ComponentOne

Zip for Windows Phone

Copyright © 1987-2012 GrapeCity, Inc. All rights reserved.

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)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are 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 CD (or disk) 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 a defective CD (or disk) by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) 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. We are 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

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

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

Table of Contents

ComponentOne Zip for Windows Phone Overview 1
- Help with ComponentOne Studio for Windows Phone 1
- ComponentOne Zip for Windows Phone Assemblies and Controls 1

Key Features 1

Zip for Windows Phone Fundamentals 2
- High Level: C1ZipFile, C1ZipEntry and C1ZipEntryCollection Classes 2
- Medium Level: C1ZStreamReader and C1ZStreamWriter Classes 3
- Low Level: ZStream Class 4

Zip for Windows Phone Frequently Asked Questions 5

Zip for Windows Phone Task-Based Help 5
- Extracting Files from Zip Entry to Memory 5
- Reading a Zipped File Using a StreamReader 6
- Retrieving Images from a Zip File 7
- Saving a String Variable to a Zip File 9
- Setting the Level of Compression 9
- Using Passwords to Protect Zip Files 10

Zip for Windows Phone Tutorials 11
- Compressing Data in Memory 11
- Compressing Files 17
- Compressed Serialization 23

ComponentOne Zip for Windows Phone Overview

ComponentOne Zip™ for Windows Phone provides a complete implementation of the Zip compression standard. Improve the performance of your Phone apps by compressing the data sent over the wire, and use isolated storage efficiently and securely with data compression and encryption.

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 Zip for Windows Phone Assemblies and Controls

Key Features

Some of the main features of **ComponentOne Zip for Windows Phone** that you may find useful include the following:

- **Standard-based**

  Load data from existing zip files, or create and edit zip files that can be read by external applications.

- **Compress Files and Streams**

  Store resources in compressed files, download them efficiently, and then load them on the client side. You can also compress streams and bypass files altogether.

- **Encrypt Data**

  Encrypt the data you save in isolated storage for additional security.

- **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.

- **Get More Mileage Out of Isolated Storage**

  Windows Phone applications don’t have access to the file system; they only have access to a limited amount of isolated storage. Use this storage efficiently by compressing the data.

Zip for Windows Phone Fundamentals

The **C1.Phone.Zip.dll** contains classes that provide data compression services. In Windows Phone applications, data compression is especially useful when transferring between client and server, and also when storing application data in isolated storage (which is limited to 1MB by default). 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#

```
IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            C1ZipFile zip = new C1.C1Zip.C1ZipFile();
            var fs = isf.CreateFile("Books\\books.zip");
            zip.Create(fs);

            // add all files with extension cs to the zip file
            foreach (string fileName in isf.GetFileNames("Books\\*.txt"))
                zip.Entries.Add(isf.OpenFile("Books\\" + fileName,
FileMode.Open), 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 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

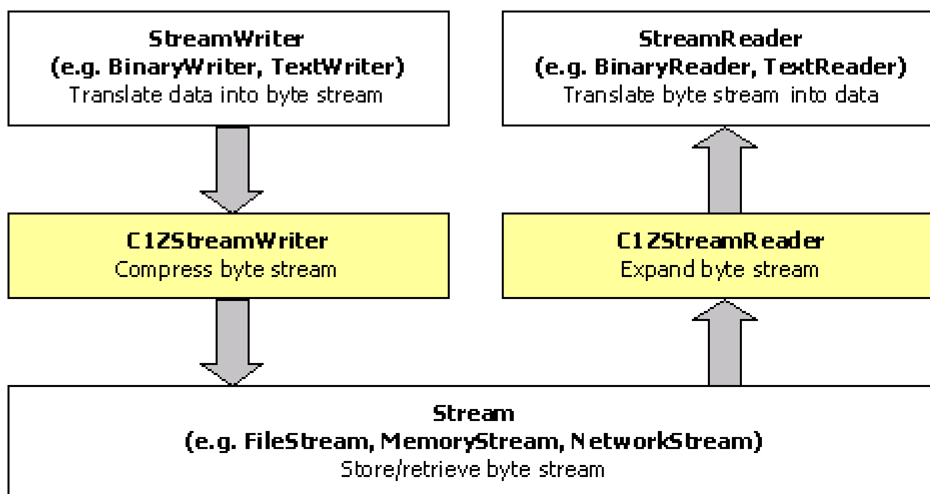
The C1ZipEntry class exposes properties and methods that describe each entry, including its original file name, size, compressed size, and so on. It also has a OpenReader method that returns a stream object, so you can read the entry contents without expanding it first.

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 PersonList object into a stream and then reads it back:

```
PersonList personList = new PersonList();
        private int idx = -1;
```

```
        // Constructor
        public MainPage()
        {
            InitializeComponent();
        }

        //Create the data
        private void button1_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)
                });
            }

