ComponentOne

Excel for UWP

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 $2 5 (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 UWP

Export your data to Excel with **Excel for UWP** and for that you don't even need to have Microsoft Excel installed! Create and load XLS and XLSX files with this easy to use component. Access and modify data in individual sheets as if they were a simple grid composed of rows, columns and cells.

Getting Started with UWP

Help With UWP Edition

Getting Started

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

Key Features

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

- **Save or load a workbook with one command**

  Excel for UWP 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.

- **Supports CSV, XLS, and XLSX**

  Excel for UWP works with three file formats: XLS, XLSX (OpenXML format), and CSV (comma separated values). The OpenXml format allows you to save smaller, compressed files.

- **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:

  C#

```
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 Font("Courier New", 14);
 XLSheet sheet = C1XLBook.Sheets[0];
 sheet[0, 0].Value = DateTime.Now;
 sheet[0, 0].Style = style;
```

- **Add Cell Formulas**

  Excel for UWP fully supports cell formulas and binary parsing. The XLCell.Formula property allows you to specify a formula for the cell.

- **Grouping and Subtotals**

  Calculate subtotals for rows and columns. Declare outline level grouping in code to best display totals and subtotals.

Excel for UWP Quick Start

This quick start guide will familiarize you with some of the features of **Excel for UWP**. 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.UWP.Excel assembly.

1. Create a new Windows Store project. When the project opens, double-click the **MainPage.xaml** file to open it.
2. Right-click **References** in the Solution Explorer and select **Add Reference** from the list.
3. Browse to find C1.UWP.Excel.dll.
4. Click **OK** to add the assembly reference to your application.
5. In XAML View, place your cursor between the <Grid> </Grid> tags.
6. Add two standard Button controls and one standard TextBox control to the page.
7. Edit the markup for the first button so that it resembles the following:

Markup

```
<Button x:Name="HelloButton" Content="Click Hello" />
```

8. Edit the markup for the second button so that it resembles the following:

Markup

```
<Button x:Name="SaveButton" Content="Save" />
```

9. Edit the markup for the TextBox control so that it resembles the following:

Markup

```
<TextBox
   Name="_tbContent"
   Text="Empty"
   IsReadOnly="True"
   AcceptsReturn="True"
   FontFamily="Courier New"
   Background="White" Margin="465,10,242,722" />
```

10. Create an event named HelloButton_Click for HelloButton and switch to the code view of MainPage.xaml. This will also add a **HelloButton_Click** event to the code.
11. Switch back to Design View and double-click the SaveButton to add a **SaveButton_Click** event to the code.
12. Add the **using** (C#) statement to the code at the top of the form so you can use all names within the C1.Xaml.Excel namespace.

C#

```
using C1.Xaml.Excel;
using Windows.UI;
```

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

Step 2 of 4: Adding Content to a C1XLBook

In this step, you will add code to set up your project, and you will edit the **HelloButton_Click** event.

1. To define the C1XLBook, add the following code directly below the MainPage class:

C#

```
C1XLBook _book;
```

2. Add a new C1XLBook and some text to the TextBox control by adding the following code after the **InitializeComponent()** method:

C#

```
_book = new C1XLBook();
_tbContent.Text = "Empty workbook";
```

3. Add the following to define the **RefreshView()** method after the MainPage constructor:

C#

```
void RefreshView()
        {
        }
```

4. Locate the **HelloButton_Click** event next. Add the following code to create a new workbook, get the sheet that was created by default, create some styles for the data, and write some content and format the cells:

