ComponentOne

Excel for WPF and Silverlight

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

Excel for WPF and Silverlight Overview

Your Excel® data is just a simple command away from any of your .NET applications with **Excel for WPF and Silverlight (C1Excel)** – you don't even need to have Microsoft® Excel installed! Create or load XLS files for Excel 97 and later. **Excel for WPF and Silverlight** supports the new Office 2007 OpenXml format, which allows you to save smaller, compressed XLSX files.

The main component in **Excel for WPF and 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 WPF and Silverlight Edition

Getting Started

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

Key Features

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

- Save or load a workbook with one command

  Excel for WPF and 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:

  C#

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

- Use **Excel for WPF and Silverlight** to export to XLS files

  Other ComponentOne components use **C1Excel** to export XLS files. For example, **C1Report** uses **C1Excel** to create XLS versions of reports so they can be viewed and edited by anyone with a copy of Microsoft Excel.

- Reads and writes .xls and .xlsx files without using Microsoft Excel

  Excel for WPF and Silverlight reads and writes .xls (Excel 97 and later) and xlsx (OpenXml format) files, the latter of which can be reused and easily exchanged or compressed to create smaller file sizes. You don't even need to have Microsoft Excel installed.

- Create and position images within a cell

  Not only can you add images to cells, but now you can specify the cell size, the position of the image within the cell and whether the image is scaled, clipped, or stretched to fit the cell.

- 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.

- Add images to the header and footer of a sheet

  Use properties in the XLPrintSettings class to add images to the left, center, or right part of a sheet's header or footer.

QuickStart: Excel for WPF

This quick start guide will familiarize you with some of the features of **Excel for WPF**. In this quick start you will learn how to create a workbook, 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 add a reference to the C1.WPF.Excel.dll to your form.

1. Create a new WPF project.
2. Select the **Add Reference** option from the **Project** menu of your project. The **Add Reference** dialog box appears.
3. Browse to locate and select the **C1.WPF.Excel.dll** and click **OK**.
4. Double-click the form to add the **Form1_Load** event and switch to code view. 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.WPF.Excel namespace.

Visual Basic

```
Imports C1.WPF.C1Excel
```

C#

```
using C1.WPF.Excel;
```

Now you can create and begin adding content to a **C1XLBook**.

Step 2 of 4: Adding Content to a C1XLBook

While you are still in code view in the Visual Studio project, add the following code to create the book and its content:

Visual Basic

```
Dim book As New C1XLBook()
Dim i As Integer
   Dim sheet As 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#

```
C1XLBook book = new C1XLBook();
int i;
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;
  }
```

Step 3 of 4: Formatting the Content

Next we will format the content using styles. The code in this step should be added after the code from Step 2 of 4.

1. Add the following code to create two new styles: stye1 and style2.

Visual Basic

```
' Add style 1.
Dim style1 As New XLStyle(book)
style1.Font = New XLFont("Tahoma", 9, true, false)
style1.ForeColor = Colors.RoyalBlue
' Add style 2.
Dim style2 As New XLStyle(book)
style2.Font = New XLFont("Tahoma", 9, false, true)
style2.BackColor = Colors.RoyalBlue
style2.ForeColor = Colors.White
```

C#

```
// Add style 1.
XLStyle style1 = new XLStyle(book);
style1.Font = new XLFont("Tahoma", 9, true, false);
style1.ForeColor = Colors.RoyalBlue;
// Add style 2.
XLStyle style2 = new XLStyle(book);
style2.Font = new XLFont("Tahoma", 9, false, true);
style2.BackColor = Colors.RoyalBlue;
style2.ForeColor = Colors.White;
```

2. Then add the following code to apply the new styles to the content.

Visual Basic

```
For i = 0 To 9
    ' Apply styles to the content.
    If (i + 1) Mod 2 = 0 Then
        sheet(i, 0).Style = style2
        sheet(i, 1).Style = style1
        sheet(i, 2).Style = style2
    Else
        sheet(i, 0).Style = style1
        sheet(i, 1).Style = style2
        sheet(i, 2).Style = style1
    End If
Next i
```

C#

