ComponentOne

Zip 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

Zip for UWP

Zip for UWP provides a complete implementation of the Zip compression standard. Improve the performance of your Universal Windows apps by compressing the data sent over the wire and securing it with data encryption. The C1Zip library is the only complete, cross-platform Zip implementation.

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

Some of the main features of **Zip for UWP** are the following:

- **Get and Set File Information**

  Easily obtain zip file information, including a detailed list of the zip file's contents. You can also control file information by adding and retrieving comments and control path information for individual entries in the zip file, and getting and setting the global zip file comment. Access files within a zip archive by index or file name with ease using the C1ZipFile and C1ZipEntry classes.

- **Data Encryption**

  C1Zip supports data encryption, so you can create secure archives that require a password to unzip. With C1Zip you can also decompress archives that are encrypted so long as you have the original password used when the zip was created.

- **Perfect for XML Data**

  XML is one of the most common formats used by Web applications. XML data typically compresses to about 10% of the original size.

- **Cross-Platform Compatibility**

  The C1Zip library is available for every .NET platform including Windows Forms, ASP.NET, WPF, Silverlight, Windows Phone, WinRT and UWP. Re-use much of the same code and support more than one platform. The Microsoft ZipArchive and Compression classes are only available in .NET 4.5 on Windows 10.

- **Compress Multiple Files and Folders**

  Compress multiple folders together using **Zip for UWP** and extract the files from the zipped folder while preserving its folder structure.

Zip for UWP Fundamentals

The **C1.UWP.Zip.dll** contains classes that provide data compression services. In Universal Windows applications, data compression is especially useful when transferring between client and server. These classes are divided into three levels:

Level	Main Classes	Description
High	C1ZipFile, C1ZipEntry, C1ZipEntryCollection	Use these classes to create, open, and manage ZIP files. You can inspect the contents of ZIP files, test their integrity, add, delete, and extract entries to and from ZIP files.
Medium	C1ZStreamReader, C1ZStreamWriter	Use these classes to compress and expand data into and out of regular .NET streams (including memory, file, and network streams).
Low	ZStream	This is the lowest level class in **C1Zip**. It is a 100% C# implementation of Zlib, the popular data-compression library written by Jean-loup Gailly and Mark Adler. ZStream is used by the higher level classes in C1Zip.

High Level: C1ZipFile, C1ZipEntry and C1ZipEntryCollection Classes

These are the highest level classes in the C1Zip library. They allow you to create and manage zip files. Using zip files to store application data provides the following benefits:

- You can consolidate many files into one, making application deployment easier.
- You can compress the data, saving disk space and network bandwidth.
- The zip format is an open standard, supported by many popular applications.

C1ZipFile Class

The C1ZipFile class encapsulates a zip file. After you create a C1ZipFile object, you can attach it to an existing zip file or tell it to create a new empty zip file for you.

The code below creates a zip file called sources.zip and adds all files with a "cs" extension to the zip file:

C#

```
// get path for zip file and files to compress
string path = Application.ExecutablePath;
int pos = path.IndexOf(@"\bin");
path = path.Substring(0, pos + 1);
// create a zip file
C1ZipFile zip = new C1ZipFile();
zip.Create(path + "source.zip");
// add all files with extension cs to the zip file
foreach (string fileName in Directory.GetFiles(path, "*.cs"))
```

```
    zip.Entries.Add(fileName);
// show result
foreach (C1ZipEntry ze in zip.Entries)
{
    Console.WriteLine("{0} {1:#,##0} {2:#,##0}",
        ze.FileName, ze.SizeUncompressed, ze.SizeCompressed);
}
```

C1ZipEntryCollection Class

After you have created or opened a zip file, use the C1ZipFile.Entries collection to inspect the contents of the zip file, or to add, expand, and delete entries. For example:

Visual Basic

```
myZip.Entries.Add(stream1, "MyData.txt")
myZip.Entries.Add(stream2, "MyData.xml")
Dim zipEntry As C1ZipEntry
For Each zipEntry In myZip.Entries
    Console.WriteLine(zipEntry.FileName)
Next zipEntry
```

C#

```
myZip.Entries.Add(stream1, "MyData.txt");
myZip.Entries.Add(stream2, "MyData.doc");
foreach (C1ZipEntry zipEntry in myZip.Entries)
    Debug.WriteLine(zipEntry.FileName);
```

C1ZipEntry Class

The C1ZipEntry class exposes properties and methods that describe each entry, including its original file name, size, compressed size, and so on. It also has a C1ZipEntry.OpenReader method that returns a stream object, so you can read the entry contents without expanding it first.

Medium Level: C1ZStreamReader and C1ZStreamWriter Classes

The C1ZStreamReader and C1ZStreamWriter classes allow you to use data compression on any .NET streams, not only in zip files.

