ComponentOne

# Excel for Silverlight

**ComponentOne, a division of GrapeCity**

201 South Highland Avenue, Third Floor

Pittsburgh, PA 15206 • USA

**Internet:** info@ComponentOne.com

**Web site:** http://www.componentone.com

**Sales**

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)



This manual was produced using ComponentOne Doc-To-Help™.

# Table of Contents

# ComponentOne Excel for Silverlight Overview

Your Excel® data is just a simple command away from any of your .NET applications with **ComponentOne Excel for Silverlight** – you don't even need to have Microsoft® Excel installed! Create or load XLS files for Excel 97 and later. **Excel for Silverlight** supports the new Office 2007 OpenXml format, which allows you to save smaller, compressed XLSX files.

The main component in **Excel for Silverlight** is the C1XLBook object, which represents an Excel workbook containing one or more sheets. Use the C1XLBook to load existing Excel files or create new ones. Then add sheets, styles, hyperlinks, images, headers and footers, page breaks and more. When you are done, save the C1XLBook to a file or a Stream and you're done. Anyone with a copy of Excel can access your data. It's that easy!

## Help with ComponentOne Studio for Silverlight

**Getting Started**

For information on installing ComponentOne Studio for Silverlight, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for Silverlight.

**What's New**

For a list of the latest features added to **ComponentOne Studio for Silverlight**, visit What's New in Studio for Silverlight.

## Key Features

The following are some of the main features of **ComponentOne Excel for Silverlight** that you may find useful:

- Save or load a workbook with one command

  **Excel for Silverlight** is easy-to-use, allowing you to use a single command to load or save a workbook and manipulate sheets as if they were grid controls.

- Read and write data in individual cells

  After loading or creating a C1XLBook, you can access data in individual sheets as if they were a simple grid. For example:

```
XLSheet sheet = C1XLBook.Sheets[0];
sheet[0, 0].Value = DateTime.Now;
```

- Format the data in each cell.

  The format associated with each cell is as easy to access as the data stored in the cell. For example:

```
XLStyle style = new XLStyle(c1XLBook1);
style.Format = "dd-MM-yyyy";
style.Font = new Font("Courier New", 14);
XLSheet sheet = C1XLBook.Sheets[0];
sheet[0, 0].Value = DateTime.Now;
sheet[0, 0].Style = style;
```

- Use **Excel for Silverlight** to export to XLSX files

Other ComponentOne components use **Excel for Silverlight** to export XLS files. For example, **C1Report** uses **Excel for Silverlight** to create XLSX versions of reports so they can be viewed and edited by anyone with a copy of Microsoft Excel.

- Reads and writes.xlsx files without using Microsoft Excel

  **Excel for Silverlight** reads and writes .xlsx (OpenXml format) files, the which can be reused and easily exchanged or compressed to create smaller file sizes. You don't even need to have Microsoft Excel installed.

- Save and load files to and from streams

  Workbooks can now be directly read to and written from memory streams with new overloads for the Load and the Save methods so you no longer have to use temporary files.

# Excel for Silverlight QuickStart

This quick start guide will familiarize you with some of the features of **Excel for Silverlight**. In this quick start you will learn how to add a **C1XLBook** to the project, add formatted data to the workbook, and save and open the XLS file.

## Step 1 of 4: Setting up the Project

In this step you will create a new project and add a reference to the C1.Silverlight.Excel assembly.

1. Create a new Silverlight project.
2. Select **Project** | **Add Reference** and browse to find C1.Silverlight.Excel.dll. Click **OK**.
3. Add a standard button control to the page and set its **Content** property to **Save**.
4. Double-click the button to switch to the code view of MainPage.xaml.cs. This will also add a **button1_Click** event to the code.
5. Add the **Imports** (Visual Basic) or **using** (C#) statement to the code at the top of the form so you can use all names within the C1.Silverlight.Excel namespace.

   - Visual Basic

     ```
     Imports C1.Silverlight.Excel
     ```

   - C#

     ```
     using C1.Silverlight.Excel;
     ```

Now you can add some content to a **C1XLBook**.