C#

```
// step 1: create a new workbook
            _book = new C1XLBook();

// step 2: get the sheet that was created by default, give it a name
            XLSheet sheet = _book.Sheets[0];
            sheet.Name = "Hello World";

// step 3: create styles for odd and even values
            XLStyle styleOdd = new XLStyle(_book);
            styleOdd.Font = new XLFont("Tahoma", 9, false, true);
            styleOdd.ForeColor = Color.FromArgb(255, 0, 0, 255);
            XLStyle styleEven = new XLStyle(_book);
            styleEven.Font = new XLFont("Tahoma", 9, true, false);
            styleEven.ForeColor = Color.FromArgb(255, 255, 0, 0);

// step 4: write content and format 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 5: allow user to save the file
            _tbContent.Text = "'Hello World' workbook has been created, you can
save it now.";
            RefreshView();
```

Step 3 of 4: Saving the XLSX File

Add the following code to save the Excel workbook. When you click the **SaveButton**, you will be able to save the project you created to any location.

1. Edit the **SaveButton_Click** event to resemble the following code:

C#

```
async void SaveButton_Click(object sender, RoutedEventArgs e)
        {

        }
```

2. Insert the following code into the **SaveButton_Click** event to handle saving the Excel workbook:

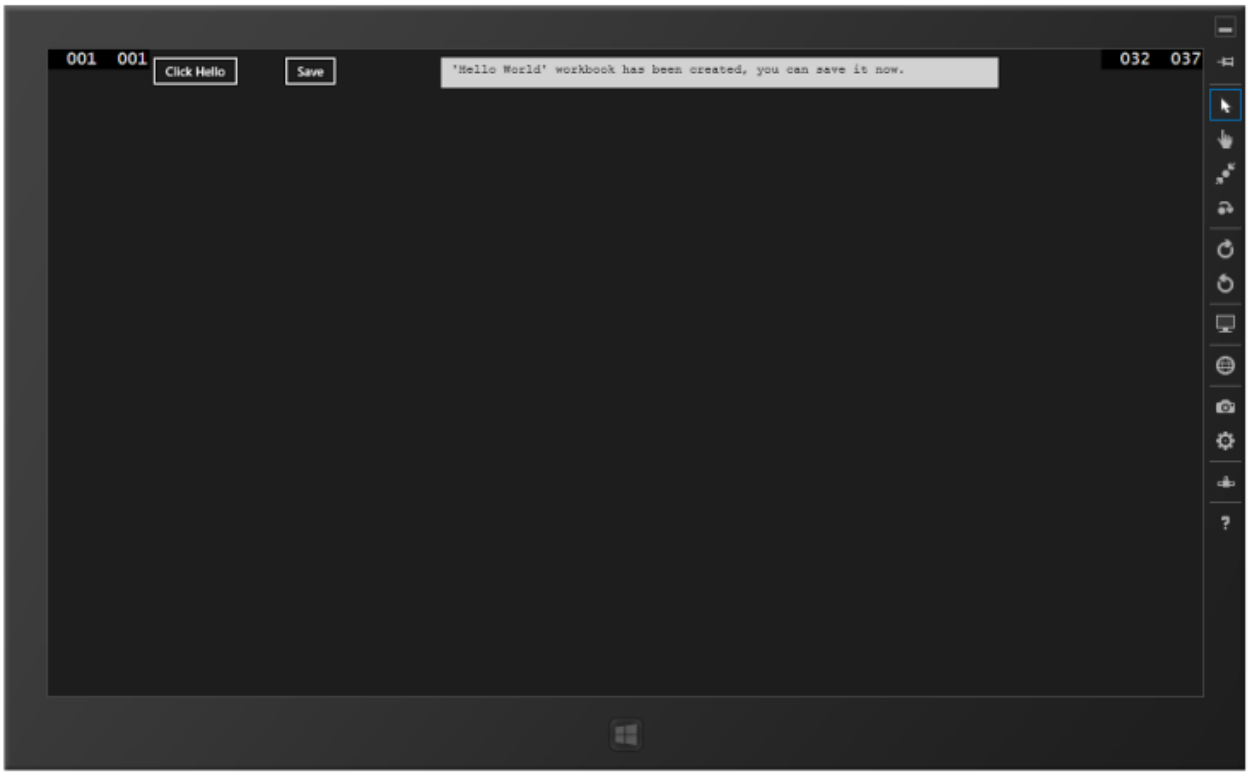
C#

```
Debug.Assert(_book != null);

var picker = new Windows.Storage.Pickers.FileSavePicker();
picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
picker.FileTypeChoices.Add("Open XML Excel file", new List < string > ()
{
  ".xlsx"
});
picker.FileTypeChoices.Add("BIFF Excel file", new List < string > ()
{
  ".xls"
});
picker.SuggestedFileName = "New Book";

var file = await picker.PickSaveFileAsync();
if (file != null)
{
  try
  {
    // step 1: save file
    var fileFormat = Path.GetExtension(file.Path).Equals(".xls") ?
FileFormat.OpenXmlTemplate : FileFormat.OpenXml;
    await _book.SaveAsync(file, fileFormat);
    // step 2: user feedback
    _tbContent.Text = string.Format("File has been saved to: {0}.", file.Path);
    RefreshView();
  } catch (Exception x)
  {
    _tbContent.Text = string.Format("EXCEPTION: {0}", x.Message);
  }
}
```

