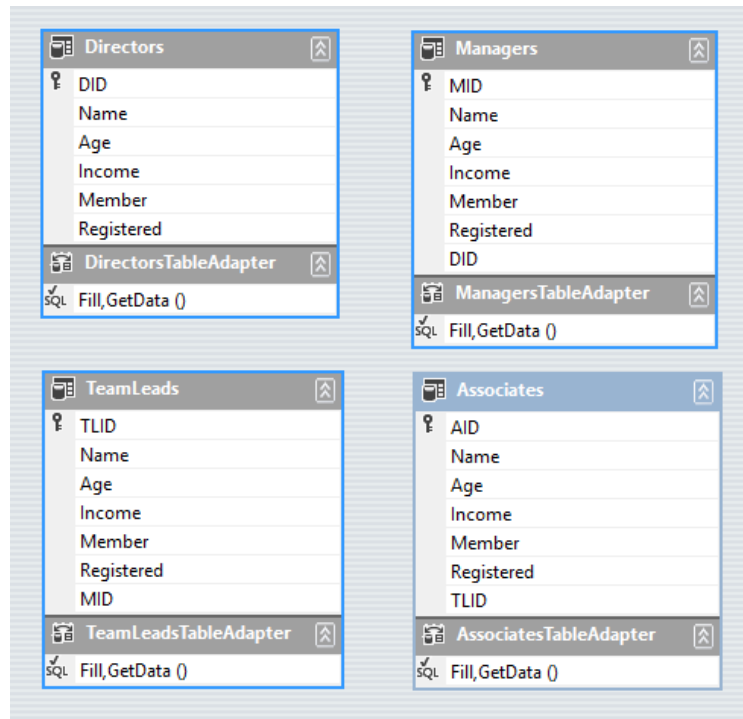
1	2	A	B	C	D	E	F	G
1		Team Leads	How Old?	Registration Date	Member?	Income	Team Lead ID	
2		Mary	30	30-Jan-17	TRUE	\$ 100,000.00	1	
3		Peter	30	30-Jan-17	TRUE	\$ 100,000.00	2	
4		Authur	30	30-Jan-17	TRUE	\$ 100,000.00	3	
5					Associate	Member?	Date	
6					Dan	TRUE	2017/01/30 13:28:20	
7					Samuel L	TRUE	2017/01/30 13:28:20	
8					Samuel P	TRUE	2017/01/30 13:28:20	
9					Samuel D	TRUE	2017/01/30 13:28:20	
10		Willa	30	30-Jan-17	TRUE	\$ 100,000.00	4	
11		Jack	30	30-Jan-17	TRUE	\$ 100,000.00	5	
12		Ann	30	30-Jan-17	TRUE	\$ 100,000.00	6	
22								
23								

Directors Managers TeamLeads +

Entity Relationship – Four Unrelated Tables:

This example has four unrelated tables Directors, Managers, TeamLeads and Associates. All four will go on different sheets.

This ER diagram illustrates the data structure:



Programmatic Construction:

Here is an example of how the structure should be constructed programmatically:

```
public static DataSet GetMyMockDataSet()
{
    var ds = new DataSet();
    try
    {
        using (var sqlConnection = new
            SqlConnection(ConfigurationManager.ConnectionStrings["MyConnection"].ConnectionString))
        {
            sqlConnection.Open();

            var directors = new SqlDataAdapter("SELECT * FROM Directors", sqlConnection);
            var managers = new SqlDataAdapter("SELECT * FROM Managers", sqlConnection);
            var teamLeads = new SqlDataAdapter("SELECT * FROM TeamLeads", sqlConnection);
            var associates = new SqlDataAdapter("SELECT * FROM Associates", sqlConnection);

            directors.Fill(ds, "Directors");
            managers.Fill(ds, "Managers");
            teamLeads.Fill(ds, "TeamLeads");
            associates.Fill(ds, "Associates");
        }
    }
    catch (Exception ex)
    {
        throw ex;
    }
    return ds;
}
```

Output:

Here is what the output will look like:

	A	B	C	D	E	F
1	Director	Chronology	Compensation	Member ?	Date Registered	
2	Allen	30	\$ 100,000.00	TRUE	30-Jan-17	
3	Bill	30	\$ 100,000.00	TRUE	30-Jan-17	
4	Markus	30	\$ 100,000.00	TRUE	30-Jan-17	
5	Thomas	30	\$ 100,000.00	TRUE	30-Jan-17	
6						
7						
8						

Directors Managers TeamLeads Associates +

	A	B	C	D	E	F
1	Managers	Age	Compensation	Date Registered		
2	Sam	30	\$100,000	1/30/2017		
3	Andrew	30	\$100,000	1/30/2017		
4	Martha	30	\$100,000	1/30/2017		
5	Sonja	30	\$100,000	1/30/2017		
6	Joe	30	\$100,000	1/30/2017		
7						
8						