            ShowData(personList.Persons[0]);
            idx = 0;
        }

        //Compress data
        private void button2_Click(object sender, RoutedEventArgs e)
        {
            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            var stream = isf.CreateFile("persons.tmp");
            C1ZStreamWriter compressor = new C1ZStreamWriter(stream);
            DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
            dcs.WriteObject(compressor, personList);
            stream.Close();
        }

        //Load the compress data
        private void button3_Click(object sender, RoutedEventArgs e)
        {
            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            var stream = isf.OpenFile("persons.tmp", FileMode.Open);
            DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
            C1ZStreamReader compressor = new C1ZStreamReader(stream);
            personList = (PersonList)dcs.ReadObject(compressor);
            stream.Close();
        }
```

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 need to use ZStream directly. 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 Windows Phone Frequently Asked Questions

Here are some frequently asked questions (FAQs) about **ComponentOne Zip for Windows Phone**:

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 Windows Phone Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio.NET environment. If you are a novice to **C1Zip**, see the Zip for Windows Phone Tutorials (page 11) first.

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 Silverlight project.

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.

Make sure to add these Imports (Visual Basic)/using (C#) statements at the top of the code:

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

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

- 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();

IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
var fs = Isf.OpenFile(zipFileName, FileMode.Open);
zip.Open(fs);
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.

Make sure to add these Imports (Visual Basic)/using (C#) statements at the top of the code:

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

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

- 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#

```
IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            C1ZipFile zip = new C1.C1Zip.C1ZipFile();
            var fs = isf.OpenFile("Books\\books.zip", FileMode.Open);
            zip.Open(fs);

            // Open an input stream on any entry.
            C1ZipEntry ze = zip.Entries["a.txt"];
            Stream s = ze.OpenReader();

            // Open the StreamReader on the stream.
            StreamReader sr = new StreamReader(s);
```

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 **MainPage** constructors, which creates a new .zip file for the images when the page is initialized.

- C#

```
public MainPage()
        {
            InitializeComponent();

            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();

            //Create Images fold.
            isf.CreateDirectory("Images");
            var fs = isf.CreateFile("Images\\images.zip");

            // Create the zip file.
            zipFile.Create(fs);

            this.listBox1.Items.Add("activex.png");
            this.listBox1.Items.Add("asp.net wijmo.png");
            this.listBox1.Items.Add("doc-to-help.png");
```

```
            this.listBox1.Items.Add("entity framework.png");
            this.listBox1.Items.Add("iphone.png");
            this.listBox1.Items.Add("lightswitch.png");
            this.listBox1.Items.Add("sharepoint.png");
            this.listBox1.Items.Add("silverlight.png");
            this.listBox1.Items.Add("windows phone.png");
            this.listBox1.Items.Add("winforms.png");
            this.listBox1.Items.Add("wpf.png");

            foreach (var item in this.listBox1.Items)
            {
                var fn = item.ToString();

                using (var s = Application.GetResourceStream(new
Uri("/RetrieveImagesFromZipFile;component/resources/" +
                    fn, System.UriKind.Relative)).Stream)
                {
                    zipFile.Entries.Add(s, fn);
                }
            }
        }
```

To allow you to select an image, retrieve a stream with the image data (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();
        }
```

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 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 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)
        {
string str = "Shall I compare thee to a summer's day? " +
        "Thou art more lovely and more temperate. " +
        "Rough winds do shake the darling buds of May, " +
        "And summer's lease hath all too short a date.";

           IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            isf.CreateDirectory("Temp");
            var fs = isf.CreateFile("Temp\\strings.zip");
            C1ZipFile zipFile = new C1ZipFile();
            zipFile.Create(fs);

            // Method 1: OpenWriter.
           Stream stream = zipFile.Entries.OpenWriter("Shakespeare.txt",
true);
            StreamWriter sw = new StreamWriter(stream);
            sw.Write(str);
            sw.Close();

            // Method 2: Memory Stream.
            stream = new MemoryStream();
            sw = new StreamWriter(stream);
            sw.Write(str);
            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#

```
IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
```

```
            var fs = isf.CreateFile("temp.txt");
            C1ZStreamWriter compressor = new C1ZStreamWriter(fs,
            CompressionLevelEnum.BestCompression)
            fs.Close();
```