## Step 2 of 4: Adding Content to a C1XLBook

In the **button1_Click** event we created in step 1, add the following code to create the book and its content so it looks like the following:

- Visual Basic

  ```
  Private Sub button1_Click(sender As Object, e As RoutedEventArgs)


     SaveBook(Function(book)
  ```

```
        Dim i As Integer
        Dim sheet As C1.Silverlight.Excel.XLSheet = book.Sheets(0)
        For i = 0 To 9
              sheet(i, 0).Value = (i + 1) * 10
              sheet(i, 1).Value = (i + 1) * 100
              sheet(i, 2).Value = (i + 1) * 1000
        Next

    End Function)
    End Sub
```

- C#

```
    private void button1_Click(object sender, RoutedEventArgs e)
            {

                SaveBook(book =>
              {



                 int i;
                 C1.Silverlight.Excel.XLSheet sheet = book.Sheets[0];
                 for (i = 0; i <= 9; i++)
                 {
                     sheet[i, 0].Value = (i + 1) * 10;
                     sheet[i, 1].Value = (i + 1) * 100;
                     sheet[i, 2].Value = (i + 1) * 1000;
                 }
                });
            }
```

You'll notice a call to the **SaveBook** method. We'll add the code for this method in the next step.

# Step 3 of 4: Saving the XLSX File

Add the following code to save the Excel workbook. When you run the application, the code will open the **Save As** dialog box so you can save your .xlsx file wherever you'd like.

- Visual Basic

```
Private Sub SaveBook(action As Action(Of C1XLBook))
Dim dlg = New SaveFileDialog()
dlg.Filter = "Excel Files (*.xlsx)|*.xlsx"
If dlg.ShowDialog() = True Then
      Try
            Dim book = New C1XLBook()
```

```
            RaiseEvent action(book)
            Using stream = dlg.OpenFile()
                  book.Save(stream)
            End Using
      Catch x As Exception
            MessageBox.Show(x.Message)
      End Try
End If
End Sub
```

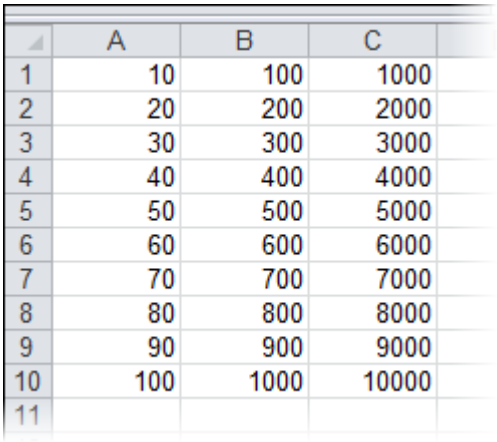
- C#

```
private void SaveBook(Action<C1XLBook> action)
        {
            var dlg = new SaveFileDialog();
            dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
            if (dlg.ShowDialog() == true)
            {
                try
                {
                    var book = new C1XLBook();
                    if (action != null)
                    {
                        action(book);
                    }
                    using (var stream = dlg.OpenFile())
                    {
                        book.Save(stream);
                    }
                }
                catch (Exception x)
                {
                    MessageBox.Show(x.Message);
                }
            }
        }
```

## Step 4 of 4: Run the Program

Press F5 to run the application.

1. Click the **Save** button. The **Save As** dialog box appears.
2. Enter a file name for your workbook and click **Save**.
3. Open the book. It will look similar to the following image.



| | A | B | C |
|---|---|---|---|
| 1 | 10 | 100 | 1000 |
| 2 | 20 | 200 | 2000 |
| 3 | 30 | 300 | 3000 |
| 4 | 40 | 400 | 4000 |
| 5 | 50 | 500 | 5000 |
| 6 | 60 | 600 | 6000 |
| 7 | 70 | 700 | 7000 |
| 8 | 80 | 800 | 8000 |
| 9 | 90 | 900 | 9000 |
| 10 | 100 | 1000 | 10000 |
| 11 | | | |

Congratulations! You've completed the **Excel for Silverlight** quick start.

# Using Excel for Silverlight

The following topics explain how to create an XLSX file, as well as describe the main **Excel for Silverlight** classes used to create the components that make up the file, which include worksheets, rows and columns, cells and styles.

