ComponentOne

Excel for Windows Phone

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 Windows Phone Overview

Import your Excel data using **ComponentOne Excel™ for Windows Phone**. Create and load XLSX files with this easy-to-use class library. Access and modify data in individual sheets as if they were a simple grid composed of rows, columns, and cells. Take files to and from the cloud and device.

Help with ComponentOne Studio for Windows Phone

Getting Started

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

What's New

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

ComponentOne Excel for Windows Phone Assembly

The **C1.Phone.Excel.dll** assembly includes the following main class:

- **C1XLBook:** The **C1XLBook** class represents an Excel workbook containing one or more worksheets. It contains methods for loading and saving XLS files and exposes a collection of **XLSheet** objects that represent the individual worksheets.

Excel for Windows Phone Key Features

Make the most of **Excel for Windows Phone** by taking advantage of the following key features:

- **Save and Load Workbooks with One Command**

  Excel for Windows Phone includes one key component, C1XLBook. C1XLBook is easy to use, allowing you to use a single command to load or save a workbook from your C#/VB.NET code. Workbooks, which can contain multiple sheets, can be directly read to and written from memory streams too, so you don't have to use isolated storage files.

- **Read and Write Data in Individual Cells**

  After loading or creating an Excel Workbook into C1XLBook, you can access data in individual sheets as if they were a simple grid composed of rows, columns, and cells. Modify an existing spreadsheet and save the changes.

- **Import/Export XLSX**

  Excel for Windows Phone reads and writes one primary format: XLSX (OpenXml format). OpenXml format allows you to save smaller, compressed files.

- **Format the Data in Each Cell**

You can format cells to ensure that end-users enter correct data. The format associated with each cell is as easy to access as the data stored in the cell.

- **Add Cell Formulas**

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

- **Calculate Subtotals & Group Data**

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

- **Manage Cell Comments (Notes)**

  Add or edit cell comments in your Excel files using the easy API. Cell comments (or notes) allow you to attach a comment to a cell to inform, remind or explain the content of a cell or range of cells.

Using Excel for Windows Phone

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

1. Add a reference to C1.Phone.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 30.

- C#

```
// step 1: create a new workbook to be saved
SaveBook(book =>
              {
                  var sheet = book.Sheets[0];

                  // step 2: write content into some cells
                  for (int i = 0; i < 30; i++)
                  {
                      XLCell cell = sheet[i, 0];
                      cell.Value = i + 1;
                  }
              });

// step 3: save the file
   private void SaveBook(Action<C1XLBook> action)
           {
               try
               {
                   var book = new C1XLBook();
                   if (action != null)
                   {
                       action(book);
                   }

                   // save Excel content in base64
                   string excelBase64 = "";
                   using (var memoryStream = new MemoryStream())
                   {
                       book.Save(memoryStream);
                       memoryStream.Flush();
                       excelBase64 =
   Convert.ToBase64String(memoryStream.ToArray());
                   }

                   // create HTML file with inline Excel content
                   // and save it to the IsolatedStorage
```

```
                var file = 
IsolatedStorageFile.GetUserStoreForApplication();
                if (file.FileExists(FILE_NAME))
                    file.DeleteFile(FILE_NAME);
                using (var writer = new 
StreamWriter(file.CreateFile(FILE_NAME)))
                {
                    writer.Write(HTML_PART1);
                    writer.Write(excelBase64);
                    writer.Write(HTML_PART2);
                }

                // Open WebBrowser to open it
                Microsoft.Phone.Controls.WebBrowser webBrowser = new 
Microsoft.Phone.Controls.WebBrowser();
                //this.Content = webBrowser;
                //webBrowser.Navigate(new Uri(FILE_NAME, 
UriKind.Absolute));
                //Microsoft.Phone.Tasks.ShareLinkTask shareTask = new 
Microsoft.Phone.Tasks.ShareLinkTask();
                //shareTask.LinkUri = new Uri("data:application/vnd.ms-
excel;base64," + excelBase64, UriKind.RelativeOrAbsolute);
                //shareTask.Show();
            }
            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 assigns them values from 1 to 30.

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 btnStyles_Click(object sender, RoutedEventArgs e)
        {
            SaveBook(book =>
            {
                // step 1: get the sheet that was created by default, give it a name
                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 and format into some cells
                for (int i = 0; i < 30; i++)
                {
                    XLCell cell = sheet[i, 0];
                    cell.Value = i + 1;
                    cell.Style = ((i + 1) % 2 == 0) ? styleEven : styleOdd;
                }
            });
        }

private void SaveBook(Action<C1XLBook> action)
        {
            try
            {
                var book = new C1XLBook();
                if (action != null)
                {
                    action(book);
                }

                // save Excel content in base64
                string excelBase64 = "";
                using (var memoryStream = new MemoryStream())
                {
                    book.Save(memoryStream);
                    memoryStream.Flush();
                    excelBase64 = Convert.ToBase64String(memoryStream.ToArray());
                }

                // create HTML file with inline Excel content
                // and save it to the IsolatedStorage
                var file = IsolatedStorageFile.GetUserStoreForApplication();
                if (file.FileExists(FILE_NAME))
                    file.DeleteFile(FILE_NAME);
                using (var writer = new StreamWriter(file.CreateFile(FILE_NAME)))
                {
                    writer.Write(HTML_PART1);
                    writer.Write(excelBase64);
                    writer.Write(HTML_PART2);
                }

                // Open WebBrowser to open it
                Microsoft.Phone.Controls.WebBrowser webBrowser = new
Microsoft.Phone.Controls.WebBrowser();
                //this.Content = webBrowser;
                //webBrowser.Navigate(new Uri(FILE_NAME, UriKind.Absolute));


                //Microsoft.Phone.Tasks.ShareLinkTask shareTask = new
Microsoft.Phone.Tasks.ShareLinkTask();
                //shareTask.LinkUri = new Uri("data:application/vnd.ms-excel;base64," +
excelBase64, UriKind.RelativeOrAbsolute);
                //shareTask.Show();


            }
            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.

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.

The following example sets the column width, in twips.

```
      // column width in twips
      sheet.Columns[0].Width = 2000;
      sheet.Columns[1].Width = 2200;
```

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).

The following example code creates different styles for odd and even values in the workbook.

```
            // get the sheet that was created by default, give it a
name
               var sheet = book.Sheets[0];

               // 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;

               // write content and format into some cells
               for (int i = 0; i < 30; i++)
               {
                   XLCell cell = sheet[i, 0];
                   cell.Value = i + 1;
                   cell.Style = ((i + 1) % 2 == 0) ? styleEven :
styleOdd;
               }
```

This code was taken from the **Excel Styles** sample in the **ControlExplorer** installed with this product.

Note: The **ControlExplorer** sample is located in the *C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Windows Phone* or *C:\Users\<username>\Documents\ComponentOne Samples\Studio for Windows Phone* folder.