Zip for WPF/Silverlight

2018.04.11 更新

グレープシティ株式会社

Zip for WPF/Silverlight

目次

製品の概要

Zip for WPF/Silverlight は、Zip 圧縮標準を完全に実装しています。ネットワークを介して送信されるデータを圧縮することで、WPF/Silverlight アプリケーションのパフォーマンスを向上させることができます。また、データ圧縮や暗号化を使用して、分離ストレージを効率的かつ安全に使用できます。

> **メモ**: 説明内に含まれるクラスおよびメンバーに対するリファレンスへのリンクは、原則としてWPF版のリファレンスページを参照します。Silverlight版については、目次から同名のメンバーを参照してください。

ComponentOne for WPF/Silverlight のヘルプ

はじめに

ComponentOne for WPF/Silverlight のすべてのコンポーネントで共通の使用方法については、「ComponentOne for WPF/Silverlight ユーザーガイド」を参照してください。

主な特長

Zip for WPF/Silverlight が備える便利な機能の一部を以下に示します。

- **標準ベース**
  既存の zip ファイルからデータをロードできます。また、外部アプリケーションで読み取り可能な zip ファイルを作成および編集することができます。

- **ファイルとストリームの圧縮**
  リソースを圧縮ファイルに保存し、クライアント側でリソースを効率的にダウンロードして読み込むことができます。ストリームを圧縮したり、いくつかのファイルをまとめてバイパスすることもできます。

- **データの暗号化**
  分離ストレージ内に保存したデータを暗号化して、セキュリティを強化することができます。

- **XML データに最適**
  XML は Web アプリケーションで最もよく使用される形式の1つです。通常、XML データは元のサイズの約 10 % に圧縮されます。

- **分離ストレージの有効活用**
  WPF/Silverlight アプリケーションはファイルシステムにアクセスできません。限られた容量の分離ストレージにのみアクセスできます。データを圧縮することで、この分離ストレージを効率的に使用できます。

Zip の基本

C1.WPF.Zip.4.dllまたは**C1.Silverlight.Zip.dll** には、データ圧縮サービスを提供するクラスが含まれます。WPF/Silverlight アプリケーションでは、特にクライアントとサーバーの間でデータを転送するときや、アプリケーションデータを分離ストレージに保存するとき(デフォルトでは1 MB に制限)に、データ圧縮が役立ちます。これらのクラスは、次の3つのレベルに分類されます。

レベル	主要なクラス	説明
高	**C1ZipFile**, **C1ZipEntry**, **C1ZipEntryCollection**	このレベルのクラスを使用して、Zip ファイルを作成および管理したり、Zip ファイルを開くことができます。また、Zip ファイルの内容を調べたり、その整合性をテストすることができるほか、Zip ファイルのエントリを追加、削除、および抽出することもできます。
中	**C1ZStreamReader**, **C1ZStreamWriter**	このレベルのクラスを使用して、通常の .NET ストリーム(メモリ、ファイル、ネットワークストリームを含む)にデータを圧縮したり、.NET ストリームからデータを展開することができます。
低	**ZStream**	これは、**C1Zip** で最も低いレベルのクラスです。これは、Zlib(Jean-loup Gailly と Mark Adler による一般的なデータ圧縮ライブラリ)の 100 % C# 実装です。ZStream は、**C1Zip** のより高レベルのクラスで使用されます。

高レベル:C1ZipFile、C1ZipEntry、および C1ZipEntryCollection クラス

これらは、**C1Zip** ライブラリで最も高いレベルのクラスです。このレベルのクラスを使用して、zip ファイルを作成および管理することができます。zip ファイルを使用してアプリケーションデータを保存することには、次の利点があります。

- 複数のファイルを1つに集約することができるので、アプリケーションの展開が容易になります。
- データを圧縮することで、ディスク容量とネットワーク帯域幅を節約することができます。
- zip 形式は、一般的なアプリケーションの多くでサポートされているオープンスタンダードです。