## Creating Documents

To create a new XLSX file using **Excel for Silverlight**, three steps are required:

1. Add a reference to C1.Silverlight.Excel.dll and create a **C1XLBook**.
2. Add content to the sheets. Each sheet contains cells (XLCell objects) that have a Value and a Style property.
3. Save the book to a file using the Save method.

For example, the following code creates a new Excel file with a single sheet containing numbers from 1 to 100.

- C#

```
private void button1_Click_1(object sender, RoutedEventArgs e)
        {
            // step 1: create a new workbook to be saved
            SaveBook(book =>
{
    // step 2: write content into some cells
    XLSheet sheet = book.Sheets[0];
    for (int i = 0; i < 100; i++)
        sheet[i, 0].Value = i + 1;
});
        }


// step 3: save the file
        private void SaveBook(Action<C1XLBook> action)
        {
            var dlg = new SaveFileDialog();
            dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
            if (dlg.ShowDialog() == true)
```

```
            {
                try
                {
                    var book = new C1XLBook();
                    if (action != null)
                    {
                        action(book);
                    }
                    using (var stream = dlg.OpenFile())
                    {
                        book.Save(stream);
                    }
                }
                catch (Exception x)
                {
                    MessageBox.Show(x.Message);
                }
            }
        }
```

Step 2 is the most interesting one. The code starts by retrieving an XLSheet object that represents the single worksheet in the new Excel workbook. This sheet is created automatically when you add or create a new C1XLBook. Then the code uses the sheet indexer to reference cells in the sheet and assign them values from 1 to 100.

Note that the indexer in the **XLSheet** object automatically creates cells, if necessary. This makes it easy to fill worksheets that you create. If you want to find out the sheet dimensions, use the sheet's **Rows.Count** and **Columns.Count** properties.

Of course, you are not limited to assigning values to cells. You can also use styles to format the cells. Just create one or more XLStyle objects and assign them to cells much like you did values. This revised version of the code above creates a sheet where even numbers are shown in bold red characters with yellow highlighting and odd numbers are shown in italic blue.

- C#

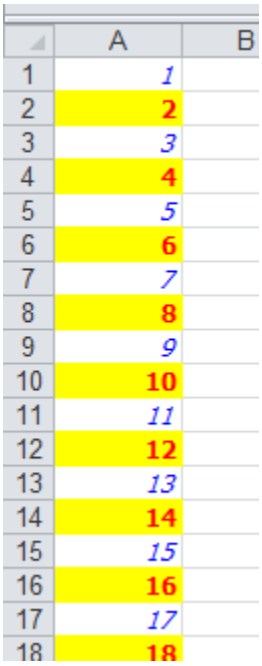
```
private void button1_Click_1(object sender, RoutedEventArgs e)
        {


            // step 1: create a new workbook to be saved
            SaveBook(book =>
{



    var sheet = book.Sheets[0];
    // step 2: create styles for odd and even values
    var styleOdd = new XLStyle(book);
    styleOdd.Font = new XLFont("Tahoma", 9, false, true);
    styleOdd.ForeColor = Colors.Blue;

    var styleEven = new XLStyle(book);
    styleEven.Font = new XLFont("Tahoma", 9, true, false);
```

```
    styleEven.ForeColor = Colors.Red;
    styleEven.BackColor = Colors.Yellow;

    // step 3: write content into some cells
    for (int i = 0; i < 100; i++)
    {
        XLCell cell = sheet[i, 0];
        cell.Value = i + 1;
        cell.Style = ((i + 1) % 2 == 0) ? styleEven : styleOdd;
    }

});



        }


        // step 4: save the file
        private void SaveBook(Action<C1XLBook> action)
        {
            var dlg = new SaveFileDialog();
            dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
            if (dlg.ShowDialog() == true)
            {
                try
                {
                    var book = new C1XLBook();
                    if (action != null)
                    {
                        action(book);
                    }
                    using (var stream = dlg.OpenFile())
                    {
                        book.Save(stream);
                    }
                }
                catch (Exception x)
                {
```

```
                MessageBox.Show(x.Message);
            }
        }
    }
```

The code is similar. The main difference is the new step 2, which creates styles for odd and even cells. The new styles are assigned to cells in step 3, along with the cell values.