To use C1ZStreamReader and C1ZStreamWriter objects, attach them to regular streams and read or write the data through them. The data is compressed (or expanded) on the fly into (or out of) the underlying stream.
This design allows great integration with native .NET streams. The diagram below illustrates how it works:

For example, the code below saves a DataGrid into a stream and then reads it back:

C#

```
PersonList personList = new PersonList();
        public MainPage()
        {
            InitializeComponent();
        }
```

```
        // create the DataGrid
        private void btnCreate_Click(object sender, RoutedEventArgs e)
        {
            for (int i = 0; i < 1000; i++)
            {
                personList.Persons.Add(new Person()
                {
                    FirstName = string.Format("First Name {0}", i),
                    LastName = string.Format("Last Name {0}", i),
                    Age = i,
                    City = string.Format("City {0}", i)
                });
            }
            this.dataGrid1.ItemsSource = personList.Persons;
        }
        // save the data to a stream
        private void btnSave_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog dlgSaveFile = new SaveFileDialog();
            if (dlgSaveFile.ShowDialog() == true)
            {
                Stream stream = dlgSaveFile.OpenFile();
                DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
                dcs.WriteObject(stream, personList);
                stream.Close();
            }


        }
       // Compress the data
        private void btnSaveCompressed_Click(object sender, RoutedEventArgs e)
        {
            var dlgSaveFile = new SaveFileDialog();
            if (dlgSaveFile.ShowDialog() == true)
            {
                Stream stream = dlgSaveFile.OpenFile();
                //Use C1ZStreamWriter to compress the stream.
                C1ZStreamWriter compressor = new C1ZStreamWriter(stream);
                DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
                dcs.WriteObject(compressor, personList);
                stream.Close();
            }
        }
 // Load the data from the stream
        private void btnLoad_Click(object sender, RoutedEventArgs e)
        {
            var dlgOpenFile = new OpenFileDialog();
            this.dataGrid1.ItemsSource = null;
```

```
            if (dlgOpenFile.ShowDialog() == true)
            {
                Stream stream = dlgOpenFile.File.OpenRead();

                DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
                PersonList pl = (PersonList)dcs.ReadObject(stream);
                stream.Close();
                this.dataGrid1.ItemsSource = pl.Persons;
            }
        }
 // Load the compressed data
        private void btnLoadCompressed_Click(object sender, RoutedEventArgs e)
        {
            var dlgOpenFile = new OpenFileDialog();
            this.dataGrid1.ItemsSource = null;
            if (dlgOpenFile.ShowDialog() == true)
            {
                Stream stream = dlgOpenFile.File.OpenRead();
                DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
                //Use C1ZStreamReader to uncompress the stream.
                C1ZStreamReader compressor = new C1ZStreamReader(stream);
                PersonList pl = (PersonList)dcs.ReadObject(compressor);
                stream.Close();
                this.dataGrid1.ItemsSource = pl.Persons;
            }
        }
```

Low Level: ZStream Class

This is the lowest-level class in the C1Zip library, and it is used extensively by the higher-level classes described above.

Most users will never use ZStream class directly since it is the most flexible, but the hardest to use component in the C1Zip library. ZStream is a C# implementation of the ZLIB library. ZLIB is an open-source library written by Jean-loup Gailly and Mark Adler.

For more information on ZLIB, check http://www.info-zip.org/ or http://www.gzip.org/.

Zip for UWP Frequently Asked Questions

Here are some frequently asked questions (FAQs) about **Zip for UWP**:

How much data can be stored in a ZIP file comment? I would like to use it to store information in XML.

They are limited to 32k. However, if you are going anywhere near that value, it would be better to add that information as a separate file instead.

What is the maximum number of files that can be stored in a ZIP file?

Same limit, 32k entries. The total length of the zip file is limited to 4 gigs. All these limits are related to the types of variables used in the zip file specification. Interestingly, there's a proposed spec for 64-bit extensions to the zip format, but that's not widely used yet, and there is a lot of debate still going on.

Zip for UWP Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio environment.

Each task-based help topic provides a solution for specific tasks referencing the C1.C1Zip namespace. Each topic also assumes that you have created a new Universal Windows project.

Zipping Multiple Files and Folders

Multiple files and folders can be zipped or compressed easily by using AddFolderAsync method in C1ZipEntryCollection class. Use the following code to select multiple files and folders and then compress all of them together. In this code example, the following button click events are defined:

- **_btnPickFolder_Click** to select multiple folders to be compressed.
- **_btnPickFiles_Click** to select files to be compressed.
- **_btnCompress_Click** to initiate compressing of selected files and folders into one folder.
- **_btnExtract_Click** to extract zipped files and folders.

Complete the following steps:

1. Open the **MainPage.xaml** file and locate the opening <Page> tag. This tag will include the necessary namespaces. Edit the tag so that it resembles the following markup:

XAML

```
<Page
    x:Class="ZipUWP.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:ZipUWP"
    xmlns:flexGrid="using:C1.Xaml.FlexGrid"
    xmlns:c1="using:C1.Xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
```