Note that 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 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 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);
```

Zip for Windows Phone Tutorials

The following topics contain tutorials that illustrate some of the main features in the **C1Zip** library. The tutorials walk you through the creation of several simple projects, describing each step in detail.

Tutorial	Description
Compressing Data in Memory (page 11)	Shows how you can compress and expand arbitrary data in memory. This technique is useful in any situation where you keep memory streams while the application is running. By compressing the streams, you can reduce the memory requirements for your application.
Compressing Files (page 17)	Shows how you can compress individual files so they take up less disk space and are less exposed to access by users. Note that these are not zip files, just individual compressed files. The last tutorial covers zip files.
Compressed Serialization (page 23)	Shows how you can combine ComponentOne Zip with Silverlight serialization to save objects into streams that are a fraction of the regular size. If you are serializing objects into XML streams, the savings in disk space and network bandwidth can be huge.

Compressing Data in Memory

This tutorial shows how you can compress basic data types such as strings and doubles into memory streams, and expand the data back when you read it from the streams.

Step 1: Create the main form.

Create a new Silverlight project in Visual Studio project so it looks like this:

From the Toolbox, add the following controls to the form, by performing a drag-and-drop operation or by double-clicking the component:

- Four **Button** controls. In the Properties window make the following changes to each **Button** control:

Button	Button.Content Property	Button.Name Property	Button.IsEnabled Property
1	Compress String	**btnCompressString**	True (Default)
2	Decompress String	**btnExpandString**	**False**
3	Compress Data	**btnCompressData**	True (Default)
4	Decompress Data	**btnExpandData**	**False**

Note: The **Decompress String** and **Decompress Data** buttons cannot be used until we have some compressed data to expand.

- A **TextBox** control. Set the **Height** property to **350** and set the **TextWrapping** property to **Wrap**.
- A **TextBlock** control below the text box.

Step 2: Add a reference to the C1.Phone.Zip assembly.

Go to the Solution Explorer window and click the **Show All Files** button. Right-click on **References**, and select the **Add Reference** menu option. Select the C1.Phone.Zip assembly from the list, or browse to find the C1.Phone.Zip.dll file.

Select the **MainPage.xaml.cs** for C# (**MainPage.xaml.vb** tab for Visual Basic) or go to **View | Code** to open the Code Editor. At the top of the file, add the following statements:

- C#

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

This makes the objects defined in the C1.Phone.Zip assembly visible to the project and saves a lot of typing.

Step 3: Add code to compress strings.

Double-click the **Compress String** command button, and add the following code to handle the **btnCompressString_Click** event:

- C#

```
private byte[] _compressedString;
      private void btnCompressString_Click(object sender, RoutedEventArgs
e)
        {
            // Compress the string.
            long ticks = DateTime.Now.Ticks;
            richTextBox1.SelectAll();
            string text = richTextBox1.SelectedText;
            _compressedString = CompressString(text);

            // Tell the user how long it took.
            int ms = (int)((DateTime.Now.Ticks - ticks) /
TimeSpan.TicksPerMillisecond);
            int lenBefore = text.Length * 2;
            int lenAfter = _compressedString.Length;
            string msg = string.Format("Compressed from {0} bytes to " +
"{1} bytes in {2} milliseconds.", lenBefore, lenAfter, ms);
            MessageBox.Show(msg, "Compressed", MessageBoxButton.OK);

            // We can now expand it.
            btnExpandString.IsEnabled = true;
        }
```

This code declares a member variable called **_compressedString** which will be used to hold the compressed data (encoded as a byte array). It then calls a utility function **CompressString** that compresses a given string into a byte array that can later be expanded to restore the original string. The remainder of the code is used to measure how long the compression process took and to show a dialog box with statistics.

(Note that the **lenBefore** variable is calculated as the length of the string times two. This is because .NET strings are Unicode, and each character actually takes up two bytes.)

Add the following code which implements the **CompressString** function:

- C#

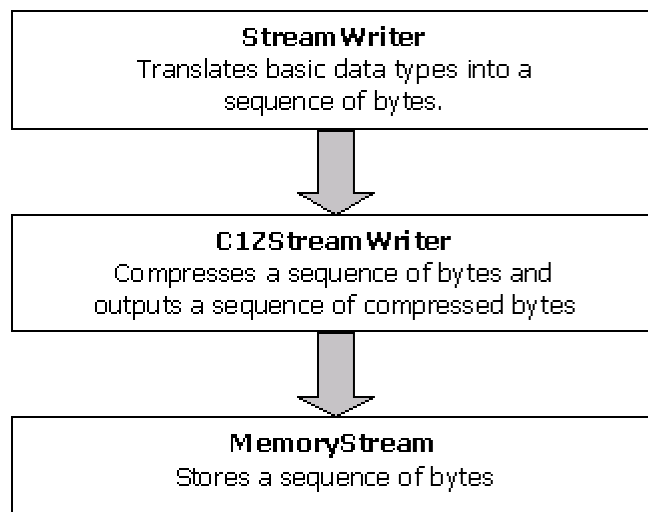
```
public byte[] CompressString(string str)
{

    // Open the memory stream.
    MemoryStream ms = new MemoryStream();

    // Attach a compressor stream to the memory stream.
    C1ZStreamWriter sw = new C1ZStreamWriter(ms);

    // Write the data into the compressor stream.
    StreamWriter writer = new StreamWriter(sw);
```