C1ZipFile クラス

C1ZipFile クラスは、zip ファイルをカプセル化します。**C1ZipFile** オブジェクトを作成したら、そのオブジェクトを既存の zip ファイルにアタッチしたり、そのオブジェクトに基づいて空の zip ファイルを新しく作成することができます。次に例を示します。

VisualBasic

```
' C1ZipFile オブジェクトを作成します。
Dim myZip As New C1ZipFile()
' 新しい(空の)zip ファイルを作成します。
myZip.Create(writeStream)
' 既存の zip ファイルを開きます。
myZip.Open(readStream)
```

C#

```
// C1ZipFile オブジェクトを作成します。
C1ZipFile myZip = new C1ZipFile();
// 新しい(空の)zip ファイルを作成します。
myZip.Create(writeStream);
// 既存の zip ファイルを開きます。
myZip.Open(readStream);
```

C1ZipEntryCollection クラス

zip ファイルを作成または開いた後、**Entries** コレクションを使用して、zip ファイルの内容を調べたり、エントリの追加、展開、および削除を行います。次に例を示します。

VisualBasic

```
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 クラス

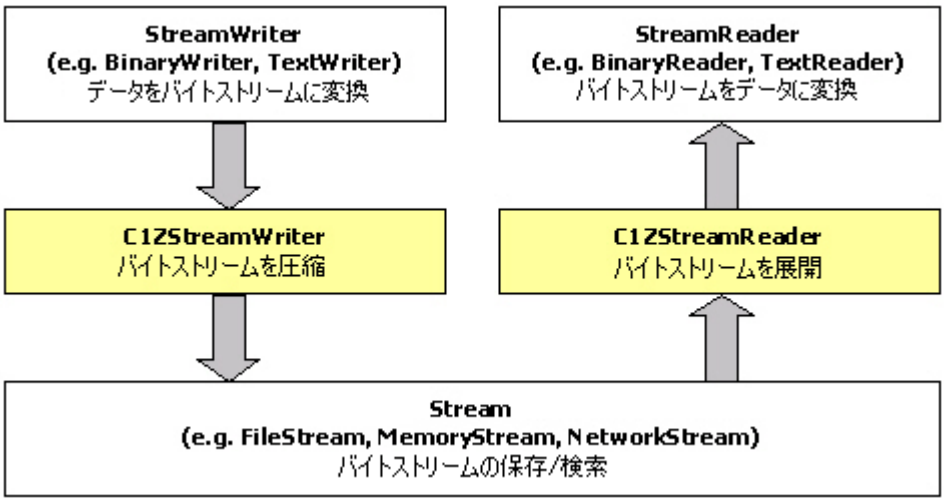
C1ZipEntry クラスは、各エントリについて説明するプロパティとメソッドを公開します。たとえば、エントリの元のファイル名、サイズ、圧縮サイズなどです。また、このクラスには、ストリームオブジェクトを返す **OpenReader** メソッドが含まれているため、エントリを展開しなくても内容を読み取ることができます。

中レベル:C1ZStreamReader および C1ZStreamWriter クラス

C1ZStreamReader クラスと **C1ZStreamWriter** クラスを使用することで、zip ファイル内だけでなく、任意の .NET ストリームでデータ圧縮を使用することができます。

C1ZStreamReader オブジェクトと **C1ZStreamWriter** オブジェクトを使用するには、それらを通常のストリームにアタッチし、そのストリームを通してデータを読み書きします。データは、基底のストリームで直ちに圧縮(または展開)されます。

この設計により、ネイティブ .NET ストリームとの統合性が高まります。次の図に、この設計がどのように機能するかを示します。



たとえば、次のコードは、ストリームに DataGrid を保存して読み戻します:

C#

```
PersonList personList = new PersonList();

        public MainPage()
        {
            InitializeComponent();
        }
        // DataGrid を作成します。
        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;
        }
        // ストリームにデータを保存します。
        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();
            }
        }
       // データを圧縮します。
```