2. Add the following XAML markup after the <Page></Page> tags. This adds buttons and other UI elements to the page.

XAML

```
    <Page.Resources>
        <ResourceDictionary>
            <ResourceDictionary.ThemeDictionaries>
                <ResourceDictionary x:Key="Default">
                    <SolidColorBrush x:Key="ApplicationPageBackgroundThemeBrush"
Color="#FF1D1D1D"/>
                </ResourceDictionary>
                <ResourceDictionary x:Key="Light">
                    <SolidColorBrush x:Key="ApplicationPageBackgroundThemeBrush"
Color="#FFF2F2F2"/>
                </ResourceDictionary>
            </ResourceDictionary.ThemeDictionaries>

            <Style x:Key="bigButton" TargetType="Button" >
```

```
                <Setter Property="Width" Value="124" />
                <Setter Property="FontSize"  Value="16" />
                <Setter Property="Margin" Value="10,0,0,0" />
                <Setter Property="Padding" Value="2" />
            </Style>
            <Style x:Key="smallButton" TargetType="Button" >
                <Setter Property="Width" Value="100" />
                <Setter Property="FontSize" Value="14" />
                <Setter Property="Margin" Value="4,0,0,0" />
                <Setter Property="Padding" Value="0" />
            </Style>

        </ResourceDictionary>
    </Page.Resources>
    <Grid>
        <Grid x:Name="_mainpage">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <RelativePanel Grid.Row="0">
                <RelativePanel Name="_compressPanel" >
                    <TextBlock Name="_compress" Text="compress" FontSize="16"
Width="100"/>
                    <Button Name="_btnPickSingleFolder"
Content="PickSingleFolder" Click="_btnPickFolder_Click"/>
                    <Button Name="_btnPickFiles" Content="PickFiles"
RelativePanel.RightOf="_btnPickSingleFolder"
RelativePanel.AlignTopWith="_btnPickSingleFolder" Click="_btnPickFiles_Click"/>
                    <Button Name="_btnCompress" Content="Compress"
RelativePanel.RightOf="_btnPickFiles"
RelativePanel.AlignTopWith="_btnPickSingleFolder" Click="_btnCompress_Click"
IsEnabled="False" Height="51" Margin="4,0,-36,-47" Width="136"/>
                </RelativePanel>

                <RelativePanel Name="_extractPanel"  >
                    <TextBlock Name="_extract" Text="extract" FontSize="16"
Width="100"/>
                    <Button Name="_btnOpen" Content="Open"
Click="_btnOpen_Click" />
                    <Button Name="_btnExtract" Content="Extract"
RelativePanel.RightOf="_btnOpen" RelativePanel.AlignTopWith="_btnOpen"
Click="_btnExtract_Click" IsEnabled="False"/>
                </RelativePanel>

                <RelativePanel Name="_actionPanel" >
                    <Button Name="_btnRemove" Content="Remove"
Click="_btnRemove_Click" IsEnabled="False" />
                    <Button Name="_btnView" Content="View"
Click="_btnView_Click" IsEnabled="False"/>
```

```
                    <Button Name="_btnClear" Content="Clear"
Click="_btnClear_Click" />

                    <ListBox Name="listBox1" />
                </RelativePanel>
            </RelativePanel>
            <ProgressBar x:Name="progressBar" Grid.Row="1" Width="400"
Canvas.ZIndex="10" HorizontalAlignment="Center" VerticalAlignment="Center"
IsIndeterminate="True" Visibility="Collapsed" Background="Black"/>
            <flexGrid:C1FlexGrid BorderBrush="Gray" BorderThickness="1"
Margin="4"
                                 x:Name="_flex" Grid.Row="2"
                                 SelectionMode="Row"
                                 HeadersVisibility="Column"
                                 AllowResizing="Columns"
                                 HeaderGridLinesBrush="Transparent"
                                 AutoGenerateColumns="False" >
                <flexGrid:C1FlexGrid.Columns>
                    <flexGrid:Column Binding="{Binding FileName}" Header="Name"
/> <!--x:Uid="/ZipSamplesLib/Resources/Name"-->
                    <flexGrid:Column Binding="{Binding SizeUncompressedLong}"
Header="Size" Format="n0" /> <!--x:Uid="/ZipSamplesLib/Resources/Size"-->
                    <flexGrid:Column Binding="{Binding SizeCompressedLong}"
Header="Compressed" Format="n0" /> <!--
x:Uid="/ZipSamplesLib/Resources/Compressed"-->
                    <flexGrid:Column Binding="{Binding CompressionRatio}"
Header="Ratio" Format="p0" /> <!--x:Uid="/ZipSamplesLib/Resources/Ratio"-->
                    <flexGrid:Column Binding="{Binding CRC32}" Header="Crc"
Format="x0" /> <!--x:Uid="/ZipSamplesLib/Resources/CRC"-->
                    <flexGrid:Column Binding="{Binding Attributes}"
Header="Attributes" /> <!--x:Uid="/ZipSamplesLib/Resources/Attributes"-->
                </flexGrid:C1FlexGrid.Columns>
            </flexGrid:C1FlexGrid>
        </Grid>
        <RelativePanel Name="_preview" Visibility="Collapsed">
            <TextBlock Name="_text" Text="text" FontSize="28" Margin="0 0 40
0"/>
            <Button  Name="_btnClosePreview" Content="ClosePreview"
Click="_btnClosePreview_Click_1" RelativePanel.RightOf="_text"
Margin="30,0,40,0" Width="100"/>
            <TextBox Name="_tbContent" IsReadOnly="True" AcceptsReturn="True"
FontFamily="Courier New" Background="White"
RelativePanel.Below="_btnClosePreview" Margin="10"/>
        </RelativePanel>

        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="WindowSizeStates">