Directors **Managers** TeamLeads Associates

	A	B	C	D	E	F
1	Team Leads	How Old?	Registration Date	Member?	Income	Team Lead ID
2	Mary	30	30-Jan-17	TRUE	\$ 100,000.00	1
3	Peter	30	30-Jan-17	TRUE	\$ 100,000.00	2
4	Authur	30	30-Jan-17	TRUE	\$ 100,000.00	3
5	Willa	30	30-Jan-17	TRUE	\$ 100,000.00	4
6	Jack	30	30-Jan-17	TRUE	\$ 100,000.00	5
7	Ann	30	30-Jan-17	TRUE	\$ 100,000.00	6
8						

Directors Managers **TeamLeads** Associates +

	A	B	C	D	E	F
1	Associate	Member ?	Date			
2	Dan	TRUE	2017/01/30 13:50:53			
3	Samuel L	TRUE	2017/01/30 13:50:53			
4	Samuel P	TRUE	2017/01/30 13:50:53			
5	Samuel D	TRUE	2017/01/30 13:50:53			
6	Kyle A	TRUE	2017/01/30 13:50:53			
7	Kyle B	TRUE	2017/01/30 13:50:53			
8	Kyle C	TRUE	2017/01/30 13:50:53			
9	Kyle D	TRUE	2017/01/30 13:50:53			
10	Kyle E	TRUE	2017/01/30 13:50:53			
11	Kyle F	TRUE	2017/01/30 13:50:53			
12	Kyle G	TRUE	2017/01/30 13:50:53			
13	Kyle H	TRUE	2017/01/30 13:50:53			
14						
15						

Directors Managers TeamLeads **Associates**

Setting up User-Defined-Settings

It comprised of a Settings class which serves primarily as container for the ChildSetting classes which are setup and injected.

The easiest way is to just clone the [CustomExcelExportStyles](#) class in the SLEXPtest.aspx code-behind page located here:

```
~\p01a\cs-ctg\dashboard\main\salesdashboard\upscustomersolutionsweb\support\slexporttest.aspx.cs
```

Then modify it as needed.

For the static “OutputWorkbook” DataSet overload function, there should be a Settings class that contains a ChildSetting class for each table in the DataSet. Each ChildSetting class will contain the styling and formatting for its corresponding table and injected into the Settings class in the order the tables are related to each other.

For our purposes there are four data groups in the DataSet; Directors, Managers, Team Leads and Associates. All are related sequentially. The Associates report to the Team Leads who report to the Managers who report to the Directors, etc.

**** Note:** Parents should always come before the children, so in our example here; the *Directors* will be first [table\[0\]](#), *Managers* the second [table\[1\]](#), *Team Leads* the third [table\[2\]](#), *Associates* the fourth [table\[3\]](#), and so forth.

Each table of data that is to be displayed must have a ChildSetting class instantiated and injected into the Settings class. The base Table is Directors and it will appear in the Settings class as ChildSetting[0], Managers will be ChildSetting[1], TeamLeads are ChildSetting[2], and Associates ChildSetting[3], etc.

If it is the “OutputWorkbook” that inputs a DataTable then there should be Settings class and a one ChildSetting class

**** Note:** If you’re going to use a lot of the optional setting features then is it recommended that you use Property injection, while Constructor injection is also available it is limited to the most common cases of parameter input.

The Settings Class

It has the properties to set a Custom Title on your Excel Sheet in needed, but serves primarily as container for the ChildSetting classes, which will have the style properties for each tables to be displayed.

Properties

- Name (Optional) – The name for the container.

```
public string Name
```

Sample Code:

```
// Set the Title row height  
Name = "Organization Structure",
```

- ChildSettings (Required) – List of ChildSetting classes. For each table there must be a There should always be at least one ChildSetting injected in.

```
public List<ChildSetting> ChildSettings
```

Sample Code:

```
// Inject the list  
ChildSettings = new List<ChildSetting>();
```

The ChildSetting Class

Here is where you set the properties and styles for each of the data groups (tables) as want to see them on the Excel Sheet. They will follow the same formatting configurations of the Titles discussed previously in the Settings class.

Properties

- Name (Optional) – The name for the container.

`Public string Name`

Sample Code:

```
// Set the child name
```

```
Name = "Directors",
```

- ShowColumnHeader (Required) – True/False: Show the column headers. If not set, defaults to `true`; ** Will be overridden by a `false` value in the “showColumnHeaders” parameter of the “OutputWorkbook” functions. If that value is `false` then Column Headers will not be shown in any of the data groups. If `true` then control is passed back to the individual ChildSetting Classes

`public bool ShowColumnHeader`

Sample Code:

```
// Show the column headers for this data group
```

```
ShowColumnHeader = true,
```

- ColumnOffsets (Optional) – Set the number of columns you want offset the data group to right. Intended to indent child data groups to make them stand out. If not set will default to 0.