```
        private void btnSaveCompressed_Click(object sender, RoutedEventArgs e)
        {
            var dlgSaveFile = new SaveFileDialog();

            if (dlgSaveFile.ShowDialog() == true)
            {
                Stream stream = dlgSaveFile.OpenFile();
                //C1ZStreamWriter を使用してストリームを圧縮します。
                C1ZStreamWriter compressor = new C1ZStreamWriter(stream);

                DataContractSerializer dcs = new
DataContractSerializer(typeof(PersonList));
                dcs.WriteObject(compressor, personList);
                stream.Close();
            }
        }
       // ストリームからデータを読み込みます
        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;
            }
        }
       // 圧縮されたデータを読み込みます。
        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));
                //C1ZStreamReader を使用してストリームを解凍します。
                C1ZStreamReader compressor = new C1ZStreamReader(stream);

                PersonList pl = (PersonList)dcs.ReadObject(compressor);
                stream.Close();

                this.dataGrid1.ItemsSource = pl.Persons;
            }
        }
```

低レベル:ZStream クラス

これは、**C1Zip** ライブラリ内で最も低いレベルのクラスで、前述のより高レベルのクラスで広く使用されます。

通常、直接 **ZStream** を使用する必要はありません。これは最も柔軟性の高いクラスですが、**C1Zip** ライブラリで最も使い方が難しいコンポーネントです。**ZStream** は、ZLIB ライブラリの C# 実装です。ZLIB は、Jean-loup Gailly と Mark Adler によるオープンソースライブラリです。

ZLIB の詳細については、http://www.info-zip.org/ または http://www.gzip.org/ を参照してください。

よくある質問と回答

次に、**Zip for WPF/Silverlight** に関してよくある質問（FAQ）の一部を示します。

1）zip ファイルコメントに保存可能なデータ量を教えてください。zip ファイルコメントを使用して XML に情報を保存したいのです。
データ量は 32 KB に制限されています。この上限値に近い場合は、その情報を別のファイルとして追加することをお勧めします。

2）zip ファイルには最大何個のファイルを格納できますか。
同様に、32 k のエントリに制限されます。zip ファイル全体のサイズは4 GB に制限されています。これらの制限はすべて、zip ファイル仕様で使用されている変数の型に関係しています。興味深いことに、64 ビット拡張版の zip 形式に対する提案仕様がありますが、これはまだ広く使用されておらず、いまだ多くの議論があります。

タスク別ヘルプ

タスク別ヘルプセクションは、Visual Studio.NET 環境でのプログラミングにある程度慣れていることを前提としています。**C1Zip** に精通していない場合は、まず「チュートリアル」を参照してください。

タスク別ヘルプの各トピックでは、具体的に **C1.C1Zip** 名前空間を参照するタスクを実行する方法について説明します。

Zip エントリからメモリにファイルを抽出する

zip からメモリ変数（バイト配列など）にファイルを抽出するには、次のコードを追加します。コードの先頭に、これらの Imports 文 (Visual Basic)/(C#) を追加してください：

VisualBasic

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

C#

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

VisualBasic

```
Private Function GetDataFromZipFile(zipFileName As String, entryName As String) As
Byte()
 ' zip ファイルからエントリを取得します。
 Dim zip As New C1ZipFile()
 zip.Open(zipFileName)
 Dim ze As C1ZipEntry = zip.Entries(entryName)
 ' エントリデータをメモリストリームにコピーします。
 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()
 ' 結果を返します。
 Return ms.ToArray()
```

```
End Function
```

C#

```
private byte[] GetDataFromZipFile(string zipFileName, string entryName)
 {
   // zip ファイルからエントリを取得します。
   C1ZipFile zip = new C1ZipFile();
   zip.Open(zipFileName);
   C1ZipEntry ze = zip.Entries[entryName];
   // エントリデータをメモリストリームにコピーします。
   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);
          }
     }
   // 上記の C# の「using」文の理由で Close を呼び出す必要がないですが、VB では必要です。
   //s.Close();
   // 結果を返します。
   return ms.ToArray();
 }
```

StreamReader を使用して、圧縮されたファイルを読み取る

StreamReader を使用して、圧縮されたファイルを読み取るには、次のコードを追加します。

コードの先頭に、これらの Imports 文 (Visual Basic)/(C#) を追加してください:

VisualBasic

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

C#

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

VisualBasic