This is what the file created by the code above looks like when opened in Microsoft Excel:



| | A | B |
|---|---|---|
| 1 | 1 | |
| 2 | 2 | |
| 3 | 3 | |
| 4 | 4 | |
| 5 | 5 | |
| 6 | 6 | |
| 7 | 7 | |
| 8 | 8 | |
| 9 | 9 | |
| 10 | 10 | |
| 11 | 11 | |
| 12 | 12 | |
| 13 | 13 | |
| 14 | 14 | |
| 15 | 15 | |
| 16 | 16 | |
| 17 | 17 | |
| 18 | 18 | |

# Worksheets

Worksheets are the individual grids contained in an Excel file. They are represented by XLSheet objects accessible through the **Sheets** property in the C1XLBook class. Each sheet has a name and contains a collection of rows and columns. Individual cells can be accessed using the **XLSheet** indexer, which takes row and column indices.

The Rows and Columns collections in the **XLSheet** object extend automatically when you use their indexers. For example, if you write the following code and the sheet has fewer than 1001 rows, new rows will be automatically added, and a valid row will be returned. The same applies to XLColumn and XLCell indexers. This is different from the behavior of most collection indexers in .NET, but it makes it very easy to create and populate **XLSheet** objects.

- Visual Basic
```
Dim sheet As XLSheet = book.Sheets(0)
Dim row As XLRow = sheet.Rows(1000)
```
- C#
```
XLSheet sheet = book.Sheets[0];
XLRow row = sheet.Rows[1000];
```

## Rows and Columns

The XLSheet object contains collections of rows and columns that expose each individual row and column on the sheet. The exposed XLRow and XLColumn objects allow you to assign the size (column width, row height), visibility, and style for each row and column on the sheet. If you don't assign any of these values, the sheet's defaults will be used (see the DefaultRowHeight and DefaultColumnWidth properties).

The default dimensions for **XLRow** and **XLColumn** objects are –1, which means use the sheet's default values.

## Cells

The XLSheet object also contains cells that can be accessed using an indexer that takes row and column indices. The cells are represented by XLCell objects that contain the cell value and style.

As with rows and columns, the cell indexer also extends the sheet automatically. For example, write:

- Visual Basic
```
Dim cell As XLCell = sheet(10, 10)
```
- C#
```
XLCell cell = sheet[10,10];
```

If the sheet has fewer than 11 rows and 11 columns, rows and columns will be added and a valid **XLCell** object will be returned.

Because the sheet expands automatically, this indexer will never return a **null** reference. If you want to check whether a particular cell exists on the sheet and you don't want to create the cell inadvertently, use the sheet's GetCell method instead of the indexer.

**XLCell** objects have a Value property that contains the cell contents. This property is of type **object** and it may contain strings, numeric, Boolean, DateTime, or null objects. Other types of objects cannot be saved into Excel files.

**XLCell** objects also have a Style property that defines the appearance of the cell. If the **Style** property is set to **null**, the cell is displayed using the default style. Otherwise, it should be set to an **XLStyle** object that defines the appearance of the cell (font, alignment, colors, format, and so on).

## Styles

The XLStyle class defines the appearance of a cell, row, or column on a sheet. **XLStyle** includes properties that specify style elements such as the font, alignment, colors, and format used to display cell values. Not all style elements need to be defined in every **XLStyle** object. For example, if an **XLStyle** specifies only a format, then the cell is displayed using the specified format and default settings for the other style elements (font, alignment, and so on).

# Excel for Silverlight Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos which may make use of other development tools included with the ComponentOne Studios.

Samples can be accessed from the **ComponentOne Sample Explorer**. To view samples, on the desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for Silverlight | Samples | Excel Samples**.

| Sample | Description |
|---|---|
| AutoSizeColumns | The sample loops through all cells measuring their contents, then sets the column widths based on the widest entry. This sample takes into account the cell |