```
for (i = 0; i <= 9; i++)
            {
                // Apply styles to the content.
                if ((i + 1) % 2 == 0)
                {
                    sheet[i, 0].Style = style2;
                    sheet[i, 1].Style = style1;
                    sheet[i, 2].Style = style2;
                }
                else
                {
                    sheet[i, 0].Style = style1;
                    sheet[i, 1].Style = style2;
                    sheet[i, 2].Style = style1;
                }
            }
```

Step 4 of 4: Saving and Opening the XLS File

Finally, add the following code to save and load the Excel workbook. This code should be added after the code from Step 3 of 4 within the **Form_Load** event.

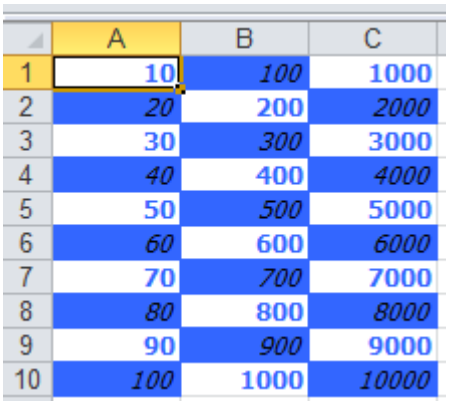
Visual Basic

```
C1XLBook1.Save("c:\mybook.xls")
System.Diagnostics.Process.Start("C:\mybook.xls")
```

C#

```
c1XLBook1.Save(@"c:\mybook.xls");
System.Diagnostics.Process.Start(@"c:\mybook.xls");
```

Run the program and observe:



	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

Formatted content is added to the workbook.

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

QuickStart: Excel for Silverlight

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
```

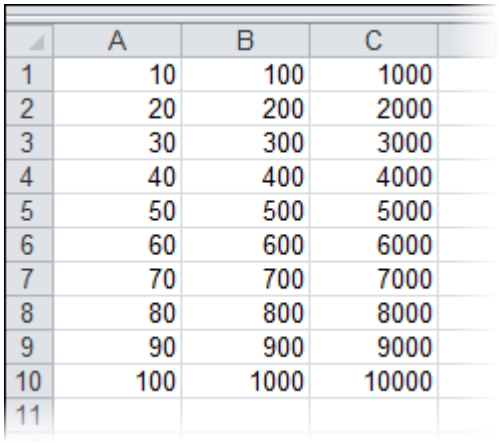
C#

```
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 WPF and Silverlight

The following topics explain how to create an XLS file, as well as describe the main **Excel for WPF and 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 XLS file using **Excel for WPF and Silverlight**, three steps are required:

1. Add a reference to C1.WPF.Excel.dll 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#

```
// step 1: create a new workbook to be saved
C1XLBook book = new C1XLBook();

// 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 and open the file
book.Save(@"C:\MyFiles\Test\test.xls");
System.Diagnostics.Process.Start(@"C:\MyFiles\Test\test.xls");
```

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.

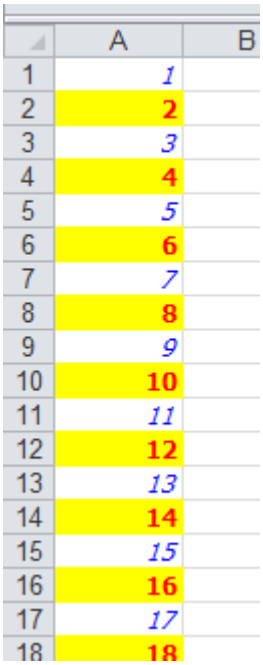
C#

```
// step 1: create a new workbook to be saved
C1XLBook book = new C1XLBook();
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 and open the file
book.Save(@"C:\MyFiles\Test\test.xls");
System.Diagnostics.Process.Start(@"C:\MyFiles\Test\test.xls");
```

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:



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 = C1XLBook1.Sheets(0)
Dim row As XLRow = sheet.Rows(1000)
```