```
' zip ファイルを開きます。
```

```
Dim zip As New C1ZipFile()
zip.Open("c:\temp\myzipfile.zip")
' 任意のエントリの入力ストリームを開きます。
Dim ze As C1ZipEntry = zip.Entries("someFile.cs")
Dim s As Stream = ze.OpenReader()
' ストリームの StreamReader を開きます。
Dim sr As New StreamReader(s)
' StreamReader を使用し、閉じます。
```

C#

```
// zip ファイルを開きます。
C1ZipFile zip = new C1ZipFile();
zip.Open(@"c:\temp\myzipfile.zip");
// 任意のエントリの入力ストリームを開きます。
C1ZipEntry ze = zip.Entries["someFile.cs"];
Stream s = ze.OpenReader();
// ストリームの StreamReader を開きます。
StreamReader sr = new StreamReader(s);
// StreamReader を使用し、閉じます。
```

zip ファイルから画像を取得する

この例では、2個のボタンと1個のリストボックスを使用して、.zip ファイルから画像を取得する方法を示します。

zip ファイルから画像を直接取得するために、最初に、いくつかの画像ファイルを1つの zip ファイルに圧縮するコードを追加します。この例では、**btnNew_Click** イベントにこのコードを追加します。ここで、ボタンがクリックされたときに画像の新しい .zip ファイルを作成します。

VisualBasic

```
' リソースディレクトリにある画像のリストを構築し、すべての画像を zip ファイルに追加します。
Dim zip As New C1ZipFile()
Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
    ' アプリケーションディレクトリを取得します。
    Dim s As String = Application.ExecutablePath
    s = s.Substring(0, s.IndexOf("\bin")) + "\resources"
    ' zip ファイルを作成します。
    zip.Create((s + "\images.zip"))
    ' zip ファイルに画像を格納し、リストを作成します。
    Dim f As String
    For Each f In  Directory.GetFiles(s)
        Dim fname As String = f.ToLower()
        ' 自分自身はスキップします。
        If fname.EndsWith("zip") Then
            GoTo ContinueForEach1
        End If
        ' リストに追加します。
        ListBox1.Items.Add(Path.GetFileName(fname))
```

```
        ' zip ファイルに追加します。
        zip.Entries.Add(fname)
        ContinueForEach1:
    Next f
End Sub
```

C#