| | |
|---|---|
| | contents, format and font. It does not account for content wrapping or merging. This sample uses the **C1XLBook** component. |
| CombineSheets | This sample scans all the .xlsx files in a folder, clones the first sheet, renames it with the file name, and adds the cloned sheets to a master workbook. It then saves the combined workbook into a file and opens that. This sample uses the **C1XLBook** component. |
| ExcelDemos | This sample shows three different example spreadsheets: one creates a simple Hello World spreadsheet, the second creates a spreadsheet with formulas, and the third creates a spreadsheet using styles, such as font, borders, background colors, and so on. |
| ExcelFormulas | This sample demonstrates how to use formulas in your spreadsheets. |
| HelloWorld | This sample actually does more than save a "Hello World" document. It shows how to create workbooks, assign names to the worksheets, create styles and assign them to cells, assign content to cells, and save workbooks.. This sample uses the **C1XLBook** component. |
| PrintSettings | This sample has a Load button that opens an existing XLS file and shows the print settings for the first sheet. You can modify the print settings, then click the Save button to save a new file (in the c:\temp folder). The new file is then displayed in Excel . This sample uses the **C1XLBook** component. |

# Excel for Silverlight Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET. By following the steps outlined in the Help, you will be able to create projects demonstrating a variety of **Excel for Silverlight** features and get a good sense of what **Excel for Silverlight** can do.

Each task-based help topic also assumes that you have created a new Silverlight project and added the appropriate directives (**using C1.C1Excel;** for **C#**; **Imports C1.C1Excel** for **Visual Basic**) to the code.

## Adding Content to a Workbook

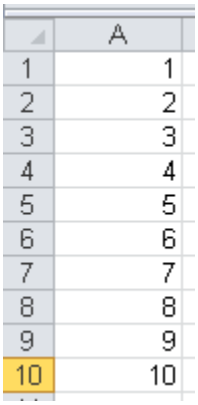
To create a new workbook and add values to the first ten cells, complete the following steps:

1. Add a reference to C1.Silverlight.Excel.dll and create a **C1XLBook**.
2. Add values to the first ten cells.
3. Save the workbook. The code looks like the following. In this example it is placed within a **button1_Click** event so the **Save As** dialog box will open when the user clicks the button.

- C#

```
private void button1_Click_1(object sender, RoutedEventArgs e)
        {
            // create a new workbook to be saved
            SaveBook(book =>
            {
                // write content for the first ten cells
                XLSheet sheet = book.Sheets[0];

                for (int i = 0; i <= 9; i++)
                {
```

```
                sheet[i, 0].Value = i + 1;
            }

        });
    }


    // save the file
    private void SaveBook(Action<C1XLBook> action)
    {
        var dlg = new SaveFileDialog();
        dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
        if (dlg.ShowDialog() == true)
        {
            try
            {
                var book = new C1XLBook();
                if (action != null)
                {
                    action(book);
                }
                using (var stream = dlg.OpenFile())
                {
                    book.Save(stream);
                }
            }
            catch (Exception x)
            {
                MessageBox.Show(x.Message);
            }
        }
    }
```



| | A |
|---|---|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | 10 |

## Formatting Cells

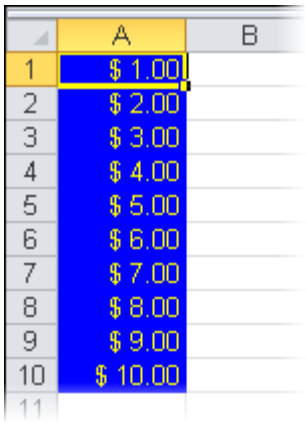
To format the cells of a book, complete the following steps:

1. Add a reference to C1.Silverlight.Excel.dll and create a **C1XLBook**.
2. Add some content to the workbook, create a new style and apply the styles to the cells in the first column of the sheet.
3. Save the workbook. The code looks like the following. In this example it is placed within a **button1_Click** event so the **Save As** dialog box will open when the user clicks the button.

- C#

```
private void button1_Click_1(object sender, RoutedEventArgs e)
        {
            // create a new workbook to be saved
            SaveBook(book =>
            {
                // Create a new style
                XLStyle style1 = new XLStyle(book);
                style1.ForeColor = Colors.Yellow;
                style1.BackColor = Colors.Blue;
                style1.Format = "$ .00";

                // Add content and apply styles to cells in first
column of sheet
                XLSheet sheet = book.Sheets[0];
                int i;

                for (i = 0; i <= 9; i++)
                {
                    sheet[i, 0].Value = i + 1;
                    sheet[i, 0].Style = style1;
                }
```

```
    });
}

// save the file
private void SaveBook(Action<C1XLBook> action)
{
    var dlg = new SaveFileDialog();
    dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
    if (dlg.ShowDialog() == true)
    {
        try
        {
            var book = new C1XLBook();
            if (action != null)
            {
                action(book);
            }
            using (var stream = dlg.OpenFile())
            {
                book.Save(stream);
            }
        }
        catch (Exception x)
        {
            MessageBox.Show(x.Message);
        }
    }
}
```




## Adding a Page Break to a Worksheet

You can easily add page breaks in rows and columns for files in OpenXML (.xlsx) format using the PageBreak and PageBreak properties.