```
    writer.Write(str);

    // Flush any pending data.
    writer.Flush();

    // Return the memory buffer.
    return ms.ToArray();
}
```

The function starts by creating a new memory stream. This stream will automatically allocate a memory buffer to hold the compressed data.

Next, the function creates a C1ZStreamWriter object and attaches it to the new memory stream. Any data written to the C1ZStreamWriter object will be compressed and written to the memory stream.

The C1ZStreamWriter object only supplies the basic Stream methods for writing bytes and byte arrays. To be able to write other basic types such as strings, integers, and so on, we attach a **StreamWriter** object to the C1ZStreamWriter. Here's a diagram that shows how this works:



After the **StreamWriter** is set up, all we need to do is call its **Write** method to write the string into the compressed memory stream. When done writing, we also call the **Flush** method to make sure all cached input is written out.

Finally, the code uses the **ToArray** method to return the byte array that was created by the memory stream.

Step 4: Add code to expand strings.

To expand the string, we need to follow the reverse sequence of steps used to compress. Double-click the **Decompress String** button, and add the following code to handle the **btnExpandString_Click** event:

- C#

```
  private void btnExpandString_Click(object sender, RoutedEventArgs e)
        {
            // Expand the string.
            long ticks = DateTime.Now.Ticks;
            string text = ExpandString(_compressedString);
            TextBox1.Selection.Text = text;

            // Tell the user how long it took.
            int ms = (int)((DateTime.Now.Ticks - ticks) /
TimeSpan.TicksPerMillisecond);
            int lenBefore = _compressedString.Length;
            int lenAfter = text.Length * 2;
```

```
            string msg;
            msg = string.Format("Expanded from {0} bytes to {1} bytes " +
"in {2} milliseconds.", lenBefore, lenAfter, ms);
            MessageBox.Show(msg, "Expanded", MessageBoxButton.OK);
        }
```

The main line calls the utility function **ExpandString** that takes a byte array and returns the original string. Add the following code for the **ExpandString** function:

- C#

```
public string ExpandString(byte[] buffer)
{

    // Turn buffer into a memory stream.
    MemoryStream ms = new MemoryStream(buffer);

    // Attach a decompressor stream to the memory stream.
    C1ZStreamReader sr = new C1ZStreamReader(ms);

    // Read uncompressed data.
    StreamReader reader = new StreamReader(sr);
    return reader.ReadToEnd();
}
```

If you run the project now, you can already experiment with string compression and decompression. You can change the text in the text box, or paste new content into it, then compress and expand the string to see how much it compresses.

Step 5: Add code to compress binary data.

Compressing binary data is just as easy as compressing strings. The only difference is that instead of attaching a StreamWriter object to the compressor stream, you attach a BinaryWriter object.

Double-click the **Compress Data** button and add the following code to handle the **btnCompressData_Click** event:

- C#

```
private byte[] _compressedData;
private void btnCompressData_Click(object sender, RoutedEventArgs e)
        {
            // Open the memory stream.
            MemoryStream ms = new MemoryStream();

            // Attach a compressor stream to the memory stream.
            C1ZStreamWriter sw = new C1ZStreamWriter(ms);

            // Attach a BinaryWriter to the compressor stream.
            BinaryWriter bw = new BinaryWriter(sw);

            // Write a bunch of numbers into the stream.
            int i;
            int count = 1000;
            bw.Write(count);
            for (i = 0; i <= count - 1; i++)
            {
                double a = i * Math.PI / 180.0;
                bw.Write(i);
                bw.Write(a);
                bw.Write(Math.Sin(a));
                bw.Write(Math.Cos(a));
            }
```

```
            // Flush any pending output.
            bw.Flush();

            // Save the compressed data.
            _compressedData = ms.ToArray();

            // Done.
            string msg;
            msg = string.Format("Generated table with {0} points," +
                " saved into {1} bytes", count, _compressedData.Length);
            textBlock.Text = msg;

            // We can now expand it.
            btnExpandData.IsEnabled = true;
        }
```

The code starts by declaring a member variable called **_compressedData** which will be used to hold the compressed data (encoded as a byte array).

Then it sets up the **MemoryStream**, C1ZStreamWriter, and **BinaryWriter** objects as before (the only difference is we're now using a **BinaryWriter** instead of a **StreamWriter**).

Next, the code writes data into the stream using the **Write** method. The **BinaryWriter** object overloads this method so you can write all basic object types into streams. Finally, the **Flush** method is used as before, to make sure any cached data is written out to the compressed stream.

Step 6: Add code to expand the binary data.

Expanding the compressed binary data is just a matter of setting up the decompressor stream and reading the data like you would read it from a regular stream.

Double-click the **Decompress Data** command button, and add the following code to handle the **Click** event handler:

- C#

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

    // Open the memory stream on saved data.
    MemoryStream ms = new MemoryStream(_compressedData);

    // Attach a decompressor stream to the memory stream.
    C1ZStreamReader sr = new C1ZStreamReader(ms);

    // Read the uncompressed data.
    int i;
    BinaryReader br = new BinaryReader(sr);
    int count = br.ReadInt32();
    for (i = 0 ; i <= count - 1; i++)
    {
        int deg = br.ReadInt32();
        double rad = br.ReadDouble();
        double sin = br.ReadDouble();
        double cos = br.ReadDouble();
    }

    // Done, tell the user about it.
    string  msg;
    msg = string .Format("Read table with {0} points " +
```

```
            "from stream with {1} bytes.", count, _compressedData.Length);
    textBlock1.Text = msg;
}
```