`public int ColumnOffsets`

Sample Code:

```
// Indent this data group one column to the right
```

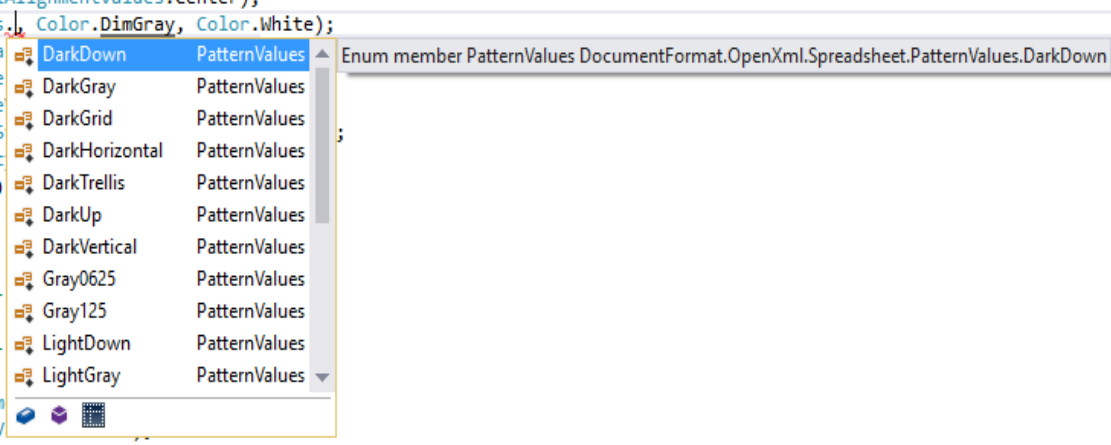
```
ColumnOffsets = 1,
```

- ColumnHeaderStyle (Required) – Setup all the properties required, then for subsequent headers clone it and make only the changes required for the next data group headers

`public SLStyle ColumnHeaderStyle`

DOCUMENTATION FOR SLSTYLE CLASS IN THE SPREADSHEETLIGHT DOCUMENTATION & SAMPLE CODES. SLSTYLE IS A MEMBER OF THE DOCUMENTFORMAT.OPENXML.SPREADSHEET LIBRARY THAT SPREADSHEETLIGHT IS BUILT FROM. FOR ALL ENUMS TYPING A PERIOD AFTER THE ENUM NAME WILL BRING UP A DROPDOWN OF ITEMS TO SELECT

```
ignment(VerticalAlignmentValues.Center);
rn(PatternValues, Color.DimGray, Color.White);
er(BorderStyleValue, DarkDown, PatternValues);
BorderStyleValue, DarkGray, PatternValues);
rder(BorderStyleValue, DarkGrid, PatternValues);
htBorder(BorderStyleValue, DarkHorizontal, PatternValues);
tBorder(BorderStyleValue, DarkTrellis, PatternValues);
annic Bold", 12), DarkUp, PatternValues);
Color.White); DarkVertical, PatternValues);
ue); Gray0625, PatternValues);
child datasets Gray125, PatternValues);
LightDown, PatternValues);
LightGray, PatternValues);
HorizontalAlignm
rticalAlignmentV
Values.Solid, Color.White, Color.Black);
```



Sample Code:

```
// Setup the style for Column Headers
var header = new SLStyle();
header.SetHorizontalAlignment(HorizontalAlignmentValues.Center);
header.SetVerticalAlignment(VerticalAlignmentValues.Center);
header.Fill.SetPattern(PatternValues.Solid, Color.DimGray, Color.White);
header.SetBottomBorder(BorderStyleValues.Medium, Color.Black);
header.SetTopBorder(BorderStyleValues.Medium, Color.Black);
header.SetVerticalBorder(BorderStyleValues.Medium, Color.Black);
header.Border.SetRightBorder(BorderStyleValues.Medium, Color.Black);
header.Border.SetLeftBorder(BorderStyleValues.Medium, Color.Black);
header.SetFont("Britannic Bold", 12);
header.SetFontColor(Color.White);
header.SetFontBold(true);
```

ColumnHeaderStyle = header,

- ColumnHeaderRowHeight (Optional) – Enter a numeric integer value representing the number of pixels you want the rows height to be. If not set, will default to `null`.

```
public short ColumnHeaderRowHeight
```

Sample Code:

```
// Set the column header row height of 25px
ColumnHeaderRowHeight = 25,
```

- ShowAlternatingRows (Optional) – True/False: Show different formatting for odd and even row. If set to false then the odd row style will be overall row style. If not set, will default to `false`.

```
public bool ShowAlternatingRows
```

Sample Code:

```
// Use the OddRowStyle formatting for both odd and even rows
ShowAlternatingRows = false,
```

- OddRowStyle (Required) – There are many more style settings available, in this case only four are used.

```
public SLStyle OddRowStyle
```

Sample Code:

```
// Setup the style for odd rows
var odd = new SLStyle();
odd.SetHorizontalAlignment(HorizontalAlignmentValues.Left);
odd.SetVerticalAlignment(VerticalAlignmentValues.Center);
odd.Fill.SetPattern(PatternValues.Solid, Color.White, Color.Black);
odd.SetFont("Helvetica", 10);
odd.SetFontColor(Color.Black);
```

OddRowStyle = odd,

- EvenRowStyle (Optional) – Set in the same manner as the odd. However here if the “ShowAlternatingRows” is set to false then the EvenRowStyle isn’t needed.

```
public CSSStyle EvenRowStyle
```