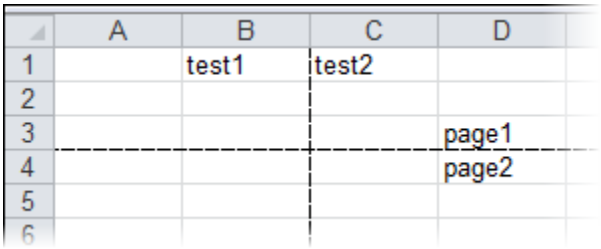
1. Add a reference to C1.Silverlight.Excel.dll and create a **C1XLBook**.
2. Add some text values and page breaks.
3. Save the workbook. The code looks like the following. In this example it is placed within a **button1_Click** event so the **Save As** dialog box will open when the user clicks the button.

- C#

```
private void button1_Click_1(object sender, RoutedEventArgs e)
        {
            // create a new workbook to be saved
            SaveBook(book =>
            {
                // add text values and page breaks
                book.Sheets[0][2, 3].Value = "page1";
                book.Sheets[0].Rows[2].PageBreak = true;

                book.Sheets[0][0, 1].Value = "test1";
                book.Sheets[0][0, 2].Value = "test2";

                book.Sheets[0].Columns[1].PageBreak = true;
                book.Sheets[0][3, 3].Value = "page2";
            });
        }

        // save the file
        private void SaveBook(Action<C1XLBook> action)
        {
```

```
            var dlg = new SaveFileDialog();
            dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
            if (dlg.ShowDialog() == true)
            {
                try
                {
                    var book = new C1XLBook();
                    if (action != null)
                    {
                        action(book);
                    }
                    using (var stream = dlg.OpenFile())
                    {
                        book.Save(stream);
                    }
                }
                catch (Exception x)
                {
                    MessageBox.Show(x.Message);
                }
            }
        }
```

4. Run the project to save the .xlsx file and then open the file.
5. In Excel, select the **Page Layout** tab, and select the **Print** checkbox under **Gridlines**. The worksheet should look similar to the following:




# Setting the Calculation Mode for a Workbook

The CalculationMode property specifies the calculation mode for all formulas in the workbook. The CalculationMode enumeration provides three options: **Manual** (you manually perform the calculation), **Auto** (the calculation is automatically performed), or **AutoNoTable** (the calculation is performed except on tables).

To set the calculation mode, follow these steps:

1. Add a reference to C1.Silverlight.Excel.dll and create a **C1XLBook**.

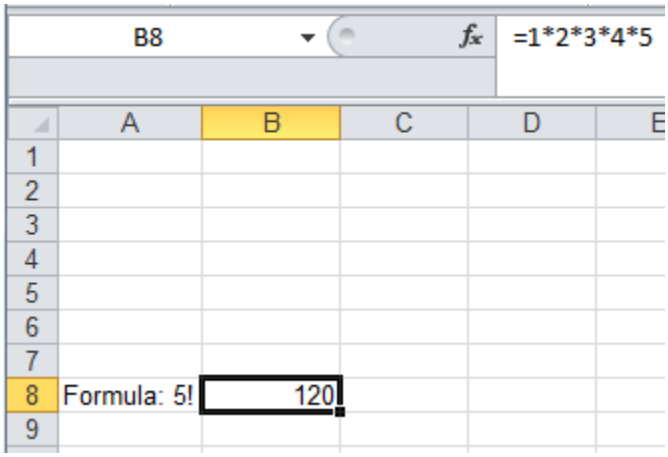
2. Add a simple formula to perform a calculation.
3. Save the workbook. The code looks like the following. In this example it is placed within a **button1_Click** event so the **Save As** dialog box will open when the user clicks the button.