                <!-- First set of VisualStates are simple responsive states
based on window size. AdaptiveTrigger is a built-in trigger that XAML knows to
automatically trigger based on window size -->
                <VisualState>
```

```
                    <VisualState.StateTriggers>
                        <!-- Trigger below indicates that this VisualState is to
be applied when the current window width is >= 481 effective pixels -->
                        <AdaptiveTrigger MinWindowWidth="481" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <!-- Simplified Setter statements are used to move
elements around to optimize for more available space -->
                        <Setter Target="_btnPickSingleFolder.
(RelativePanel.RightOf)" Value="_compress" />
                        <Setter Target="_btnPickSingleFolder.Style" Value="
{StaticResource bigButton}" />
                        <Setter Target="_btnPickFiles.(RelativePanel.RightOf)"
Value="_btnPickSingleFolder" />
                        <Setter Target="_btnPickFiles.Style" Value="
{StaticResource bigButton}" />
                        <Setter Target="_btnCompress.(RelativePanel.RightOf)"
Value="_btnPickFiles" />
                        <Setter Target="_btnCompress.Style" Value="
{StaticResource bigButton}" />

                        <Setter Target="_extractPanel.(RelativePanel.Below)"
Value="_compressPanel" />
                        <Setter Target="_extractPanel.Margin" Value="0,10,0,0"
/>
                        <Setter Target="_btnOpen.(RelativePanel.RightOf)"
Value="_extract" />
                        <Setter Target="_btnOpen.Style" Value="{StaticResource
bigButton}" />
                        <Setter Target="_btnExtract.Style" Value="
{StaticResource bigButton}" />

                        <Setter Target="_actionPanel.(RelativePanel.Below)"
Value="_extractPanel" />
                        <Setter Target="_actionPanel.Margin" Value="100,10,0,0"
/>
                        <Setter Target="_btnRemove.Style" Value="{StaticResource
bigButton}" />
                        <Setter Target="_btnView.(RelativePanel.RightOf)"
Value="_btnRemove" />
                        <Setter Target="_btnView.Style" Value="{StaticResource
bigButton}" />
                        <Setter Target="_btnClear.(RelativePanel.RightOf)"
Value="_btnView" />
                        <Setter Target="_btnClear.Style" Value="{StaticResource
bigButton}" />

                        <Setter Target="_flex.FontSize" Value="16" />

                    </VisualState.Setters>
                </VisualState>
```

```
                <VisualState>
                    <VisualState.StateTriggers>
                        <!-- Trigger below indicates that this VisualState is to
be applied when current window width is >=0 and <481 effective pixels -->
                        <AdaptiveTrigger MinWindowWidth="0" />
                    </VisualState.StateTriggers>
                    <VisualState.Setters>
                        <!-- Simplified Setter statements are used to move
elements around to optimize for lesser available space -->
                        <Setter Target="_compress.FontSize" Value="12" />
                        <Setter Target="_extract.FontSize" Value="12" />
                        <Setter Target="_btnPickSingleFolder.
(RelativePanel.Below)" Value="_compress" />
                        <Setter Target="_btnPickSingleFolder.Style" Value="
{StaticResource smallButton}" />
                        <Setter Target="_btnPickFiles.(RelativePanel.RightOf)"
Value="_btnPickSingleFolder" />
                        <Setter Target="_btnPickFiles.Style" Value="
{StaticResource smallButton}" />
                        <Setter Target="_btnCompress.(RelativePanel.RightOf)"
Value="_btnPickFiles" />
                        <Setter Target="_btnCompress.Style" Value="
{StaticResource smallButton}" />

                        <Setter Target="_extractPanel.(RelativePanel.Below)"
Value="_compressPanel" />
                        <Setter Target="_extractPanel.Margin" Value="0,10,0,0"
/>
                        <Setter Target="_btnOpen.(RelativePanel.Below)"
Value="_extract" />
                        <Setter Target="_btnOpen.Style" Value="{StaticResource
smallButton}" />
                        <Setter Target="_btnExtract.Style" Value="
{StaticResource smallButton}" />

                        <Setter Target="_actionPanel.(RelativePanel.Below)"
Value="_extractPanel" />
                        <Setter Target="_actionPanel.Margin" Value="0,10,0,0" />
                        <Setter Target="_btnRemove.Style" Value="{StaticResource
smallButton}" />
                        <Setter Target="_btnView.(RelativePanel.RightOf)"
Value="_btnRemove" />
                        <Setter Target="_btnView.Style" Value="{StaticResource
smallButton}" />
                        <Setter Target="_btnClear.(RelativePanel.RightOf)"
Value="_btnView" />
                        <Setter Target="_btnClear.Style" Value="{StaticResource
smallButton}" />

                        <Setter Target="_flex.FontSize" Value="10" />
```

```
                        <Setter Target="_btnClosePreview.Margin" Value="0,0,5,0"
/>
                        <Setter Target="_btnClosePreview.Width" Value="80" />

                    </VisualState.Setters>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
    </Grid>
</Page>
```