C#

```
XLSheet sheet = c1XLBook1.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).

C1Excel Task-Based Help

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

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

Adding a Comment to a Cell (WPF)

To add a comment to a cell, complete the following steps:

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

   ```
   <// Create a new workbook C1XLBook book = new C1XLBook();
   ```

2. Add text to a cell in the worksheet using the following code:

   ```
   book.Sheets[0][2, 3].Value = "test";
   ```

3. Add a comment to the XLCommentCollection and create a box to show it in using the following code:

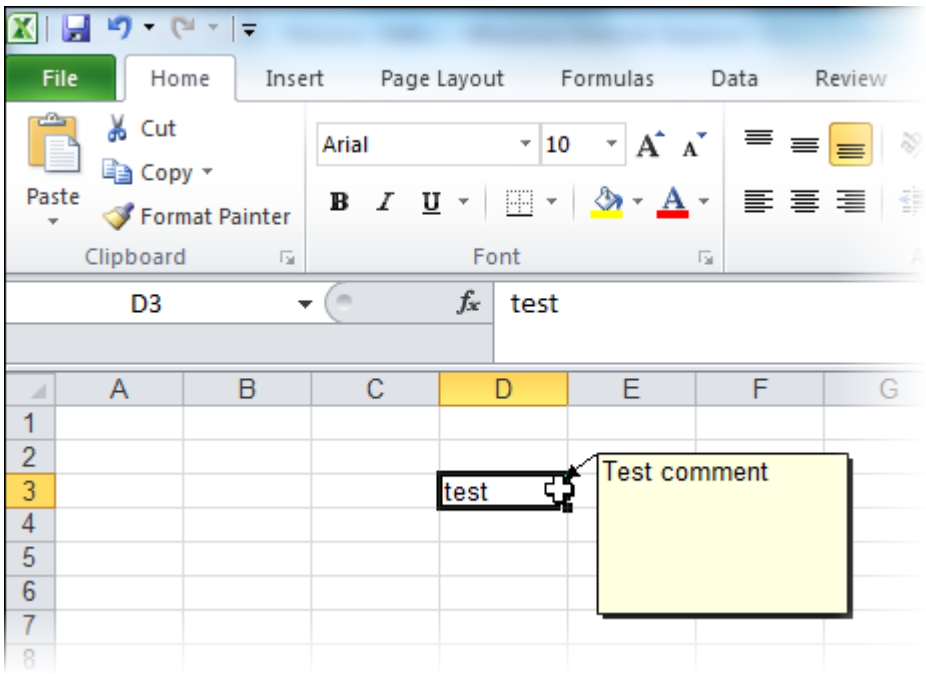
   ```
   book.Sheets[0].Comments.Add(2, 3, "John", "Test comment");
   book.Sheets[0].Comments[0].TextBox.Rectangle = new Rect(220, 210, 1900, 1200);
   ```

4. Save and open the book:

   C#

   ```
   // Save and open the file
    book.Save(@"C:\test.xlsx");
    System.Diagnostics.Process.Start(@"C:\test.xlsx");
   ```

5. Run the program. The spreadsheet will open and look similar to the following:



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.WPF.Excel.dll or C1.Silverlight.Excel.dll and create a **C1XLBook**.

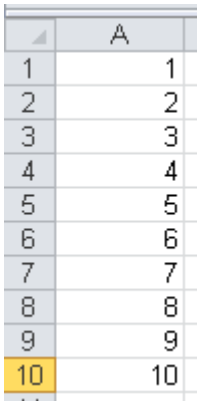
```
C#
// Create a new workbook to be saved
 C1XLBook book = new C1XLBook();
```

2. Add values to the first ten cells.

```
C#
// 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;
   }
```

3. Save and open the workbook. The code looks like the following.

```
C#
// Save and open the file
book.Save(@"C:\test.xls");
System.Diagnostics.Process.Start(@"C:\test.xls");
```



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

Adding an Image to a Cell (WPF)

Images can be added to a sheet or cell using one of the following methods.

Method 1: Assign an image directly to a cell's XLCell.Value property.

Using this method, the image is added to the sheet and kept at its original size. The upper left corner of the image coincides with the upper left corner of the specified cell.

1. Load an existing workbook or add some content to a new workbook.

```
Visual Basic
```

```
Dim wb As New C1XLBook
wb.Load("C:\Project\WorkBook1.xls")
```

C#

```
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\Project\WorkBook1.xls");
```

2. Specify the image and assign it to the cell's Value property.

Visual Basic

```
Dim img As WriteableBitmap = new WriteableBitmap(new BitmapImage(new
Uri("MyImage.bmp", UriKind.Relative)));
Dim sheet As XLSheet = wb.Sheets("Forecasting Report")
sheet(0, 0).Value = img
```

C#