- C#

```
private void button1_Click_1(object sender, RoutedEventArgs e)
        {
            // create a new workbook to be saved
            SaveBook(book =>
            {
                XLSheet sheet = book.Sheets[0];

                // simple formula
                sheet[7, 0].Value = "Formula: 5!";
                sheet[7, 1].Value = 122;
                sheet[7, 1].Formula = "1*2*3*4*5";
                book.CalculationMode = CalculationMode.Auto;
            });
        }

        // save the file
        private void SaveBook(Action<C1XLBook> action)
        {
            var dlg = new SaveFileDialog();
            dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
            if (dlg.ShowDialog() == true)
            {
                try
                {
                    var book = new C1XLBook();
                    if (action != null)
                    {
                        action(book);
                    }
                    using (var stream = dlg.OpenFile())
                    {
                        book.Save(stream);
                    }
                }
```

```
                catch (Exception x)
                {
                    MessageBox.Show(x.Message);
                }
            }
        }
```

4. Run the project and save and open the Excel file. Notice that the value for the cell in (7,1) is **120**, or the total of **1*2*3*4*5**, not **122**, since we set the CalculationMode to **Auto**.




## Creating Subtotals

The following code provides an example of how to format the cells of a book.

1. Add a reference to C1.Silverlight.Excel.dll and create a **C1XLBook**.
2. Add code to format the cells.
3. Save the workbook. The code looks like the following. In this example it is placed within a **button1_Click** event so the **Save As** dialog box will open when the user clicks the button

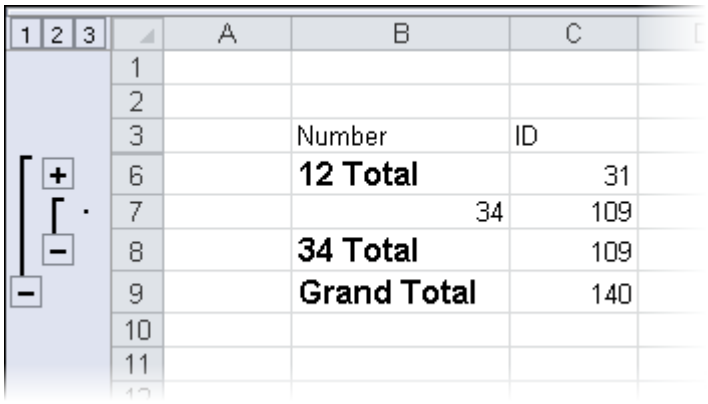
- C#

```
private void button1_Click_1(object sender, RoutedEventArgs e)
        {
            // create a new workbook to be saved
            SaveBook(book =>
            {
                XLSheet sheet = book.Sheets[0];
                // create a style
                XLStyle totalStyle = new XLStyle(book);
                 totalStyle.Font = new XLFont("Arial", 12, true,
false);
               // create an outline and apply styles
```

```
            sheet[2, 1].Value = "Number";
            sheet[2, 2].Value = "ID";
            sheet[3, 1].Value = 12;
          sheet[3, 2].Value = 17;
            sheet.Rows[3].OutlineLevel = 2;
            sheet.Rows[3].Visible = false;
            sheet[4, 1].Value = 12;
            sheet[4, 2].Value = 14;
            sheet.Rows[4].OutlineLevel = 2;
            sheet.Rows[4].Visible = false;
            sheet[5, 1].Value = "12 Total";
            sheet[5, 1].Style = totalStyle;
            sheet[5, 2].Value = 31;
            sheet[5, 2].Formula = "SUBTOTAL(9,C4:C5)";
            sheet.Rows[5].OutlineLevel = 1;
            sheet[6, 1].Value = 34;
            sheet[6, 2].Value = 109;
            sheet.Rows[6].OutlineLevel = 2;
            sheet[7, 1].Value = "34 Total";
            sheet[7, 1].Style = totalStyle;
            sheet[7, 2].Value = 109;
            sheet[7, 2].Formula = "SUBTOTAL(9,C7:C7)";
            sheet.Rows[7].OutlineLevel = 1;
            sheet[8, 1].Value = "Grand Total";
            sheet[8, 1].Style = totalStyle;
            sheet[8, 2].Value = 140;
            sheet[8, 2].Formula = "SUBTOTAL(9,C4:C7)";
            sheet.Rows[8].OutlineLevel = 0;
        });
    }
    // save the file
    private void SaveBook(Action<C1XLBook> action)
    {
        var dlg = new SaveFileDialog();
        dlg.Filter = "Excel Files (*.xlsx)|*.xlsx";
        if (dlg.ShowDialog() == true)
        {
            try
```

```
            {
                var book = new C1XLBook();
                if (action != null)
                {
                    action(book);
                }
                using (var stream = dlg.OpenFile())
                {
                    book.Save(stream);
                }
            }
            catch (Exception x)
            {
                MessageBox.Show(x.Message);
            }
        }
    }
```