3. Open the **MainPage.xaml.cs** file. Add the following namespaces at the top of the code.

C#

```
using System.IO;
using C1.C1Zip;
using Windows.Storage.Pickers;
using Windows.Storage;
using Windows.UI.Popups;
```

4. Add the following code in the file. This code selects multiple files and folders, compress them into one zipped file, and finally extract them.

C#

```
namespace ZipUWP
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a
Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        C1ZipFile _zip;
        CollectionViewSource _cvs = new CollectionViewSource();
        MemoryStream zipMemoryStream = null;
        public MainPage()
        {
            this.InitializeComponent();
            _flex.SelectedItemChanged += _flex_SelectedItemChanged;
            // bind grid to entries collection
        }
        void _flex_SelectedItemChanged(object sender, EventArgs e)
        {
            if (_flex.SelectedItem != null)
            {
                _btnView.IsEnabled = true;
                _btnRemove.IsEnabled = true;
            }
            else
            {
                _btnView.IsEnabled = false;
                _btnRemove.IsEnabled = false;
            }
        }
```

```
        void RefreshView()
        {
            //var sel = _flex.SelectedItem;
            _flex.ItemsSource = null;
            if (_zip == null)
            {
                return;
            }
            _flex.ItemsSource = _zip.Entries;
            if (_zip.Entries.Count == 0)
            {
                _btnCompress.IsEnabled = false;
                _btnRemove.IsEnabled = false;
                _btnView.IsEnabled = false;
                _btnExtract.IsEnabled = false;
            }
            //_flex.SelectedItem = sel;
        }
        // open an existing zip file
        async void _btnOpen_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                var picker = new Windows.Storage.Pickers.FileOpenPicker();
                picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
                picker.FileTypeFilter.Add(".zip");
                StorageFile _zipfile = await picker.PickSingleFileAsync();
                if (_zipfile != null)
                {
                    Clear();
                    progressBar.Visibility = Visibility.Visible;
                    if (_zip == null)
                    {
                        _zip = new C1ZipFile(new System.IO.MemoryStream(),
true);
                    }
                    var stream = await
_zipfile.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);
                    _zip.Open(stream.AsStream());

                    _btnExtract.IsEnabled = true;
                    RefreshView();
                }
            }
            catch (Exception x)
            {
                System.Diagnostics.Debug.WriteLine(x.Message);
            }
            progressBar.Visibility = Visibility.Collapsed;
        }
```

```
        // remove selected entries from zip
        void _btnRemove_Click(object sender, RoutedEventArgs e)
        {
            foreach (C1ZipEntry entry in _flex.SelectedItems)
            {
                _zip.Entries.Remove(entry.FileName);
            }
            RefreshView();
        }
        // show a preview of the selected entry
        void _btnView_Click(object sender, RoutedEventArgs e)
        {
            var entry = _flex.SelectedItem as C1ZipEntry;
            if (entry != null)
            {
                using (var stream = entry.OpenReader())
                {
                    var sr = new System.IO.StreamReader(stream);
                    _tbContent.Text = sr.ReadToEnd();
                }
                _preview.Visibility = Visibility.Visible;
                _mainpage.Visibility = Visibility.Collapsed;
            }
        }

        // close the preview pane by hiding it
        void _btnClosePreview_Click_1(object sender, RoutedEventArgs e)
        {
            _preview.Visibility = Visibility.Collapsed;
            _mainpage.Visibility = Visibility.Visible;
        }

        // add files by folder
        private async void _btnPickFolder_Click(object sender, RoutedEventArgs
e)
        {
            try
            {
                FolderPicker folderPicker = new FolderPicker();
                folderPicker.FileTypeFilter.Add("*");
                StorageFolder pickedFolder = await
folderPicker.PickSingleFolderAsync();
                if (pickedFolder != null)
                {

                    if (_btnExtract.IsEnabled)
                    {
                        Clear();
                    }
                    progressBar.Visibility = Visibility.Visible;
                    if (zipMemoryStream == null)
```

```
                    {
                        zipMemoryStream = new MemoryStream();
                    }
                    if (_zip == null)
                    {
                        _zip = new C1ZipFile(zipMemoryStream, true);
                    }
                    await _zip.Entries.AddFolderAsync(pickedFolder);

                    _btnCompress.IsEnabled = true;
                }
            }
            catch
            {
            }
            RefreshView();
            progressBar.Visibility = Visibility.Collapsed;
        }

        // add files
        private async void _btnPickFiles_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                var picker = new Windows.Storage.Pickers.FileOpenPicker();
                picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
                picker.FileTypeFilter.Add("*");
                var files = await picker.PickMultipleFilesAsync();
                if (files != null)
                {
                    if (files.Count == 0)
                    {
                        return;
                    }
                    if (_btnExtract.IsEnabled)
                    {
                        Clear();
                    }
                    progressBar.Visibility = Visibility.Visible;
                    if (zipMemoryStream == null)
                    {
                        zipMemoryStream = new MemoryStream();
                    }
                    if (_zip == null)
                    {
                        _zip = new C1ZipFile(zipMemoryStream, true);
                    }
                    foreach (var f in files)
                    {
                        await _zip.Entries.AddAsync(f);
```

```
                    }
                    _btnCompress.IsEnabled = true;
                }
            }
            catch
            {
            }
            RefreshView();
            progressBar.Visibility = Visibility.Collapsed;
        }
        private async void _btnCompress_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                if (zipMemoryStream != null)
                {
                    FileSavePicker fileSavePicker = new FileSavePicker();
                    fileSavePicker.FileTypeChoices.Add("Zip", new List<string> {
".zip" });
                    fileSavePicker.DefaultFileExtension = ".zip";
                    fileSavePicker.SuggestedFileName = "NewFolder";
                    fileSavePicker.CommitButtonText = "Save";
                    fileSavePicker.SuggestedStartLocation =
PickerLocationId.ComputerFolder;
                    StorageFile pickedSaveFile = await
fileSavePicker.PickSaveFileAsync();
                    await FileIO.WriteBytesAsync(pickedSaveFile,
zipMemoryStream.ToArray());
                    MessageDialog md = new MessageDialog("Compress Successfully,
the path is :" + pickedSaveFile.Path);
                    md.ShowAsync();
                }
            }
            catch
            {
            }
            RefreshView();
        }
        private async void _btnExtract_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                FolderPicker folderPicker = new FolderPicker();
                folderPicker.FileTypeFilter.Add("*");
                StorageFolder pickedFolder = await
folderPicker.PickSingleFolderAsync();
                progressBar.Visibility = Visibility.Visible;
                foreach (var entry in _zip.Entries)
                {
                    var name = entry.FileName;
                    await entry.Extract(pickedFolder, name);
```

```
                }
                MessageDialog md = new MessageDialog("Extract Successfully, the
path is :" + pickedFolder.Path);
                md.ShowAsync();
            }
            catch
            {
            }
            RefreshView();
            progressBar.Visibility = Visibility.Collapsed;
        }
        private void _btnClear_Click(object sender, RoutedEventArgs e)
        {
            Clear();
        }
        private void Clear()
        {
            _flex.ItemsSource = null;
            if (zipMemoryStream != null)
            {
                zipMemoryStream.Flush();
                zipMemoryStream.Dispose();
            }
            zipMemoryStream = null;
            _btnCompress.IsEnabled = false;
            _btnExtract.IsEnabled = false;
            if (_zip != null)
            {
                _zip.Close();
                _zip = null;
            }
        }

    }
}
```