```
WriteableBitmap img = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
XLSheet sheet = wb.Sheets["Forecasting Report"];
sheet[0,0].Value = img;
```

3. Save and open the new book:

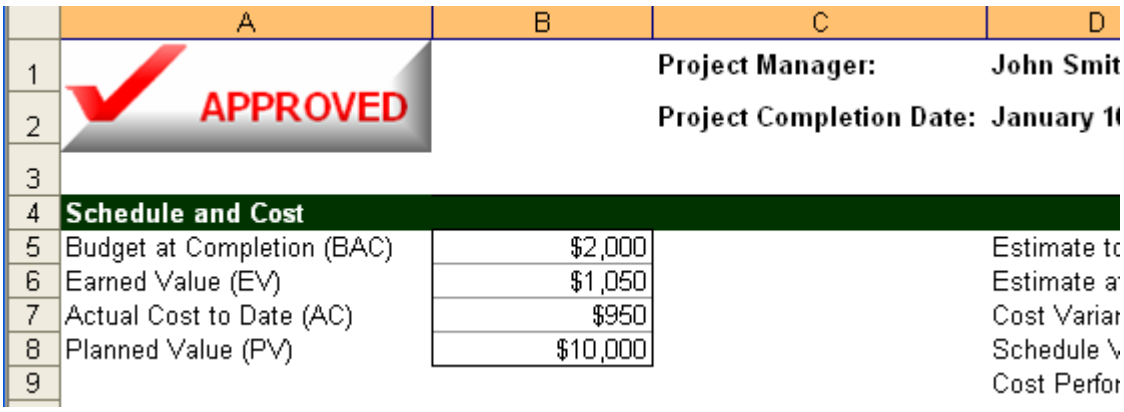
Visual Basic

```
wb.Save("C:\Project\WorkBook1.xls ")
System.Diagnostics.Process.Start("C:\Project\WorkBook1.xls")
```

C#

```
wb.Save(@"C:\Project\WorkBook1.xls");
System.Diagnostics.Process.Start(@"C:\Project\WorkBook1.xls");
```

In this example, the image replaces the value in the first cell, and it appears at its original size in the first cell.



Method 2: Create an XLPictureShape object, set its properties, and assign it to a cell's XLCell.Value property.

This second method allows you to customize the image by specifying its size, rotation angle, brightness, contrast, border, and more.

1. Load an existing workbook or add some content to a new workbook.

Visual Basic

```
Dim wb As New C1XLBook
wb.Load("C:\Project\WorkBook1.xls")
```

C#

```
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\Project\WorkBook1.xls");
```

2. Create an XLPictureShape object, set some of its properties and assign it to a cell's Value property.

Visual Basic

```
Dim img As WriteableBitmap
 = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
Dim pic As New XLPictureShape(img, 1500, 1500)
pic.Rotation = 30.0F
pic.LineColor = Color.DarkRed
pic.LineWidth = 100

' assign the pic to the first cell of the specified sheet
Dim sheet As XLSheet = wb.Sheets("Forecasting Report")
sheet(0, 0).Value = pic
```

C#

```
WriteableBitmap img = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
XLPictureShape pic = new XLPictureShape(img, 1500, 1500);
pic.Rotation = 30.0f;
pic.LineColor = Color.DarkRed;
pic.LineWidth = 100;

// assign the pic to the first cell of the specified sheet
XLSheet sheet = wb.Sheets("Forecasting Report");
sheet[0,0].Value = pic;
```

3. Save and open the book.

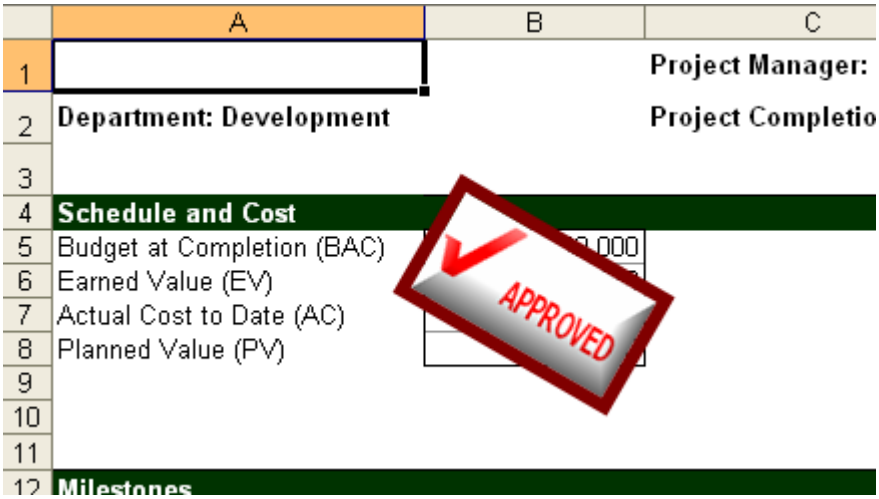
Visual Basic