Step 7: Run the project.

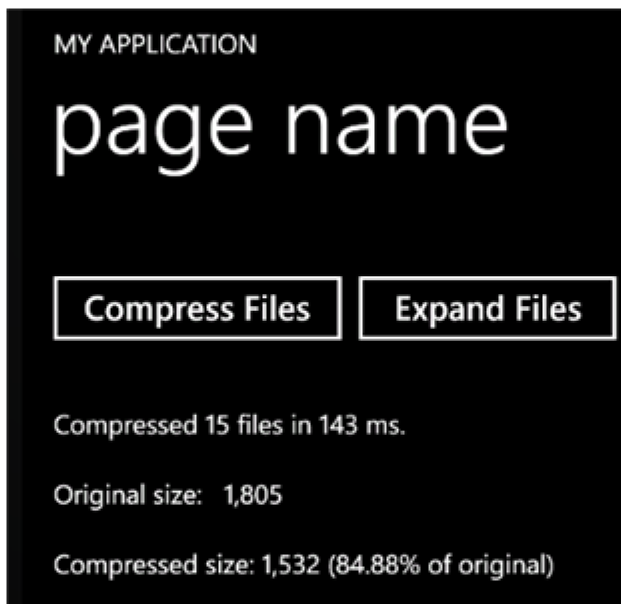
The code reads the data but does not display it. You can step through it in debug mode to make sure the data being read is the same that was written in.

If you run the project and click the compress/decompress data buttons, you will see that the data is saved in an array with 14,125 bytes. To save this data in a regular stream, it would take [4 + 1000 * (4 + 8 * 3)] = 28,004 bytes. So we compressed it to about half the original size.

This concludes the Compressing Data in Memory tutorial.

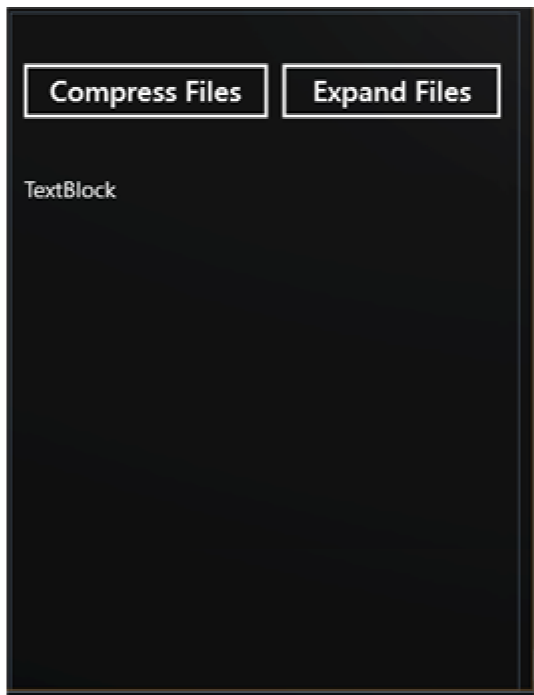
Compressing Files

This tutorial shows how you can compress and expand individual files. Note that these are not zip files; they are just compressed streams on disk. Here is what the final application will look like:



Step 1: Create the main form.

Start a new Silverlight project in Visual Studio and from the Toolbox, add the following controls to the form so it looks like this:



- Two **Button** controls. In the Properties window make the following changes:

Button	Button.Content Property	Button.Name Property
1	Compress Files	**btnCompress**
2	Expand Files	**btnExpand**

- A **TextBlock** control below the buttons. This control will display statistics about the compression/expanding process.

Step 2: Add a reference to the C1.Phone.Zip assembly.

Go to the Solution Explorer window and click the **Show All Files** button. Right-click on **References**, and select the **Add Reference** menu option. Select the C1.Phone.Zip assembly from the list, or browse to find the C1.Phone.Zip.dll file.

Select the **MainPage.xaml.vb** tab (**MainPage.xaml.cs** in C#) or go to **View | Code** to open the Code Editor. At the top of the file, add the following statements:

- Visual Basic

```
Imports System.IO
Imports System.IO.IsolatedStorage
Imports C1.C1Zip
```

- C#

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

This makes the objects defined in the **C1.Phone.Zip** and **System.IO** assemblies visible to the project and saves a lot of typing.