Note that the above code has limitation that it can not add empty folder into zip file.

Extracting Files from Zip Entry to Memory

To extract a file from a zip to memory variable (for instance, a Byte array), use the following function. First make sure to add these Imports (Visual Basic)/using (C#) statements at the top of the code:

Visual Basic

```
Imports C1.C1Zip and Imports System.IO
```

C#

```
using C1.C1Zip; and using System.IO
```

Then add the following code:

Visual Basic

```
Private Function GetDataFromZipFile(zipFileName As String, entryName As String) As
Byte()
    ' Get the entry from the zip file.
    Dim zip As New C1ZipFile()
    zip.Open(zipFileName)
    Dim ze As C1ZipEntry = zip.Entries(entryName)
    ' Copy the entry data into a memory stream.
    Dim ms As New MemoryStream()
    Dim buf(1000) As Byte
    Dim s As Stream = ze.OpenReader()
    Try
        While True
            Dim read As Integer = s.Read(buf, 0, buf.Length)
            If read = 0 Then
                Exit While
            End If
            ms.Write(buf, 0, read)
        End While
    Finally
        s.Dispose()
    End Try
    s.Close()
    ' Return result.
    Return ms.ToArray()
End Function
```

C#

```
private byte[] GetDataFromZipFile(string zipFileName, string entryName)
{
    // Get the entry from the zip file.
    C1ZipFile zip = new C1ZipFile();
    zip.Open(zipFileName);
    C1ZipEntry ze = zip.Entries[entryName];
    // Copy the entry data into a memory stream.
    MemoryStream ms = new MemoryStream();
    byte[] buf = new byte[1000];
    using (Stream s = ze.OpenReader())
    {
        for (;;)
        {
            int read = s.Read(buf, 0, buf.Length);
            if (read == 0) break;
            ms.Write(buf, 0, read);
        }
    }
    // There's no need to call close because of the C# 'using'
    // statement above but in VB this would be necessary.
    //s.Close();
```

```
    // Return result.
    return ms.ToArray();
}
```

Reading a Zipped File Using a StreamReader

To read a zipped file using a StreamReader, add the following code. First make sure to add these Imports (Visual Basic)/using (C#) statements at the top of the code:

Visual Basic

```
Imports C1.C1Zip and Imports System.IO
```

C#

```
using C1.C1Zip; and using System.IO
```

Then add the following code:

Visual Basic

```
' Open a zip file.
Dim zip As New C1ZipFile()
zip.Open("c:\temp\myzipfile.zip")
' Open an input stream on any entry.
Dim ze As C1ZipEntry = zip.Entries("someFile.cs")
Dim s As Stream = ze.OpenReader()
' Open the StreamReader on the stream.
Dim sr As New StreamReader(s)
' Use the StreamReader, then close it.
```

C#

```
// Open a zip file.
C1ZipFile zip = new C1ZipFile();
zip.Open(@"c:\temp\myzipfile.zip");
// Open an input stream on any entry.
C1ZipEntry ze = zip.Entries["someFile.cs"];
Stream s = ze.OpenReader();
// Open the StreamReader on the stream.
StreamReader sr = new StreamReader(s);
// Use the StreamReader, then close it.
```

Retrieving Images from a Zip File

In this example, two buttons and a listbox will be used to show how images can be retrieved from a .zip file.

To retrieve images directly from a zip file, first add the following code to compress several image files into a zip file. In this example, the code is added to the **btnNew_Click** event, which creates a new .zip file for the images when a button is clicked.