```
wb.Save("C:\Project\WorkBook1.xls ")
System.Diagnostics.Process.Start("C:\Project\WorkBook1.xls")
```

C#

```
wb.Save(@"C:\Project\WorkBook1.xls");
System.Diagnostics.Process.Start(@"C:\Project\WorkBook1.xls");
```

In this example, the image replaces the value in the first cell, is rotated 30°, and has a dark red border. Since we have specified the horizontal and vertical position of the image, it does not appear in the first cell.



Method 3: Create an XLPictureShape object, set its properties, and add it to a sheet's ShapeCollection.

This method uses the XLPictureShape constructor to specify the image boundaries in sheet coordinates. The shape is added directly to the sheet's ShapeCollection, rather than to a specific cell.

1. Load an existing workbook or add some content to a new workbook.

Visual Basic

```
Dim wb As New C1XLBook
wb.Load("C:\Project\WorkBook1.xls")
```

C#

```
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\Project\WorkBook1.xls");
```

2. Create an XLPictureShape object, set some of its properties and assign it to a sheet's ShapeCollection.

Visual Basic

```
Dim img As WriteableBitmap
 = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
Dim pic As New XLPictureShape(img, 3000, 3500, 2500, 900)
pic.Rotation = 30.0F
pic.LineColor = Color.DarkRed
pic.LineWidth = 100

' add the pic to specified sheet's ShapeCollection
Dim sheet As XLSheet = wb.Sheets("Forecasting Report")
sheet.Shapes.Add(pic)
```

C#

```
WriteableBitmap img = new WriteableBitmap(new BitmapImage(new Uri("MyImage.bmp",
UriKind.Relative)));
XLPictureShape pic = new XLPictureShape(img, 3000, 3500, 2500, 900);
pic.Rotation = 30.0f;
pic.LineColor = Color.DarkRed;
pic.LineWidth = 100;

// add the pic to specified sheet's ShapeCollection
XLSheet sheet = wb.Sheets("Forecasting Report");
sheet.Shapes.Add(pic);
```

3. Save and open the book.

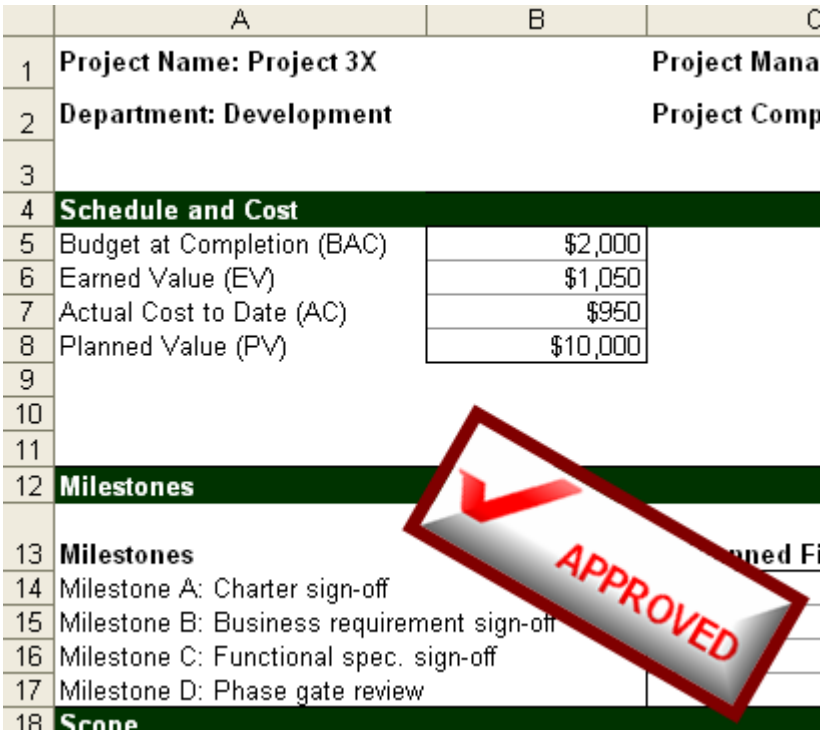
Visual Basic

```
wb.Save("C:\Project\WorkBook1.xls ")
System.Diagnostics.Process.Start("C:\Project\WorkBook1.xls")
```

C#

```
wb.Save(@"C:\Project\WorkBook1.xls");
System.Diagnostics.Process.Start(@"C:\Project\WorkBook1.xls");
```

In this example, the shape was added to the sheet's ShapeCollection; therefore, the image does not replace the value in the first cell. Here we specified the height and width of the image, as well as the horizontal and vertical positioning.



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.WPF.Excel.dll or C1.Silverlight.Excel.dll and create a **C1XLBook**.