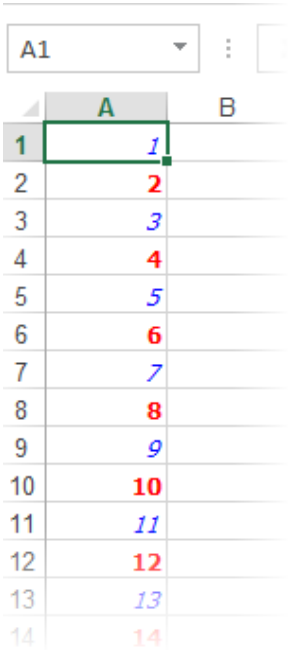
Step 4 of 4: Run the Program

Press F5 to run the application.

1. Click the **Click Hello** button to create the Hello Excel file. You should receive a message in the TextBox letting you know that it's been created:



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



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

Using Excel for UWP

The following topics explain how to create an XLSX file, as well as describe the main **Excel for UWP** 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 UWP**, three steps are required:

1. Add a reference to C1UWP.Excel.dll and create a **C1XLBook**.
2. Add content to the sheets. Each sheet contains cells (XLCell objects) that have a XLCell.Value and a XLCell.Style property.
3. Save the book to a file using the C1XLBook.SaveAsync method.

   The code starts by creating a new workbook, and then getting the XLSheet that is created automatically. The third step creates the styles that will be applied to even and odd values before content is written and cells are formatted.

   > 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.

   The style applied in step three creates a sheet where even numbers are shown in bold red characters and odd numbers are shown in italic blue.

C#

```
private void HelloButton_Click(object sender, RoutedEventArgs e)
        {
            // step 1: create a new workbook
            _book = new C1XLBook();

            // step 2: get the sheet that was created by default, give it a name
            XLSheet sheet = _book.Sheets[0];
            sheet.Name = "Hello World";

            // step 3: create styles for odd and even values
            XLStyle styleOdd = new XLStyle(_book);
            styleOdd.Font = new XLFont("Tahoma", 9, false, true);
            styleOdd.ForeColor = Color.FromArgb(255, 0, 0, 255);
            XLStyle styleEven = new XLStyle(_book);
            styleEven.Font = new XLFont("Tahoma", 9, true, false);
            styleEven.ForeColor = Color.FromArgb(255, 255, 0, 0);

            // step 4: write content and format 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 5: allow user to save the file
```

```
            _tbContent.Text = "'Hello World' workbook has been created, you can
save it now.";
            RefreshView();
        }
```

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 XLSheet.Rows and XLSheet.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 XLSheet.DefaultRowHeight and XLSheet.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 XLSheet.GetCell method instead of the indexer.

XLCell objects have a XLCell.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 XLCell.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 UWP 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 UWP** features and get a good sense of what **Excel for UWP** can do.