Sample Code:

```
// Setup the style for even rows
// Easiest method is to clone the odd style then make only changes that are needed
var even = odd.Clone();
even.Fill.SetPattern(PatternValues.Solid, Color.WhiteSmoke, Color.Black);
```

```
EvenRowStyle = even,
```

- UserDefinedColumns (Optional) – Create a list object of predefined Column classes and inject them in. If not set then the column headers from the data group will be used by default. For more information on User-Defined Columns review the .section on [Setting up User-Defined-Columns \(UDCs\)](#).

```
public List<Column> UserDefinedColumns
```

Sample Code:

```
// Setup a List of Column classes for the UserDefinedColumns. In this case you must “new up” a List to inject
UserDefinedColumns = new List<Column>
{
    new Column("TLID", "Team Lead ID",
        NumberFormats.General, HorizontalAlignmentValues.Left,
        true, 6),
    new Column("Registered", "Registration Date",
        NumberFormats.UserDefined, HorizontalAlignmentValues.Center,
        true, 2, "d-mmm-yy"),
    new Column("Name", "Team Leads",
        NumberFormats.General, HorizontalAlignmentValues.Left,
        true, 0),
    new Column("Age", "How Old?",
        NumberFormats.General, HorizontalAlignmentValues.Center,
        true, 1),
    new Column("Member", "Member?",
        NumberFormats.General, HorizontalAlignmentValues.Center,
        true, 3),
    new Column("Income", "Income",
        NumberFormats.Accounting2Red, HorizontalAlignmentValues.Right,
        true, 4),
    new Column("MID", "Foreign Key",
        NumberFormats.General, HorizontalAlignmentValues.Right,
        False)
};
```

Setting up User-Defined-Columns (UDCs)

The purpose for doing this is to override the default column names from the input data, resetting their column display name, visibility, order, number formatting and order.

**** Note:** If you're going to use a lot of the optional setting features then it is recommended that you use Property injection, while Constructor injection is also available it is limited to the most common cases of parameter input.

Properties

- "BoundColumnName" (Required) – Enter the Bound column name from the DataTable
`public string BoundColumnName`

Sample Code:

```
// Enter the exact data group column header you want to change.  
BoundColumnName = "Registered",
```

- "UserDefinedColumnName" (Required) – Enter a Custom Column name that will replace the Bound Field name. If left blank if then the bound name will be the column header value.
`public string UserDefinedColumnName`

Sample Code:

```
// Enter the name you want for the column header  
UserDefinedColumnName = "Date Registered",
```

- "NumberFormat" (Optional) – Select a NumberFormats Enum value for a Data Formatting. Once a value is selected then hover the mouse over the item and intellisense will display all the information about it. Here is the list of the Enum items available:
 - Misc.
 - "UserDefined" If the user wants to use a custom format, then there must be a valid excel format entered in the "UserDefinedNumberFormat" field.
 - "General" 36000.1234 / -45 number with no commas; negative values are preceded by a minus sign and text as text.
 - Decimal – Set number of decimal places with commas
 - "Decimal0" 42,050
 - "Decimal1" 42,050.2
 - "Decimal2" 42,050.23
 - "Decimal3" 42,050.233
 - "Decimal-4" 42,050.2334
 - Percent – Input values must be decimal values between 0-1, formatting with a set number of decimals
 - "Percent0" 89%
 - "Percent1" 89.3%
 - "Percent2" 89.25%
 - "Percent3" 89.251%
 - "Percent4" 89.2514%
 - Currency
 - "Currency0Black" \$36,000 / (\$45) "Black" – 0 decimal places; negative values are "Black" in parentheses
 - "Currency0Red" \$36,000 / (\$45) "Red" – 0 decimal places; negative values are "Red" in parentheses
 - "Currency2Black" \$36,000.12 / (\$45.00) "Black" – 2 decimal places; negative values are "Black" in parentheses
 - "Currency2Red" \$36,000.12 / (\$45.00) "Red" – 2 decimal places; negative values are "Red" in parentheses.

- Accounting – Same as the Currency format but with the \$ sign on the opposite side of the field from the numbers
 - "Accounting0Black" \$36,000 / \$(45) "Black" – 0 decimal places; negative values are "Black" in parentheses
 - Accounting0Red" \$36,000 / \$(45) "Red" – 0 decimal places; negative values are "Red" in parentheses
 - "Accounting2Black" \$36,000.45 / \$(45.30) "Black" – 2 decimal places; negative values are "Black" in parentheses
 - "Accounting2Red" \$36,000.45 / \$(45.30) "Red" – 2 decimal places; negative values are "Red" in parentheses
- Date – Short
 - "DateShort1" 9/5/2016
 - "DateShort2" 09/05/2016
 - "DateShort3" 09/5
 - "DateShort4" 5-Sep
 - "DateShort5" 5-Sep-16
 - "DateShort6" 05-Sep-16
 - "DateShort7" Sep-5
 - "DateShort8" September-5
- Date – General
 - "DateGeneral1" 2016/9/5
 - "DateGeneral2" 2016/09/05
- Date – Long
 - "DateLong1" Sep 9, 2016
 - "DateLong2" Fri, Sep 9, 2016
 - "DateLong3" Friday, Sep 9, 2016
 - "DateLong4" Friday, September 9, 2016
- Time
 - "Time112" 9:30 PM – 12 hour clock
 - "Time124" 21:30 – 24 hour clock
 - "Time212" 09:30 PM - 12 hour clock
 - "Time224" 21:30 – 24 hour clock
 - "Time312" 09:30:45 PM – 12 hour clock
 - "Time324" 21:30:45 – 24 hour clock
- Timestamp
 - "TimeStamp112" 2016/09/05 02:42:15 PM – 12 hour timestamp
 - "TimeStamp124" 2016/09/05 14:42:15 – 24 hour timestamp
 - "TimeStamp212" 2016/09/05 02:42:15.544 PM – 12 hour timestamp with Milliseconds
 - "TimeStamp224" 2016/09/05 14:42:15.544 – 24 hour timestamp with Milliseconds