```
C#
// Create a new workbook to be saved
 C1XLBook book = new C1XLBook();
```

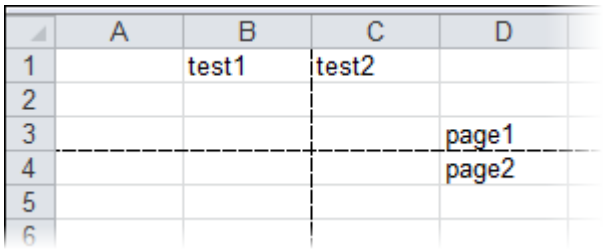
2. Add some text values and page breaks using the following code:

```
C#
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";
```

3. Save and open the .xlsx file.

```
C#
// Save and open the file
book.Save(@"C:\test.xlsx");
System.Diagnostics.Process.Start(@"C:\test.xlsx");
```

4. In Excel, select the **Page Layout** tab, and select the **Print** checkbox under **Gridlines**. The worksheet should look similar to the following:



Copying Rows from One Book to Another (WPF)

To copy the rows of a sheet from one book to a second book, complete the following steps:

1. Add a reference to C1.WPF.Excel.dll.
2. Load an existing book.

```
C#
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\test.xlsx");
```

3. Create a new workbook with an **XLSheet** named **Test**.

```
C#
```

```
C1XLBook xb = new C1XLBook();
xb.Sheets.Add("Test");
```

4. Copy each row from the sheet of the existing book to the new workbook **Test** sheet.

C#

```
XLSheet source = wb.Sheets[0];
            XLSheet dest = xb.Sheets["Test"];
            for (int row = 0; row <= source.Rows.Count - 1; row++)
            {
                for (int col = 0; col <= source.Columns.Count - 1; col++)
                {
                    dest[row, col].Value = source[row, col].Value;
                }
            }
```

5. Save and open the new workbook.

C#

```
// Save and open the file
xb.Save(@"C:\test2.xlsx");
System.Diagnostics.Process.Start(@"C:\test2.xlsx");
```

6. Open the new book. The rows from the first book have been added to the new book.

Creating Subtotals

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

1. Select **View | Code** and add one of the following statements at the top of the form:
   - `Import C1.C1Excel` (Visual Basic)
   - `using C1.C1Excel;` (C#)
2. Add a reference to C1.WPF.Excel.dll or C1.Silverlight.Excel.dll and create a **C1XLBook**.

C#

```
// Create a new workbook
C1XLBook book = new C1XLBook();
```

3. Add the following code to the **Form_Load** event:

Visual Basic

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
   Dim book As New C1XLBook()
   Dim sheet As XLSheet = book.Sheets(0)
   Dim totalStyle As New XLStyle(book)
   totalStyle.Font = New XLFont("Arial", 10, true, false)
   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
```

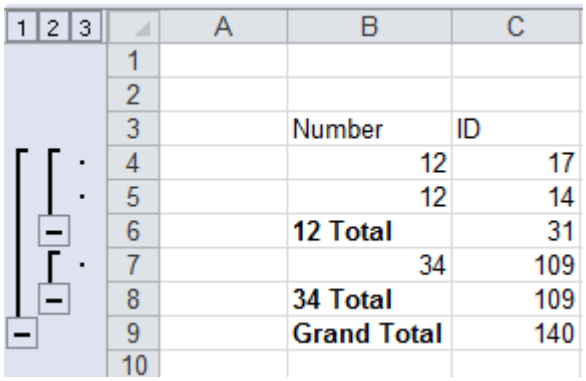
```
    sheet.Rows(4).Visible = False
    sheet(5, 1).Value = "12 Total"
    sheet(5, 1).Style = totalStyle
    sheet(5, 2).Value = 31
    sheet(5, 2).Formula = "SUBTOTAL(9,C4:C5)"
    sheet.Rows(5).OutlineLevel = 1
    sheet(6, 1).Value = 34
    sheet(6, 2).Value = 109
    sheet.Rows(6).OutlineLevel = 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
    book.Save("c:\mybook.xls")
    System.Diagnostics.Process.Start("C:\mybook.xls")
End Sub
```