Step 3: Define the directory names for the compressed and expanded files.

In the Code Editor of the form, define the following constants:

- C#

```
private FileStream srcStream = null;
        private const string DIR_FILES = @"\files";
        private const string DIR_FILE = @"\file";
        private const string DIR_COMP = @"\compressed";
        private const string DIR_EXP = @"\expanded";
        private string content = "This is a just test file for compressing
files sample " +
            "which is located in the isolated storage in wp7.  It's named
file{0}";
```

These are the directory names where the compressed and expanded files will be stored (relative to the directory where the tutorial application is located on your disk).

Step 4: Add code to create the source files.

Add the following code to create the source files:

- C#

```
// Constructor
        public MainPage()
        {
            InitializeComponent();
            this.btnExpand.IsEnabled = false;
            this.CreateFiles();
        }

        private void CreateFiles()
        {
            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            isf.CreateDirectory(DIR_FILES);

            for (int idx = 0; idx < 15; idx++)
            {
                var stream = isf.CreateFile(DIR_FILES + DIR_FILE + idx +
".txt");
                StreamWriter sw = new StreamWriter(stream);
                sw.Write(string.Format(content, idx));
                sw.Close();
            }
        }
```

Step 5: Add code to compress the file.

Add the following code to handle the **Click** event for the **Compress File** command button:

- C#

```
private void btnCompress_Click(object sender, RoutedEventArgs e)
        {
            // Create compress directory.
            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();

            if (!isf.DirectoryExists(DIR_COMP))
            {
```

```
                isf.CreateDirectory(DIR_COMP);
            }

            // Prepare to collect compression statistics.
            long count = 0;
            long size = 0;
            long sizeCompressed = 0;
            long ticks = DateTime.Now.Ticks;

            // Compress all files in the application dir into the
compressed dir.
            foreach (string srcFile in isf.GetFileNames(DIR_FILES +
"\\*.txt"))
            {
                string dstFile = DIR_COMP + "\\" + srcFile + ".cmp";

                dstFile = dstFile.Replace(".txt", "");
                var srcStream = isf.OpenFile(DIR_FILES + "\\" + srcFile,
FileMode.Open);
                var dstStream = isf.CreateFile(dstFile);

                // Compress file.
                CompressFile(dstStream, srcStream);

                // Update stats.
                count++;
                size += srcStream.Length;
                sizeCompressed += dstStream.Length;

                // Always close our streams.
                if (srcStream != null) srcStream.Close();
                if (dstStream != null) dstStream.Close();
            }

            // Show stats.
            string msg = string.Format("Compressed {0} files in {1}
ms.\n\r" + "Original size:   {2:#,###}\n\r" + "Compressed size: {3:#,###}
({4:0.00}% of original)", count, (DateTime.Now.Ticks - ticks) /
TimeSpan.TicksPerMillisecond, size, sizeCompressed,
((double)sizeCompressed / size) * 100.0);
            this.textBlock1.Text = msg;

            // Now we can expand.
            btnExpand.IsEnabled = true;
        }
```

Add the following code for the **CompressFile** function:

- C#

```
private bool CompressFile(Stream dstStream, FileStream srcStream)
        {
            // Prepare to compress file.
            bool retval = true;
            try
            {
```

```
             // Open a compressor stream on the destination file.
                C1ZStreamWriter sw = new C1ZStreamWriter(dstStream);

             // Copy the source into the compressor stream.
                StreamCopy(sw, srcStream);
            }
            catch
            {
                // Exception? Tell the caller we failed.
                retval = false;
            }

            // Done.
            return false;
        }
```

The function starts by creating two new file streams: one for the source file and one for the compressed file. Then it creates a C1ZStreamWriter object and attaches it to the destination stream. Next, it calls the **StreamCopy** function to transfer data from the source file and write it into the compressor stream.

The **StreamCopy** function simply copies bytes from one stream to another. Here's the code:

- C#

```
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();
}
```

Note that the function calls the **Flush** method after it is done to ensure that any cached data is written out when the function is done copying. This is especially important when dealing with compressed streams, since they cache substantial amounts of data in order to achieve good compression rates.

Step 6: Add code to open the compressed file.