`public string` NumberFormat

Sample Code:

`// Select number format.`

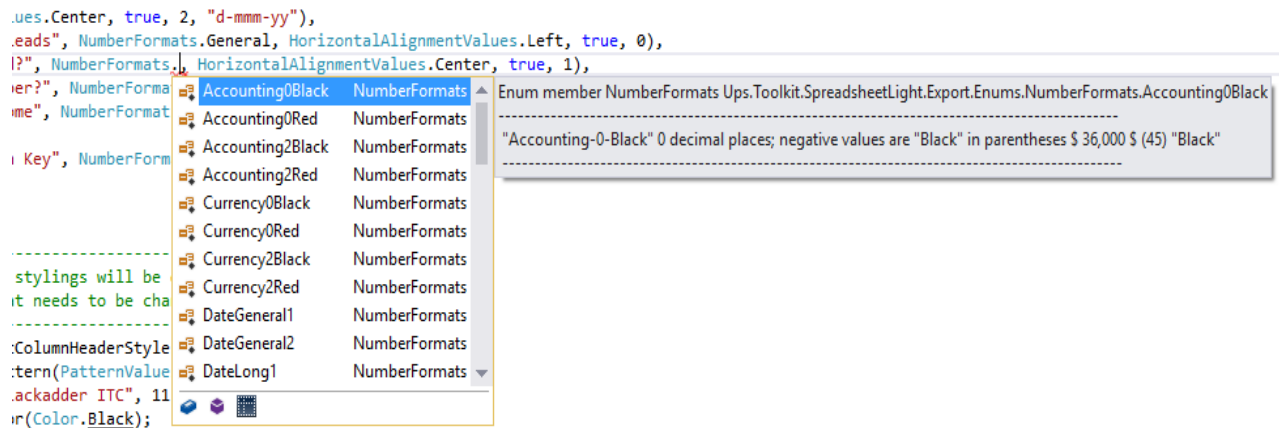
NumberFormat = `NumberFormats`.Accounting2Red,

- “HorizontalAlignment” (Optional) – Select a HorizontalAlignmentValues Enum value for a Data Alignment.
`public string HorizontalAlignment`

Sample Code:

```
// Select horizontal alignment.
HorizontalAlignment = HorizontalAlignmentValues.Right,
```

FOR THE NUMBERFORMAT ENUM AS WITH OTHER ENUMS TYPING A PERIOD AFTER THE ENUM NAME WILL BRING UP A DROPDOWN OF ITEMS TO SELECT



- “ShowField” (Optional) – Enter a boolean value of true or false to show or hide the field.
`public bool ShowField`

Sample Code:

```
// Set field visibility.
ShowField = true,
```

- “FieldOrder” (Optional) – Enter an integer value of 1, 2, 3, 4, etc. to set the field display order.
`public int? FieldOrder`

Sample Code:

```
// Set field order.
FieldOrder = 1,
```

- “UserDefinedNumberFormat” (Optional) – Enter a valid excel format.
`public string UserDefinedNumberFormat`

Sample Code:

```
// Set field order.
UserDefinedNumberFormat = "_($* #,##0.00_);[Red]_($* (#,##0.00);_($* \"-\"??_);_(@_)",
```

Examples

Configuring the Settings Class

```
var settings = new List<ChildSetting>()

var settings new Settings("Organization", settings);

var settings new Settings {
    Name = "Organization",
    ChildSettings = settings
};
```

Configuring the ChildSetting Class

Parameter Injection

This method is more flexible if you're going to populate a limited number of parameters

```
var childList = new List<ChildSetting>();
var baseColumnHeaderStyle = new SLStyle();
var oddRowStyle = new SLStyle();
var evenRowStyle = oddRowStyle.Clone();
var columns = new List<Column>();
/* -----
 * U
 * -----*/
childList.Add(new ChildSetting
{
    // Optional name
    Name = "Directors",
    // Set column visibility
    ShowColumnHeader = true,
    // Make the base column header row a little larger
    // so it will stand out. Value is in pixels
    ColumnHeaderRowHeight = 25,
    // Setup the style for Column Headers
    ColumnHeaderStyle = baseColumnHeaderStyle,
    // Row and Alternating Row Styles
    // If set to false then the odd row style will be overall row style
    ShowAlternatingRows = true,
    // Setup the style for all rows
    OddRowStyle = oddRowStyle,
    EvenRowStyle = evenRowStyle,
    // Add the user-defined columns
    UserDefinedColumns = columns
});
```

Constructor Injection

This method works well when you're going to use all the parameters.