Each task-based help topic also assumes that you have created a new Windows Store project and added the appropriate directives (using C1.Xaml.Excel**;** for C# and Imports C1.Xaml.Excel 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:

In XAML View

1. Right-click **References** in the Solution Explorer and select **Add Reference** from the list.
    1. Browse to find C1.UWP.Excel.dll.
    2. Click **OK** to add the assembly reference to your application.
2. In XAML View, place your cursor between the <Grid> </Grid> tags.
3. Add two standard Button controls and one standard **TextBox** control to the page.
    1. Edit the markup for the first button so that it resembles the following:

        Markup

        ```
        <Button x:Name="HelloButton" Content="Click Hello" />
        ```

    2. Edit the markup for the second button so that it resembles the following:

        Markup

        ```
        <Button x:Name="SaveButton" Content="Save" />
        ```

    3. Edit the markup for the TextBox control so that it resembles the following:

        Markup

        ```
        <TextBox
           Name="_tbContent"
           Text="Empty"
           IsReadOnly="True"
           AcceptsReturn="True"
           FontFamily="Courier New"
           Background="White" Margin="465,10,242,722" />
        ```

4. Create an event named HelloButton_Click for HelloButton and switch to the code view of MainPage.xaml. This will also add a **HelloButton_Click** event to the code.
5. Switch back to Design View and double-click the SaveButton to add a **SaveButton_Click** event to the code. This will open the Code View.

In Code View

1. Add a using statement to the top of the page:

C#

```
using C1.Xaml.Excel;
```

2. Add the following code to the **MainPage** class so that it resembles the following:

C#

```
public sealed partial class MainPage : Page
    {
        C1XLBook _book;
    }
```

3. Create a C1XLBook by adding the following code to the **InitializeComponent()** method:

C#

```
_book = new C1XLBook();
```

4. Add the RefreshView() method. You will call this method later in the code:

C#

```
void RefreshView()
{
}
```

5. Get the default sheet and name it.
6. Create styles for the odd and even values.
7. Add values to the first ten cells.

C#

```
// step 1: create a new workbook
            _book = new C1XLBook();

            // step 2: get the sheet that was created by default, give it a name
            XLSheet sheet = _book.Sheets[0];
            sheet.Name = "Hello World";

            // step 3: create styles for odd and even values
            XLStyle styleOdd = new XLStyle(_book);
            styleOdd.Font = new XLFont("Tahoma", 9, false, true);
            styleOdd.ForeColor = Color.FromArgb(255, 0, 0, 255);
            XLStyle styleEven = new XLStyle(_book);
            styleEven.Font = new XLFont("Tahoma", 9, true, false);
            styleEven.ForeColor = Color.FromArgb(255, 255, 0, 0);

            // step 4: write content and format 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;
```

```
}
```

8. Save the workbook so that you can open it in Excel.

C#

```
async void SaveButton_Click(object sender, RoutedEventArgs e)
        {
            Debug.Assert(_book != null);

            var picker = new Windows.Storage.Pickers.FileSavePicker();
            picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
            picker.FileTypeChoices.Add("Open XML Excel file", new List<string>()
{ ".xlsx" });
            picker.FileTypeChoices.Add("BIFF Excel file", new List<string>() {
".xls" });
            picker.SuggestedFileName = "New Book";

            var file = await picker.PickSaveFileAsync();
            if (file != null)
            {
                try
                {
                    // step 1: save file
                    var fileFormat = Path.GetExtension(file.Path).Equals(".xls")
? FileFormat.Biff8 : FileFormat.OpenXml;
                    await _book.SaveAsync(file, fileFormat);
                    // step 2: user feedback
                    _tbContent.Text = string.Format("File has been saved to:
{0}.", file.Path);
                    RefreshView();
                }
                catch (Exception x)
                {
                    _tbContent.Text = string.Format("EXCEPTION: {0}",
x.Message);
                }
            }
        }
```

Formatting Cells

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

In XAML View

1. Right-click **References** in the Solution Explorer and select **Add Reference** from the list.
    1. Browse to find C1.UWP.Excel.dll.
    2. Click **OK** to add the assembly reference to your application.