C#

```
private void Form1_Load(object sender, EventArgs e)
        {
            C1XLBook book = new C1XLBook();
            XLSheet sheet = book.Sheets[0];
            XLStyle totalStyle = new XLStyle(book);
            totalStyle.Font = new XLFont("Arial", 10, true, false);
            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;

            book.Save(@"c:\mybook.xls");
            System.Diagnostics.Process.Start(@"C:\mybook.xls");
        }
```

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



	A	B	C
1			
2			
3		Number	ID
4		12	17
5		12	14
6		**12 Total**	31
7		34	109
8		**34 Total**	109
9		**Grand Total**	140
10			

The SUBTOTAL formulas get the sum of the specified rows.

Formatting Cells

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

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

C#

```
// Create a new workbook to be saved
C1XLBook book = new 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.

C#

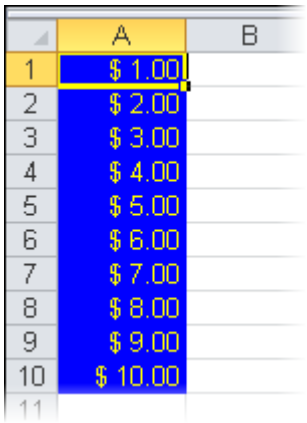
```
// Create a new style
 XLStyle style1 = new XLStyle(book);
 style1.ForeColor = Colors.Yellow;
 style1.BackColor = Colors.Blue;
 style1.Format = "$ .00";
// Add content and apply styles to cells in first column of sheet
  XLSheet sheet = book.Sheets[0];
        int i;
        for (i = 0; i <= 9; i++)
          {
```

```
            sheet[i, 0].Value = i + 1;
            sheet[i, 0].Style = style1;
        }
```

3. Save and open the workbook.

C#

```
// Save and open the file
book.Save(@"C:\test.xls");
System.Diagnostics.Process.Start(@"C:\test.xls");
```



Importing and Exporting OpenXml Files (WPF)

Excel for WPF can now read and write Microsoft Excel 2007 OpenXml files. OpenXml is an open, standards-based format introduced by Microsoft in Office 2007. OpenXml files are easier to manipulate by applications because OpenXml is based on XML and is publicly documented, as opposed to proprietary binary formats, such as BIFF8. OpenXml files contain a number of XML files compressed using Zip compression. Because they are compressed, OpenXml files are usually much smaller than traditional document files, such as .doc and .xls files.

Excel for WPF can load and save data and formatting information in OpenXml files; however, formulas are not loaded or saved. They are copied in BIFF format as opaque, which is not supported by the OpenXML format at this time. If you load files containing formulas and then save them, the formulas will be removed. This is in contrast to the traditional .xls, or BIFF8, format, which preserves the formulas.

To support the OpenXml format, the C1XLBook Load and Save methods received overloads that take a **FileFormat** parameter that is used to specify the file format to use when loading or saving files.

If the file name is not specified, then **Excel for WPF** infers the file format from the file name extension: files with an "XLSX" and "ZIP" extension are loaded and saved as OpenXml files, by default. Other files are loaded and saved as BIFF8, or .xls, format.

For example:

C#

```
    // load and save relying on file extension
    book.Load("somefile.xls");    // load biff 8 file
    book.Save("somefile.xlsx");   // save file as OpenXml
    book.Save("somefile.zip");    // save file as OpenXml
    // load and save specifying the FileFormat
    book.Load("somefile.xls", FileFormat.Biff8);
    book.Save("somefile.xlsx", FileFormat.OpenXml);
```

You can also specify the format when loading or saving files to and from streams. If the **FileFormat** is not specified, then **Excel for WPF** uses the BIFF8 format as a default.

Note that there is a small behavior change implied here. Consider the statement below:

```
book.Save("somefile.xlsx");
```

In previous versions of **Excel for WPF**, this would save a BIFF8 file (with the wrong extension). Now, this will save an OpenXml file (with the correct extension). If you have code like this in your applications, you should change it to the following when upgrading:

```
// deliberately save file with wrong extension
book.Save("somefile.xlsx", FileFormat.Biff8);
```

To export a book to an OpenXml file, complete the following steps:

1. Load an existing book:

Visual Basic

```
Dim wb As New C1XLBook()
wb.Load("C:\test.xlsx")
' or
Dim wb As New C1XLBook()
wb.Load("C:\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml)
```

C#

```
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\test.xlsx");
// or
C1XLBook wb = new C1XLBook();
wb.Load(@"C:\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml);
```

2. Export the book to an OpenXml Format file:

Visual Basic

```
Dim wb As New C1XLBook()
' Add some content
Dim sheet As XLSheet = wb.Sheets(0)
Dim i As Integer
  For i = 0 To 9
    sheet(i,0).Value = i + 1
  Next i
' Export to OpenXml Format file
wb.Save("C:\test.xlsx")
' or
' Export to OpenXml Format file
wb.Save("C:\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml)
```

C#

```
C1XLBook wb = new C1XLBook();
// Add some content
XLSheet sheet = wb.Sheets[0];
 for (int i = 0; i <= 9; i++)
{
  sheet[i,0].Value = i + 1;
 }
// Export to OpenXml Format file
wb.Save(@"C:\test.xlsx");
// or
// Export to OpenXml Format file
wb.Save(@"C:\test.xlsx", C1.WPF.Excel.FileFormat.OpenXml);
```

Saving a CSV File (Silverlight)

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:

Visual Basic

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

C#

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

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

C#

```
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;
```

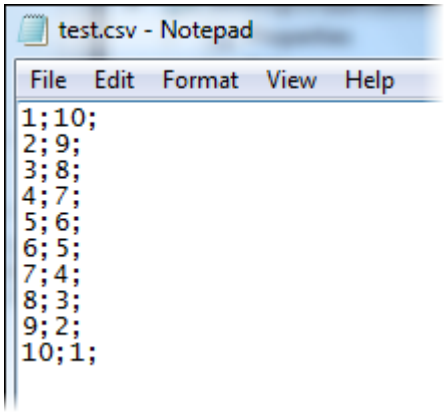
```
            }
        }
}
```

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.

C#

```
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:



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.WPF.Excel.dll or C1.Silverlight.Excel.dll and create a **C1XLBook**.

   C#

   ```
   // Create a new workbook
   C1XLBook book = new C1XLBook();
   ```

2. Add a simple formula using the following code:

   C#

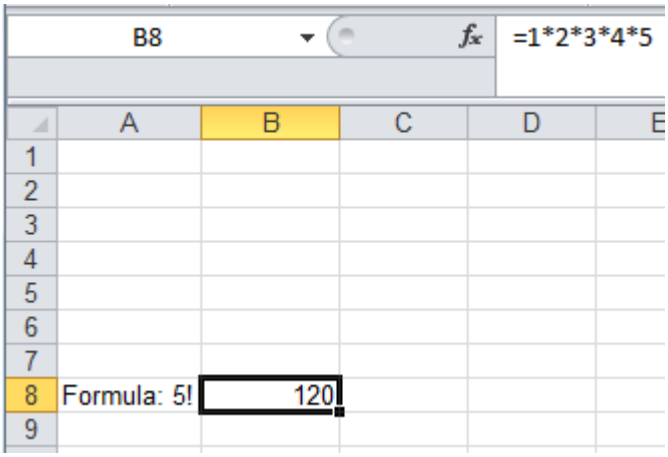
   ```
   XLSheet sheet = book.Sheets[0];
   // simple formula
     sheet[7, 0].Value = "Formula: 5!";
     sheet[7, 1].Value = 122;
     sheet[7, 1].Formula = "1*2*3*4*5";
     book.CalculationMode = CalculationMode.Auto;
   ```

3. Save and open the .xlsx file.

   C#

   ```
   // Save and open the file
   book.Save(@"C:\test.xlsx");
   System.Diagnostics.Process.Start(@"C:\test.xlsx");
   ```

   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**.



C1Excel 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 Studio Enterprise.

Please refer to pre-installed product samples through the following path:

Documents\ComponentOne Samples\WPF

or

Documents\ComponentOne Samples\Silverlight

WPF Sample	Description
Formulas	This sample demonstrates how to use formulas in your spreadsheets.
HelloWorld	This sample shows how to create a workbook.
Styles	This sample demonstrates how to use styles, such as fonts, borders, background colors and so on, in spreadsheets.

Silverlight 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.