Example 1

```
childList.Add(new ChildSetting(  
    name: "Managers",           // Name  
    showColumnHeader: true,     // Show Column Headers  
    columnOffset: 1,           // Column Offset to the Right  
    columnHeaderHeight: null,   // Column Header Row Height  
    columnHeaderStyle: firstColumnHeaderStyle, // Column Header Style  
    showAlternatingRows: false, // Show Alternating Rows, false will default to Odd  
    oddRowStyle: oddRowStyle,   // Odd Row Style  
    evenRowStyle: null,        // Even Row Style  
    userDefinedColumns: columns // User-Defined Column (UDCs)  
));
```

Example 2

```
childList.Add(new ChildSetting("Associates", true, 3, 30, thirdColumnHeaderStyle,  
    true, oddRowStyle, evenRowStyle, columns));
```

Configuring the Column Class

Parameter Injection

```
var columns = new List<Column>  
{  
    new Column  
    {  
        BoundColumnName = "DID",  
        UserDefinedColumnName = "ID",  
        NumberFormat = NumberFormats.General,  
        HorizontalAlignment = HorizontalAlignmentValues.Center,  
        ShowField = false,  
        FieldOrder = 0  
    },  
    new Column  
    {  
        BoundColumnName = "Age",  
        UserDefinedColumnName = "Chronology",  
        NumberFormat = NumberFormats.Decimal0,  
        HorizontalAlignment = HorizontalAlignmentValues.Center,  
        ShowField = true,  
        FieldOrder = 2  
    },  
    new Column  
    {  
        BoundColumnName = "Income",  
        UserDefinedColumnName = "Compensation",  
        NumberFormat = NumberFormats.UserDefined,  
        HorizontalAlignment = HorizontalAlignmentValues.Right,  
        ShowField = true,  
        FieldOrder = 3,  
        UserDefinedNumberFormat = "_( $* #,##0.00_);[Red]_( $* (#,##0.00);_( $* \"-  
        \"??_);_(@_)\"  
    }  
};
```

Constructor Injection

Example 1

```
columns = new List<Column>
{
    new Column
    (
        boundColumnName: "Name",
        userDefinedColumnName: "Managers",
        numberFormat: NumberFormats.General,
        horizontalAlignment: HorizontalAlignmentValues.Left,
        showField: true,
        fieldOrder: 1
    ),
    new Column
    (
        "Age",
        "Age",
        NumberFormats.UserDefined,
        HorizontalAlignmentValues.Center,
        true,
        2,
        "_(* #,##0_);_(* (#,##0);_(* \"-\"_);_(@_)"
    )
};
```

Example 2

```
columns = new List<Column> // User-Defined Column (UDCs)
{
    new Column("TLID", "Team Lead ID", NumberFormats.General, HorizontalAlignmentValues.Left,
        true, 6),
    new Column("Registered", "Registration Date", NumberFormats.UserDefined,
        HorizontalAlignmentValues.Center, true, 2, "d-mmm-yy"),
    new Column("Name", "Team Leads", NumberFormats.General,
        HorizontalAlignmentValues.Left, true, 0),
    new Column("Age", "How Old?", NumberFormats.Decimal0, HorizontalAlignmentValues.Center,
        true, 1),
    new Column("Member", "Member?", NumberFormats.General, HorizontalAlignmentValues.Center,
        true, 3),
    new Column("Income", "Income", NumberFormats.Accounting2Red,
        HorizontalAlignmentValues.Right, true, 4),
    new Column("MID", "Foreign Key", NumberFormats.General, HorizontalAlignmentValues.Right,
        false)
}..
```