2. In XAML View, place your cursor between the <Grid> </Grid> tags.
3. Add two standard Button controls and one standard TextBox control to the page.
    1. Edit the markup for the first button so that it resembles the following:

```
Markup
<Button x:Name="HelloButton" Content="Click Hello" />
```

    2. Edit the markup for the second button so that it resembles the following:

```
Markup
<Button x:Name="SaveButton" Content="Save" />
```

    3. Edit the markup for the TextBox control so that it resembles the following:

```
Markup
<TextBox
   Name="_tbContent"
   Text="Empty"
   IsReadOnly="True"
   AcceptsReturn="True"
   FontFamily="Courier New"
   Background="White" Margin="465,10,242,722" />
```

4. Create an event named HelloButton_Click for HelloButton and switch to the code view of MainPage.xaml. This will also add a **HelloButton_Click** event to the code.
5. Switch back to Design View and double-click the SaveButton to add a **SaveButton_Click** event to the code. This will open the Code View.

In Code View

1. Add a using statement to the top of the page:

```
C#
using C1.Xaml.Excel;
```

2. Add the following code to the **MainPage** class so that it resembles the following:

```
C#
public sealed partial class MainPage : Page
    {
        C1XLBook _book;
    }
```

3. Create a C1XLBook by adding the following code to the **InitializeComponent()** method:

```
C#
_book = new C1XLBook();
```

4. Add the **RefreshView()** method. You will call this method later in the code:

C#

```
void RefreshView()
        {
        }
```

5. Get the sheet that was created by default and give it a name.
6. Add some content to the workbook, create a new style and apply the styles to the cells.

C#

```
private void HelloButton_Click(object sender, RoutedEventArgs e)
        {
            // step 1: create a new workbook
            _book = new C1XLBook();

            // step 2: get the sheet that was created by default, give it a name
            XLSheet sheet = _book.Sheets[0];
            sheet.Name = "Hello World";

            // step 3: set the forecolor and backcolor properties and add some
formatting to the cells.
            XLStyle style1 = new XLStyle(_book);
            style1.ForeColor = Colors.Yellow;
            style1.BackColor = Colors.Blue;
            style1.Format = "$ .00";

            // step 4: write content and format into some cells
            int i;
                for (i = 0; i <= 9; i++)
                {
                    sheet[i, 0].Value = i + 1;
                    sheet[i, 0].Style = style1;
                }
```

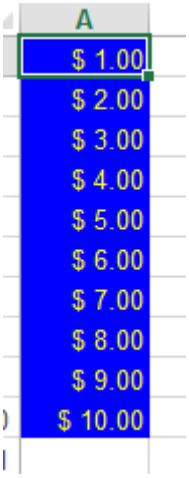
7. Save the workbook.

C#

```
async void SaveButton_Click(object sender, RoutedEventArgs e)
        {
         Debug.Assert(_book != null);

            var picker = new Windows.Storage.Pickers.FileSavePicker();
            picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
            picker.FileTypeChoices.Add("Open XML Excel file", new List<string>()
{ ".xlsx" });
            picker.FileTypeChoices.Add("BIFF Excel file", new List<string>() {
".xls" });
            picker.SuggestedFileName = "New Book";
```

```
            var file = await picker.PickSaveFileAsync();
            if (file != null)
            {
                try
                {
                    // step 1: save file
                    var fileFormat = Path.GetExtension(file.Path).Equals(".xls")
? FileFormat.Biff8 : FileFormat.OpenXml;
                    await _book.SaveAsync(file, fileFormat);
                    // step 2: user feedback
                    _tbContent.Text = string.Format("File has been saved to:
{0}.", file.Path);
                    RefreshView();
                }
                catch (Exception x)
                {
                    _tbContent.Text = string.Format("EXCEPTION: {0}",
x.Message);
                }
            }

        }
```

When you open the file you've saved, it should resemble the following image:



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 XLColumn.PageBreak and XLRow.PageBreak properties.