C#

```
private void btnNew_Click(object sender, RoutedEventArgs e)
        {
            // Show open file dialog.
            SaveFileDialog dlgSaveFile = new SaveFileDialog();
            dlgSaveFile.Filter = "Zip Files (*.zip) | *.zip";
            // Open zip file.
            try
            {
                if (dlgSaveFile.ShowDialog() == true)
                {
                    zipFile.Create(dlgSaveFile.OpenFile());
                }
            }
            catch
            {
                MessageBox.Show("Can't create a ZIP file, please try again.",
"C1Zip", MessageBoxButton.OK);
            }
        }
```

Then use the following code to add image files to a list in the ListBox.

C#

```
 private void btnAdd_Click(object sender, RoutedEventArgs e)
        {
            // Get list of files to add.
            OpenFileDialog fo = new OpenFileDialog();
            fo.Multiselect = true;
            if (fo.ShowDialog() == true)
            {
                // Add files in the list.
                foreach (FileInfo file in fo.Files)
                {
                    Stream stream = file.OpenRead();
                    listBox1.Items.Add(file.Name);
                    zipFile.Entries.Add(stream, file.Name);
                }
            }
        }
```

To allow you to select an image, retrieve a stream with the image data (C1ZipEntry.OpenReader method), and add the following code to the **listBox1_SelectionChanged** and **StreamCopy** events:

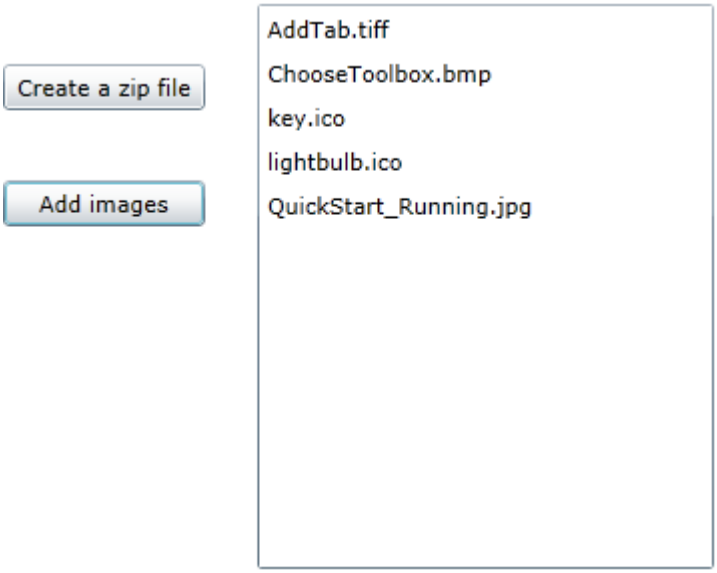
C#

```
private void listBox1_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            // Get the selected item.
            string item = (string)listBox1.SelectedItem;
            // Load the image directly from a compressed stream.
            Stream stream = zipFile.Entries[item].OpenReader();
            using (MemoryStream ms = new MemoryStream())
```

```
        {
            StreamCopy(ms, stream);
            BitmapImage img = new BitmapImage();
            img.SetSource(ms);
            this.image1.Source = img;
            // Done with stream.
            stream.Close();
        }
    }
    private void StreamCopy(Stream dstStream, Stream srcStream)
    {
        byte[] buffer = new byte[32768];
        for (; ; )
        {
            int read = srcStream.Read(buffer, 0, buffer.Length);
            if (read == 0) break;
            dstStream.Write(buffer, 0, read);
        }
        dstStream.Flush();
    }
```

This topic illustrates the following:

This example shows several types of images, including ICO, TIFF, BMP, and JPG images.



Saving a String Variable to a Zip File

To save a string variable to a zip file, use one of the following methods:

- C1ZipEntryCollection.OpenWriter method

  Use the C1ZipEntryCollection.OpenWriter method to get a stream writer, write the string into it, and then close it. The data is compressed as you write it into the stream, and the whole stream is saved into the zip file when you close it.

- **MemoryStream** method

  Use a MemoryStream method; write the data into it, and then add the stream to the zip file. Note that this method requires a little more work than the C1ZipEntryCollection.OpenWriter method, but is still very manageable.

The following code shows both methods. In this example, the code for the OpenWriter method is shown in the **button1_Click** event. The code for the MemoryStream method is shown in the **button2_Click** event.

C#

```
private void button1_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog dlgSaveFile = new SaveFileDialog();
            dlgSaveFile.Filter = "Zip Files (*.zip) | *.zip";
            if (dlgSaveFile.ShowDialog() == true)
            {
                zipFile.Create(dlgSaveFile.OpenFile());
            }
            // Method 1: OpenWriter.
            Stream stream = zipFile.Entries.OpenWriter("Shakespeare.txt", true);
            C1ZStreamWriter sw = new C1ZStreamWriter(stream);
            byte[] text = System.Text.Encoding.Unicode.GetBytes(shakespeareText);
            sw.Write(text, 0, text.Length);
            sw.Flush();
            stream.Close();
        }
        private void button2_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog dlgSaveFile = new SaveFileDialog();
            dlgSaveFile.Filter = "Zip Files (*.zip) | *.zip";
            if (dlgSaveFile.ShowDialog() == true)
            {
                zipFile.Create(dlgSaveFile.OpenFile());
            }
            // Method 2: Memory Stream.
            Stream stream = new MemoryStream();
            C1ZStreamWriter sw = new C1ZStreamWriter(stream);
            byte[] text = System.Text.Encoding.Unicode.GetBytes(shakespeareText);
            sw.Write(text, 0, text.Length);
            sw.Flush();
            stream.Position = 0;
            zipFile.Entries.Add(stream, "Shakespeare2.txt");
            stream.Close();
        }
```

Setting the Level of Compression

To minimize the file size of the compressed file, set the compression level on the C1ZStreamWriter's constructor by using the following code:

C#

```
SaveFileDialog dlgSaveFile = new SaveFileDialog();

            if (dlgSaveFile.ShowDialog() == true)
            {
                C1ZStreamWriter compressor = new
C1ZStreamWriter(dlgSaveFile.OpenFile(),
                    CompressionLevelEnum.BestCompression);
            }
```

The code sample above sets the compression level to BestCompression, which has the highest compression time and the lowest speed.