Putting It All Together

In this example the various methods of construction and implementation are explored, use the ones that best suit your requirements

```
var childList = new List<ChildSetting>();
/* -----
 * Setup the column header base style for the child datasets
 * Documentation for SLStyle class in the SpreadsheetLight
 * documentation & examples.
 * For every Enum typing a period after the name will bring up
 * a drop-down of items to select.
 * -----*/
var baseColumnHeaderStyle = new SLStyle();
baseColumnHeaderStyle.SetHorizontalAlignment(HorizontalAlignmentValues.Center);
baseColumnHeaderStyle.SetVerticalAlignment(VerticalAlignmentValues.Center);
baseColumnHeaderStyle.Fill.SetPattern(PatternValues.Solid, Color.DimGray, Color.White);
baseColumnHeaderStyle.SetBottomBorder(BorderStyleValues.Medium, Color.Black);
baseColumnHeaderStyle.SetTopBorder(BorderStyleValues.Medium, Color.Black);
baseColumnHeaderStyle.SetVerticalBorder(BorderStyleValues.Medium, Color.Black);
baseColumnHeaderStyle.Border.SetRightBorder(BorderStyleValues.Medium, Color.Black);
baseColumnHeaderStyle.Border.SetLeftBorder(BorderStyleValues.Medium, Color.Black);
baseColumnHeaderStyle.SetFont("Britannic Bold", 12);
baseColumnHeaderStyle.SetFontColor(Color.White);
baseColumnHeaderStyle.SetFontBold(true);

/* -----
 * Setup the odd row style for the child datasets
 * -----*/
var oddRowStyle = new SLStyle();
oddRowStyle.SetHorizontalAlignment(HorizontalAlignmentValues.Left);
oddRowStyle.SetVerticalAlignment(VerticalAlignmentValues.Center);
oddRowStyle.Fill.SetPattern(PatternValues.Solid, Color.White, Color.Black);
oddRowStyle.SetFont("Helvetica", 10);
oddRowStyle.SetFontColor(Color.Black);

/* -----
 * Setup the even row style derived from the odd,
 * change only what is necessary.
 * -----*/
var evenRowStyle = oddRowStyle.Clone();
evenRowStyle.Fill.SetPattern(PatternValues.Solid, Color.WhiteSmoke, Color.Black);

/* -----
 * Create the user-defined columns with property dependency
 * injection for the base dataset.
 * With this method hover the cursor over the property and
 * intellisense will show the comments for it.
 * -----*/
var columns = new List<Column>
{
    // Since this id column is not set to visible, you can just leave it out and it will be
    // ignored
    new Column
    {
        BoundColumnName = "DID",
        UserDefinedColumnName = "ID",
        NumberFormat = NumberFormats.General,
        HorizontalAlignment = HorizontalAlignmentValues.Center,
        ShowField = false,
        FieldOrder = 0
    },
    new Column
```

```

{
    BoundColumnName = "Name",
    UserDefinedColumnName = "Director",
    NumberFormat = NumberFormats.General,
    HorizontalAlignment = HorizontalAlignmentValues.Left,
    ShowField = true,
    FieldOrder = 1
},
new Column
{
    BoundColumnName = "Age",
    UserDefinedColumnName = "Chronology",
    NumberFormat = NumberFormats.Decimal0,
    HorizontalAlignment = HorizontalAlignmentValues.Center,
    ShowField = true,
    FieldOrder = 2
},
new Column
{
    BoundColumnName = "Income",
    UserDefinedColumnName = "Compensation",
    NumberFormat = NumberFormats.UserDefined,
    HorizontalAlignment = HorizontalAlignmentValues.Right,
    ShowField = true,
    FieldOrder = 3,
    UserDefinedNumberFormat = "_( $* #,##0.00_);[Red]_( $* (#,##0.00);_($* \"-
    \"?_?_);_(@_)\"
},
new Column
{
    BoundColumnName = "Member",
    UserDefinedColumnName = "Member ?",
    NumberFormat = NumberFormats.General,
    HorizontalAlignment = HorizontalAlignmentValues.Center,
    ShowField = true,
    FieldOrder = 4
},
new Column
{
    BoundColumnName = "Registered",
    UserDefinedColumnName = "Date Registered",
    NumberFormat = NumberFormats.DateShort5,
    HorizontalAlignment = HorizontalAlignmentValues.Center,
    ShowField = true,
    FieldOrder = 5
}
};

/* -----
 * Define and style base child settings.
 * This Child will always be present, it represents the
 * primary dataset for every export and is not really a child.
 * Using Property Injection Technique
 * -----*/
childList.Add(new ChildSetting
{
    // Optional name
    Name = "Directors",
    // Set column visibility
    ShowColumnHeader = true,
    // Make the base column header row a little larger
    // so it will stand out. Value is in pixels
    ColumnHeaderRowHeight = 25,

```

```

        // Setup the style for Column Headers
        ColumnHeaderStyle = baseColumnHeaderStyle,
        // Row and Alternating Row Styles
        // If set to false then the odd row style will be overall row style
        ShowAlternatingRows = false,
        // Setup the style for all rows
        OddRowStyle = oddRowStyle,
        EvenRowStyle = null,
        // Add the user-defined columns
        UserDefinedColumns = columns
    });

    /* -----
    * The first child column headers stylings will be derived
    * from the base, change only what needs to be changed.
    * -----*/
    var firstColumnHeaderStyle = baseColumnHeaderStyle.Clone();
    firstColumnHeaderStyle.Fill.SetPattern(PatternValues.Solid, Color.DarkGray, Color.Black);
    firstColumnHeaderStyle.SetBottomBorder(BorderStyleValues.Thin, Color.DarkSlateGray);
    firstColumnHeaderStyle.SetTopBorder(BorderStyleValues.Thin, Color.DarkSlateGray);
    firstColumnHeaderStyle.SetVerticalBorder(BorderStyleValues.Thin, Color.DarkSlateGray);
    firstColumnHeaderStyle.Border.SetRightBorder(BorderStyleValues.Thin, Color.DarkSlateGray);
    firstColumnHeaderStyle.Border.SetLeftBorder(BorderStyleValues.Thin, Color.DarkSlateGray);
    firstColumnHeaderStyle.SetFont("Helvetica", 10);
    firstColumnHeaderStyle.SetFontColor(Color.Black);

    /* -----
    * Create the user-defined columns with constructor dependency
    * injection for the base dataset.
    * Hover the cursor over the property and intellisense will
    * show the comments for it.
    * -----*/
    columns = new List<Column>
    {
        new Column
        (
            boundColumnName: "Name",
            userDefinedColumnName: "Managers",
            numberFormat: NumberFormats.General,
            horizontalAlignment: HorizontalAlignmentValues.Left,
            showField: true,
            fieldOrder: 1
        ),
        new Column
        (
            boundColumnName: "Age",
            userDefinedColumnName: "Age",
            numberFormat: NumberFormats.UserDefined,
            horizontalAlignment: HorizontalAlignmentValues.Center,
            showField: true,
            fieldOrder: 2,
            userDefinedNumberFormat: "_( * #,##0_);_( * ( #,##0);_( * \"-\"_);_( @_)"
        ),
        new Column
        (
            boundColumnName: "Income",
            userDefinedColumnName: "Compensation",
            numberFormat: NumberFormats.Currency0Black,
            horizontalAlignment: HorizontalAlignmentValues.Right,
            showField: true,
            fieldOrder: 3
        ),
        new Column