4. Run the program. Save and open the file. The spreadsheet will look similar to the following:



| | A | B | C |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | Number | ID |
| 6 | | **12 Total** | 31 |
| 7 | | 34 | 109 |
| 8 | | **34 Total** | 109 |
| 9 | | **Grand Total** | 140 |
| 10 | | | |
| 11 | | | |

The SUBTOTAL formulas get the sum of the specified rows.

# Saving a CSV File

**ComponentOne Excel for Silverlight** supports saving and loading comma-separated values (CSV) files. CSV is a common file format that stores tabular data, including numbers and text, in plain text form for easy readability.

The following code provides an example of how to save a .csv file.

1. Add a reference to C1.Silverlight.Excel.dll.
2. Add a standard **TextBox** control to the page and name it **txt_Status**.
3. Add a **Button** control to the page and name it **_btnSave**.
4. Select **View | Code** and add one of the following statements at the top of the form:

- Import C1.Silverlight.Excel (Visual Basic)
- using C1.Silverlight.Excel; (C#)

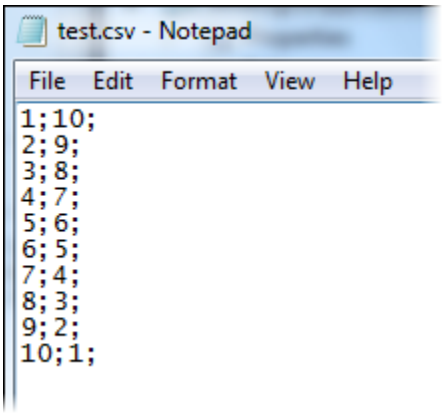
5. Create a new C1XLBook named _book and add a sheet with some values to the book. Use the following code:

```
public partial class MainPage : UserControl
    {
        C1XLBook _book = new C1XLBook();
        public MainPage()
        {
            InitializeComponent();

            XLSheet sheet = _book.Sheets[0];

            for (int i = 0; i <= 9; i++)
            {
                sheet[i, 0].Value = i + 1;
                sheet[i, 1].Value = 10 - i;
            }
           }
}
```

6. Then add the following code that will save the book to a CSV file when the user clicks the button. The text box will show updates when the file is saving and when it is finished saving.

```
private void _btnSave_Click(object sender, RoutedEventArgs e)
        {
            var dlg = new SaveFileDialog();
            dlg.Filter = "Comma-Separated Values (*.csv)|*.csv";
            if (dlg.ShowDialog().Value)
            {
                try
                {
                    // information
                    txt_Status.Text = string.Format("Saving {0}...",
dlg.SafeFileName);

                    // save workbook
                    using (var stream = dlg.OpenFile())
                    {
                        _book.Sheets[0].SaveCsv(stream);
                    }
                    txt_Status.Text = string.Format("Saved {0}", dlg.SafeFileName); ;
                }
                catch (Exception x)
                {
                    MessageBox.Show(x.Message);
                }
            }
        }
```

7. Click **View | Designer** to switch back to Design view.
8. Select the button and in the Visual Studio Properties window, click the **Events** tab.
9. Click the drop-down arrow next to the **Click** event and select **_btnSave_Click** to link the event to the code you just added.
10. Press **F5** to run the project, and then click the button.
11. Save the file to the desired location and then open the file. It should look similar to this image:


test.csv - Notepad
File Edit Format View Help
1;10;
2;9;
3;8;
4;7;
5;6;
6;5;
7;4;
8;3;
9;2;
10;1;