In XAML View

1. Right-click **References** in the Solution Explorer and select **Add Reference** from the list.

    1. Browse to find C1.UWPI.Excel.dll.
    2. Click **OK** to add the assembly reference to your application.

2. In XAML View, place your cursor between the <Grid> </Grid> tags.
3. Add two standard **Button** controls and one standard **TextBox** control to the page.
    1. Edit the markup for the first button so that it resembles the following:

        Markup
        ```
        <Button x:Name="HelloButton" Content="Click Hello" />
        ```

    2. Edit the markup for the second button so that it resembles the following:

        Markup
        ```
        <Button x:Name="SaveButton" Content="Save" />
        ```

    3. Edit the markup for the TextBox control so that it resembles the following:

        Markup
        ```
        <TextBox
           Name="_tbContent"
           Text="Empty"
           IsReadOnly="True"
           AcceptsReturn="True"
           FontFamily="Courier New"
           Background="White" Margin="465,10,242,722" />
        ```

4. Create an event named HelloButton_Click for HelloButton and switch to the code view of MainPage.xaml. This will also add a **HelloButton_Click** event to the code.
5. Switch back to Design View and double-click the SaveButton to add a **SaveButton_Click** event to the code. This will open the Code View.

In Code View

1. Add a using statement to the top of the page:

    C#
    ```
    using C1.Xaml.Excel;
    ```

2. Add the following code to the **MainPage** class so that it resembles the following:

    C#
    ```
    public sealed partial class MainPage : Page
        {
            C1XLBook _book;
        }
    ```

3. Create a C1XLBook by adding the following code to the **InitializeComponent()** method:

    C#
    ```
    _book = new C1XLBook();
    ```

4. Add the **RefreshView()** method. You will call this method later in the code:

C#

```
void RefreshView()
        {
        }
```

5. Add some text values and page breaks.

C#

```
private void HelloButton_Click(object sender, RoutedEventArgs e)
        {
            // step 1: create a new workbook
            _book = new C1XLBook();

           // 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";

            // step 2: allow user to save the file
            _tbContent.Text = "'Hello World' workbook has been created, you can
save it now.";
            RefreshView();
        }
```

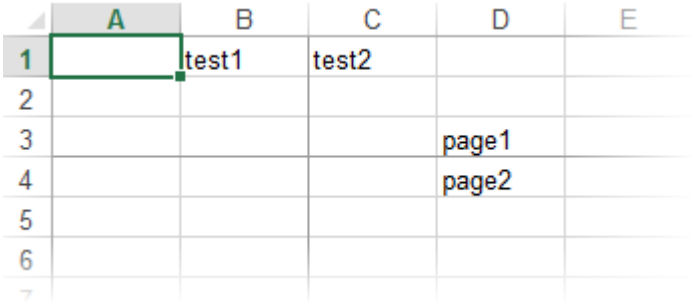
6. Save the workbook.

C#

```
async void SaveButton_Click(object sender, RoutedEventArgs e)
        {
            Debug.Assert(_book != null);

            var picker = new Windows.Storage.Pickers.FileSavePicker();
            picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
            picker.FileTypeChoices.Add("Open XML Excel file", new List<string>()
{ ".xlsx" });
            picker.FileTypeChoices.Add("BIFF Excel file", new List<string>() {
".xls" });
            picker.SuggestedFileName = "New Book";

            var file = await picker.PickSaveFileAsync();
            if (file != null)
            {
                try
                {
```

```
                    // step 1: save file
                    var fileFormat = Path.GetExtension(file.Path).Equals(".xls")
? FileFormat.Biff8 : FileFormat.OpenXml;
                    await _book.SaveAsync(file, fileFormat);
                    // step 2: user feedback
                    _tbContent.Text = string.Format("File has been saved to:
{0}.", file.Path);
                    RefreshView();
                }
                catch (Exception x)
                {
                    _tbContent.Text = string.Format("EXCEPTION: {0}",
x.Message);
                }
            }
        }
```

When you run your application, you'll be able to save the file you create. 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 C1XLBook.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).