Add the following code to handle the **Click** event for the **Expand Files** command button:

- C#

```
private void btnExpand_Click(object sender, RoutedEventArgs e)
        {
            // Create a directory for expanded files.
            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();

            if (!isf.DirectoryExists(DIR_EXP))
            {
                isf.CreateDirectory(DIR_EXP);
            }

            // Prepare to collect compression statistics.
            long count = 0;
            long size = 0;
            long sizeExpanded = 0;
            long ticks = DateTime.Now.Ticks;
```

```
            // Expand all files in the "compressed" dir to the "expanded"
dir.
            foreach (string srcFile in isf.GetFileNames(DIR_COMP +
"\\*.cmp"))
            {
                // Expand file.
                string dstFile = DIR_EXP + "\\" + srcFile;
                dstFile = dstFile.Replace(".cmp", "");

                var srcStream = isf.OpenFile(DIR_COMP + "\\" + srcFile,
FileMode.Open);
                var dstStream = isf.CreateFile(dstFile);

                ExpandFile(dstStream, srcStream);

                // Update stats.
                count++;
                size += srcStream.Length;
                sizeExpanded += dstStream.Length;

                // Always close our streams.
                if (srcStream != null) srcStream.Close();
                if (dstStream != null) dstStream.Close();
            }

            // Show stats.
            string msg = string.Format("Expanded {0} files in {1} ms.\n\r"
+ "Original size:   {2:#,###}\n\r" + "Expanded size: {3:#,###} ({4:0.00} x
size of compressed)", count, (DateTime.Now.Ticks - ticks) /
TimeSpan.TicksPerMillisecond, size, sizeExpanded, sizeExpanded / size);
            textBlock1.Text = msg;
        }
```

Here's the code for the **ExpandFile** function:

- C#

```
private bool ExpandFile(Stream dstStream, Stream srcStream)
        {
            // Prepare to expand file.
            bool retval = true;
            try
            {
           // Open an expander stream on the compressed source.
                C1ZStreamReader sr = new C1ZStreamReader(srcStream);

          // Copy the expander stream into the destination file.
                StreamCopy(dstStream, sr);
            }
            catch
            {
                // Exception? Tell the caller we failed.
                retval = false;
            }

            // Done.
            return retval;
```

```
}
```

The function is similar to **CompressFile**, except it attaches a C1ZStreamReader to the source stream instead of attaching a C1ZStreamWriter to the destination stream.

This concludes the Compressing Files tutorial.

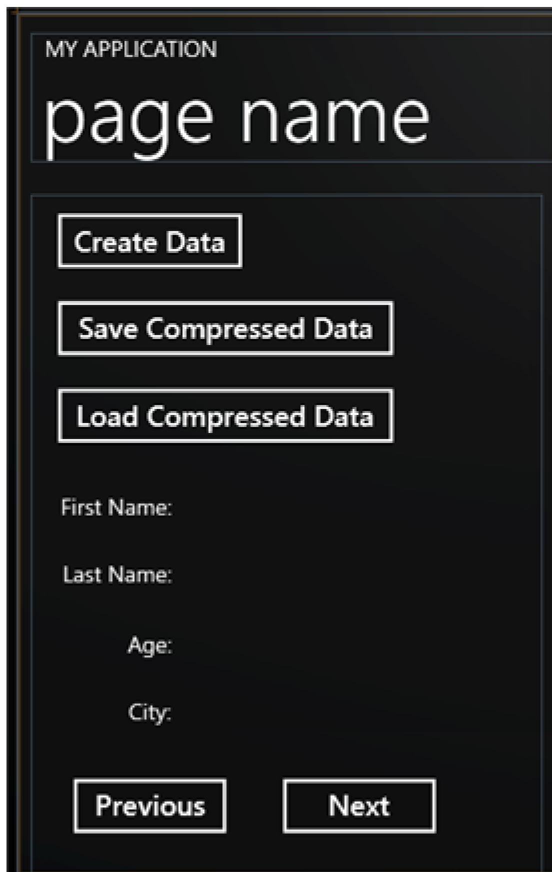
Compressed Serialization

This tutorial shows how you can serialize objects in compressed files, and then load them back into the memory.

The sample creates a data table of people. The table is saved (serialized) into compressed streams. Finally, the data is loaded back from either stream.

Step 1: Create the main form.

Start a new Silverlight for Windows Phone project in Visual Studio, and from the Toolbox, add the following controls to the form so it looks like this:



- Five **Button** controls and eight **TextBlock** controls. In the Properties window, make the following changes to each control:

Button	Button.Content Property	Button.Name Property
1	Create Data	**button1**

2	Save Compressed Data	**button2**
3	Load Compressed Data	**button3**
4	Previous	**button4**
5	Next	**button5**

TextBlock	Button.Text Property	Button.Name Property
1	First Name:	**textBlock1**
2	Last Name:	**textBlock2**
3	Age:	**textBlock3**
4	City:	**textBlock4**
5	" "	**tbFirstName**
6	" "	**tbLastName**
7	" "	**tbAge**
8	" "	**tbCity**

Step 2: Add references and Imports/using statements.

Go to the Solution Explorer window and click the **Show All Files** button. Right-click on References, and select the **Add Reference** menu option. Select the C1.Phone.Zip assembly from the list, or browse to find the C1.Phone.Zip.dll file.