```



```

        boundColumnName: "Registered",
        userDefinedColumnName: "Date Registered",
        numberFormat: NumberFormats.DateShort1,
        horizontalAlignment: HorizontalAlignmentValues.Center,
        showField: true,
        fieldOrder: 5
    )
};
/* -----
 * Define and add the first child
 * Using Constructor dependency injection
 * -----*/
childList.Add(new ChildSetting
(
    name: "Managers",                // Name
    showColumnHeader: true,           // Show Column Headers
    columnOffset: 1,                  // Column Offset to the Right
    columnHeaderRowHeight: null,      // Column Header Row Height
    columnHeaderStyle: firstColumnHeaderStyle, // Column Header Style
    showAlternatingRows: false,       // Show Alternating Rows, false will default to Odd
    oddRowStyle: oddRowStyle,         // Odd Row Style
    evenRowStyle: null,               // Even Row Style
    userDefinedColumns: columns       // User-Defined Column (UDCs)
));

/* -----
 * The second child column headers stylings will be derived
 * from the first, change only what needs to be changed.
 * -----*/
var secondColumnHeaderStyle = firstColumnHeaderStyle.Clone();
secondColumnHeaderStyle.Fill.SetPattern(PatternValues.Solid, Color.CadetBlue, Color.White);
secondColumnHeaderStyle.SetFontColor(Color.White);

/* -----
 * Define and add the second child
 * Using Constructor dependency injection
 * -----*/
childList.Add(new ChildSetting(
    "Team Leads",                    // Name
    true,                             // Show Column Headers
    2,                                // Column Offset to the Right
    null,                             // Column Header Row Height
    secondColumnHeaderStyle,          // Column Header Style
    false,                            // Show Alternating Rows, false will default to Odd
    oddRowStyle,                      // Odd Row Style
    null,                             // Even Row Style
    new List<Column>                  // User-Defined Column (UDCs)
    {
        new Column("TLID", "Team Lead ID", NumberFormats.General,
            HorizontalAlignmentValues.Left, true,
            6),
        new Column("Registered", "Registration Date", NumberFormats.UserDefined,
            HorizontalAlignmentValues.Center, true, 2, "d-mmm-yy"),
        new Column("Name", "Team Leads", NumberFormats.General,
            HorizontalAlignmentValues.Left, true, 0),
        new Column("Age", "How Old?", NumberFormats.Decimal0,
            HorizontalAlignmentValues.Center, true, 1),
        new Column("Member", "Member?", NumberFormats.General,
            HorizontalAlignmentValues.Center, true, 3),
        new Column("Income", "Income", NumberFormats.Accounting2Red,
            HorizontalAlignmentValues.Right,
            true, 4),
    }

```

```

        new Column("MID", "Foreign Key", NumberFormats.General,
HorizontalAlignmentValues.Right, false)
    }
));

/* -----
 * The third child column headers stylings will be derived
 * from the first, change only what needs to be changed.
 * -----*/
var thirdColumnHeaderStyle = firstColumnHeaderStyle.Clone();
thirdColumnHeaderStyle.Fill.SetPattern(PatternValues.Solid, Color.Aqua, Color.Black);
thirdColumnHeaderStyle.SetFont("Blackadder ITC", 11);
thirdColumnHeaderStyle.SetFontColor(Color.Black);

/* -----
 * Define and add the third child
 * Constructor Injection on all
 * -----*/
childList.Add(new ChildSetting("Associates", true, 3, 30, thirdColumnHeaderStyle, true,
oddRowStyle, evenRowStyle,
new List<Column>
{
    new Column("Registered", "Date", NumberFormats.TimeStamp124,
HorizontalAlignmentValues.Left, true, 3),
    new Column("Member", "Member?", NumberFormats.General,
HorizontalAlignmentValues.Center, true, 2),
    new Column("Name", "Associate", NumberFormats.General,
HorizontalAlignmentValues.Left, true, 0)
}
));

/* -----
 * Setup and return the primary container for the child datasets
 * Using Constructor Injection.
 * -----*/
return new Settings("Organization", childList);

```