In XAML View

1. Right-click **References** in the Solution Explorer and select **Add Reference** from the list.
    1. Browse to find C1.UWP.Excel.dll.
    2. Click **OK** to add the assembly reference to your application.
2. In XAML View, place your cursor between the <Grid> </Grid> tags.
3. Add two standard Button controls and one standard TextBox control to the page.
    1. Edit the markup for the first button so that it resembles the following:

Markup
`<Button x:Name="HelloButton" Content="Click Hello" />`

    2. Edit the markup for the second button so that it resembles the following:

Markup

```
<Button x:Name="SaveButton" Content="Save" />
```

3. Edit the markup for the TextBox control so that it resembles the following:

Markup

```
<TextBox
   Name="_tbContent"
   Text="Empty"
   IsReadOnly="True"
   AcceptsReturn="True"
   FontFamily="Courier New"
   Background="White" Margin="465,10,242,722" />
```

4. Create an event named HelloButton_Click for HelloButton and switch to the code view of MainPage.xaml. This will also add a **HelloButton_Click** event to the code.
5. Switch back to Design View and double-click the SaveButton to add a **SaveButton_Click** event to the code. This will open the Code View.

In Code View

To set the calculation mode, follow these steps:

1. Add a using statement to the top of the page:

C#

```
using C1.Xaml.Excel;
```

2. Add the following code to the **MainPage** class so that it resembles the following:

C#

```
public sealed partial class MainPage : Page
    {
        C1XLBook _book;
    }
```

3. Create a **C1XLBook** by adding the following code to the **InitializeComponent()** method:

C#

```
_book = new C1XLBook();
```

4. Add the RefreshView() method. You will call this method later in the code:

C#

```
void RefreshView()
        {
        }
```

5. Add a simple formula to perform a calculation.

C#

```
private void HelloButton_Click(object sender, RoutedEventArgs e)
        {
            // step 1: create a new workbook
            _book = new C1XLBook();

            // step 2: get the default sheet and give it a name
            XLSheet sheet = _book.Sheets[0];

            // step 3: add a 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;


            // step 4: allow user to save the file
            _tbContent.Text = "'Hello World' workbook has been created, you can
save it now.";
            RefreshView();
        }[ES5]
6.       Save the workbook.
·         C#
async void SaveButton_Click(object sender, RoutedEventArgs e)
        {
            Debug.Assert(_book != null);

            var picker = new Windows.Storage.Pickers.FileSavePicker();
            picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
            picker.FileTypeChoices.Add("Open XML Excel file", new List<string>()
{ ".xlsx" });
            picker.FileTypeChoices.Add("BIFF Excel file", new List<string>() {
".xls" });
            picker.SuggestedFileName = "New Book";

            var file = await picker.PickSaveFileAsync();
            if (file != null)
            {
                try
                {
                    // step 1: save file
                    var fileFormat = Path.GetExtension(file.Path).Equals(".xls")
? FileFormat.Biff8 : FileFormat.OpenXml;
                    await _book.SaveAsync(file, fileFormat);
                    // step 2: user feedback
                    _tbContent.Text = string.Format("File has been saved to:
{0}.", file.Path);
                    RefreshView();
                }
                catch (Exception x)
```

```
                {
                    _tbContent.Text = string.Format("EXCEPTION: {0}",
x.Message);
                }
            }
        }
```

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 to get a subtotal or your data.