Select **MainPage.xaml.vb** tab (**MainPage.xaml.cs** in C#) in the Solution Explorer or select **View | Code** to open the Code Editor. At the top of the file, add the following statements:

- Visual Basic

```
Imports System.IO
Imports System.Runtime.Serialization
Imports System.IO.IsolatedStorage
Imports C1.C1Zip
```

- C#

```
using System.IO;
using System.Runtime.Serialization;
using System.IO.IsolatedStorage;
using C1.C1Zip;
```

This declares the namespaces of the classes used in the project.

Step 3: Declare constants.

In the Code Editor of the page, type or copy the following lines in the body of the page implementation:

- C#

```
public partial class MainPage : PhoneApplicationPage
    {
        PersonList personList = new PersonList();
private int idx = -1;
```

This constant defines the list of people that will appear in the grid.

Step 4: Add code to create the data table.

1. Double-click the **Create Data** button to create the **button1_Click** handler.
2. Add the following code to handle the **Click** event:

- C#

```
//Create the data
        private void button1_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)
                });
            }

            ShowData(personList.Persons[0]);
            idx = 0;
        }
```

Step 5: Add code to save the compressed data.

1. Double-click the **Save Compressed Data** button to create the **button2_Click** handler.
2. Add the following code to handle the **Click** event for the **Save Compressed Data** button:

- C#

```
//Compress data
        private void button2_Click(object sender, RoutedEventArgs e)
        {
            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            var stream = isf.CreateFile("persons.tmp");
            C1ZStreamWriter compressor = new C1ZStreamWriter(stream);
            DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
            dcs.WriteObject(compressor, personList);
            stream.Close();
        }
```

Step 6: Add code to load the compressed data.

1. Double-click the **Load Compressed Data** button to create the **button3_Click** handler.
2. Add the following code to handle the **Click** event for the **Load Compressed Data** button:

- C#

```
//Load the compressed data
        private void button3_Click(object sender, RoutedEventArgs e)
        {
            IsolatedStorageFile isf =
IsolatedStorageFile.GetUserStoreForApplication();
            var stream = isf.OpenFile("persons.tmp", FileMode.Open);
            DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
            C1ZStreamReader compressor = new C1ZStreamReader(stream);
            personList = (PersonList)dcs.ReadObject(compressor);
            stream.Close();
        }
```

Step 7: Add code for the Previous and Next buttons.

1. Double-click the **Previous** button to create the **button4_Click** handler.
2. Add the following code to handle the **Click** event for the **button4_Click** (**Previous**) button.

- C#

```
private void button4_Click(object sender, RoutedEventArgs e)
{
    if (--idx < 0)
    {
        idx = 0;
    }

    this.ShowData(personList.Persons[idx]);
}
```

3. Double-click the **Next** button to create the **button5_Click** handler.
4. Add the following code to handle the **Click** event for the **button5_Click** (**Next**) button.

- C#

```
private void button5_Click(object sender, RoutedEventArgs e)
{
    if (++idx >= personList.Persons.Count)
    {
        idx = personList.Persons.Count - 1;
    }

    this.ShowData(personList.Persons[idx]);
}

private void ShowData(Person person)
{
    this.tbFirstName.Text = person.FirstName;
    this.tbLastName.Text = person.LastName;
    this.tbAge.Text = person.Age.ToString();
    this.tbCity.Text = person.City;
}
```

Step 8: Add a Data class to populate the TextBlocks.

In this step you will add a Data class that will add people to a list that can be viewed in the **TextBlock** controls.

1. Right-click the project in the Solution Explorer and select **Add | New Item**.
2. Select **Class**, name it **Data**, and click **Add**.
3. Add the following code to the Data.cs. Note that you must have a reference to the Sytem.Runtime.Serialization.dll in your project as well.

```
using System.Runtime.Serialization;
using System.Collections.Generic;

namespace CompressedSerialization
{

    [DataContract]
    public class Person
    {
        [DataMember]
        public string FirstName
        {
            get;
```

```
            set;
        }

        [DataMember]
        public string LastName
        {
            get;
            set;
        }

        [DataMember]
        public int Age
        {
            get;
            set;
        }

        [DataMember]
        public string City
        {
            get;
            set;
        }
    }

    [DataContract]
    public class PersonList
    {
        private List<Person> _persons = new List<Person>();

        [DataMember]
        public List<Person> Persons
        {
            get
            {
                return this._persons;
            }
            set
            {
                this._persons = value;
            }
        }
    }
}
```

Run the project and tap the **Create Data** button. Click the **Next** button to scroll through the people in the list. Now you can save and load the compressed data.

MY APPLICATION

page name

Create Data

Save Compressed Data

Load Compressed Data

First Name: First Name 2

Last Name: Last Name 2

Age: 2

City: City 2

Previous Next

This concludes the Compressed Serialization tutorial.