Other compression level options on the C1ZStreamWriter's constructor include the following:

- **BestSpeed** has low compression time and the highest speed.
- **DefaultCompression** has normal compression time and speed.
- **NoCompression** has no compression.

Using Passwords to Protect Zip Files

To create password-protected zip files, set the C1ZipFile.Password property to a non-empty string before creating any entries. Each entry may have its own password. For example:

Visual Basic

```
Dim zip As New C1ZipFile()
zip.Password = "password"
zip.Entries.Add(someFile)
```

C#

```
C1ZipFile zip = new C1ZipFile();
zip.Password = "password";
zip.Entries.Add(someFile);
```

To extract this entry later, the C1ZipFile.Password property must be set to the same value in effect when the entry was added. For example:

Visual Basic

```
' Will fail, password not set.
zip.Password = ""
zip.Entries.Extract(someFile)
' Will fail, wrong password.
zip.Password = "pass"
zip.Entries.Extract(someFile)
' Will succeed.
zip.Password = "password"
zip.Entries.Extract(someFile)
```

C#

```
// Will fail, password not set.
```

```
zip.Password = "";
zip.Entries.Extract(someFile);
// Will fail, wrong password.
zip.Password = "pass";
zip.Entries.Extract(someFile);
// Will succeed.
zip.Password = "password";
zip.Entries.Extract(someFile);
```

Opening a Zip File from Embedded Resources

You can load a file from your project's embedded resources by using the following code. In this example the project namespace is "ZipFileDemo". The Sample.zip file is located in the project's root directory and has its Build property set to Embedded Resource.

Add the following imports statement to the top of your project:

C#

```
using System.Reflection;
using C1.C1Zip;
```

Add the following code to your project:

C#

```
// Open a zip file from embedded resources
C1ZipFile zip = new C1ZipFile();
Assembly asm = typeof(MainPage).GetTypeInfo().Assembly;
Stream stream = asm.GetManifestResourceStream("ZipFileDemo.Sample.zip");
zip.Open(stream);

// Load entries into FlexGrid
//_flex.ItemsSource = zip.Entries;
```

Loading a Zip File from the Web

You can reduce application size by downloading zip files asynchronously from the Web. Use the following code to open a zip file from the Web into a C1ZipFile object.

C#

```
using C1.C1Zip;
using System.Net.Http;
using Windows.UI.Popups;

private async void LoadZipFile()
{
    // load file from the Web
    HttpClient client = new HttpClient();
    C1ZipFile zip = new C1ZipFile();
    try
```

```
    {
        // Download zip into byte array.
        var byteArray = await client.GetByteArrayAsync(new Uri("htt://yourfile.zip",
UriKind.Absolute));

        // Write byte array to stream and open
        MemoryStream ms = new MemoryStream();
        ms.Write(byteArray, 0, byteArray.Length);
        zip.Open(ms);

        // Load entries into FlexGrid
        _flex.ItemsSource = zip.Entries;
    }
    catch (Exception ex)
    {
        var dialog = new MessageDialog(ex.Message);
        dialog.ShowAsync();
    }
}
```

Open a Zip File from Users Machine

You can allow the user to select and open a Zip file from their documents library using the FileOpenPicker class. Use the following code to open a file with C1Zip. This sample also assumes you have a C1FlexGrid control on your page named “flex” to display the results, however this is not necessary.

C#

```
// open an existing zip file
async void _btnOpen_Click_1(object sender, RoutedEventArgs e)
{
    C1ZipFile zip = new C1ZipFile(new System.IO.MemoryStream(), true);
    try
    {
        var picker = new Windows.Storage.Pickers.FileOpenPicker();
        picker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
        picker.FileTypeFilter.Add(".zip");

        var file = await picker.PickSingleFileAsync();
        if (file != null)
        {
            var stream = await
file.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);
            zip.Open(stream.AsStream());

            // Load entries into FlexGrid
            flex.ItemsSource = zip.Entries;
        }
    }
    catch (Exception x)
```

```
    {
        System.Diagnostics.Debug.WriteLine(x.Message);
    }
}
```

Access a Specific File within a Zip

With C1Zip you can access individual files and folders within a Zip archive using the Entries collection. The following example assumes you have already opened a zip file with an instance of C1ZipFile named "_zip".

C#

```
// Open an input stream on any entry.
C1ZipEntry ze = _zip.Entries["someFile.cs"];
Stream s = ze.OpenReader();
// Open the StreamReader on the stream.
StreamReader sr = new StreamReader(s);
// Use the StreamReader, then close it
```