In XAML View

Complete the following steps to create the XAML View for this topic:

1. Right-click **References** in the Solution Explorer and select **Add Reference** from the list.
    1. Browse to find C1.UWP.Excel.dll.
    2. Click **OK** to add the assembly reference to your application.
2. In XAML View, place your cursor between the <Grid> </Grid> tags.
3. Add two standard Button controls and one standard TextBox control to the page.
    1. Edit the markup for the first button so that it resembles the following:

        Markup
        ```
        <Button x:Name="HelloButton" Content="Click Hello" />
        ```

    2. Edit the markup for the second button so that it resembles the following:

        Markup
        ```
        <Button x:Name="SaveButton" Content="Save" />
        ```

    3. Edit the markup for the TextBox control so that it resembles the following:

        Markup
        ```
        <TextBox
           Name="_tbContent"
           Text="Empty"
           IsReadOnly="True"
           AcceptsReturn="True"
           FontFamily="Courier New"
           Background="White" Margin="465,10,242,722" />
        ```

4. Create an event named HelloButton_Click for HelloButton and switch to the code view of MainPage.xaml. This will also add a **HelloButton_Click** event to the code.
5. Switch back to Design View and double-click the SaveButton to add a **SaveButton_Click** event to the code. This will open the Code View.

In Code View

Complete the following steps to create the Code View for this topic:

1. Add a using statement to the top of the page:

```
C#
using C1.Xaml.Excel;
```

2. Add the following code to the **MainPage** class so that it resembles the following:

```
C#
public sealed partial class MainPage : Page
    {
        C1XLBook _book;
    }
```

3. Create a C1XLBook by adding the following code to the **InitializeComponent()** method:

```
C#
_book = new C1XLBook();
```

4. Add the **RefreshView()** method. You will call this method later in the code:

```
C#
void RefreshView()
        {
        }
```

5. Add code to format the cells.

```
C#
private void HelloButton_Click(object sender, RoutedEventArgs e)
        {
        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;

            // allow user to save the file
            _tbContent.Text = "'Hello World' workbook has been created, you can
save it now.";
            RefreshView();
            }
```

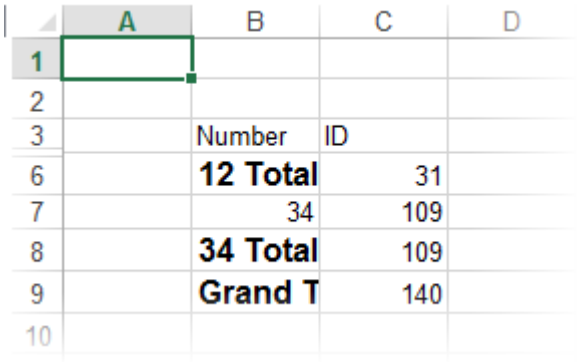
6. Save the workbook.

C#

```
async void SaveButton_Click(object sender, RoutedEventArgs e)
         {
             Debug.Assert(_book != null);

             var picker = new Windows.Storage.Pickers.FileSavePicker();
             picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
             picker.FileTypeChoices.Add("Open XML Excel file", new List<string>
() { ".xlsx" });
             picker.FileTypeChoices.Add("BIFF Excel file", new List<string>() {
".xls" });
             picker.SuggestedFileName = "New Book";

             var file = await picker.PickSaveFileAsync();
             if (file != null)
             {
                 try
                 {
                     // step 1: save file
                     var fileFormat =
Path.GetExtension(file.Path).Equals(".xls") ? FileFormat.Biff8 :
FileFormat.OpenXml;
                     await _book.SaveAsync(file, fileFormat);
                     // step 2: user feedback
```

```
                    _tbContent.Text = string.Format("File has been saved to:
{0}.", file.Path);
                    RefreshView();
                }
                catch (Exception x)
                {
                    _tbContent.Text = string.Format("EXCEPTION: {0}",
x.Message);
                }
            }
        }
```

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



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

The SUBTOTAL formulas get the sum of the specified rows.