```
// リソースディレクトリにある画像のリストを構築し、すべての画像を zip ファイルに追加します。
C1ZipFile zip = new C1ZipFile();
private void Form1_Load(object sender, System.EventArgs e)
{
    // アプリケーションディレクトリを取得します。
    string s = Application.ExecutablePath;
    s = s.Substring(0, s.IndexOf(@"\bin")) + @"\resources";
    // zip ファイルを作成します。
    zip.Create(s + @"\images.zip");
    // zip ファイルに画像を格納し、リストを作成します。
    foreach (string f in Directory.GetFiles(s))
    {
        string fname = f.ToLower();
        // 自分自身はスキップします。
        if (fname.EndsWith("zip")) continue;
        // リストに追加します。
        listBox1.Items.Add(Path.GetFileName(fname));
        // zip ファイルに追加します。
        zip.Entries.Add(fname);
    }
}
```

画像を選択できるようにするには、画像データを含むストリームを取得し（**OpenReader** メソッド）、次のコードを **listBox1_SelectionChanged** イベントと **StreamCopy** イベントに追加します。

VisualBasic

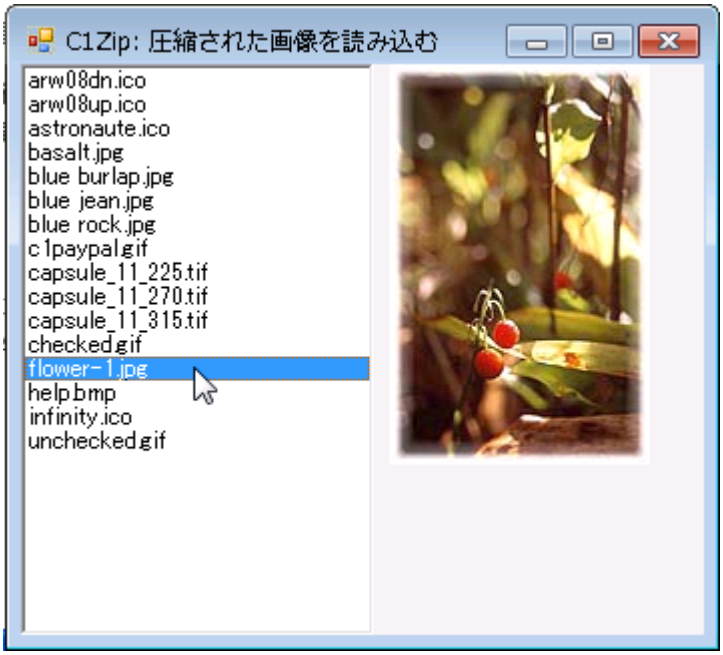
```
' 選択された画像を表示します。
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged
    ' 選択された項目を取得します。
    Dim item As String = CStr(listBox1.SelectedItem)
    ' 画像を圧縮ストリームから直接ロードします。
    Dim s As Stream = zip.Entries(item).OpenReader()
    Try
        pictureBox1.Image = CType(Image.FromStream(s), Image)
    Catch
    End Try
    ' ストリームを閉じます。
    s.Close()
End Sub
```

C#

```
// 選択された画像を表示します。
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    // 選択された項目を取得します。
    string item = (string)listBox1.SelectedItem;
    // 画像を圧縮ストリームから直接ロードします。
    Stream s = zip.Entries[item].OpenReader();
    try
    {
        pictureBox1.Image = (Image)Image.FromStream(s);
    }
    catch {}
    // ストリームを閉じます。
    s.Close();
}
```

このトピックの作業結果

ICO、TIFF、BMP、JPG などのいくつかのタイプの画像が表示されます。



文字列変数を zip ファイルに保存する

文字列変数を zip ファイルに保存するには、次のいずれかのメソッドを使用します。

- **C1ZipEntryCollection.OpenWriter** メソッド
  OpenWriter メソッドを使用して、ストリームライタを取得し、それに文字列を書き込み、最後に閉じます。データは、ストリームへの書き込み時に圧縮されます。ストリームを閉じると、ストリーム全体が zip ファイルに保存されます。

- **MemoryStream** メソッド
  MemoryStream メソッドを使用して、ストリームにデータを書き込み、zip ファイルに追加します。このメソッドで

は、**OpenWriter** メソッドを使用する場合より少し多くの作業を行う必要があります。それでも、これは扱いやすいメソッドです。

次のコードは、両方のメソッドを示しています。この例では、**OpenWriter** メソッドのコードは**button1_Click** イベントに示しています。MemoryStream メソッドのコードは**button2_Click** イベントに示しています。

VisualBasic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
    Dim str As String = "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."
    Dim zipFile As New C1ZipFile()
    zipFile.Create("c:\temp\strings.zip")
    ' 方法1:OpenWriter を使用する。
    Dim stream As Stream = zipFile.Entries.OpenWriter("Shakespeare.txt", True)
    Dim sw As New C1ZStreamWriter(stream)
    sw.Write(str)
    sw.Close()

    ' 方法2:メモリストリームを使用する。
    stream = New MemoryStream()
    sw = New C1ZStreamWriter(stream)
    sw.Write(str)
    sw.Flush()
    stream.Position = 0
    zipFile.Entries.Add(stream, "Shakespeare2.txt")
    stream.Close()
End Sub
```

C#

```
private void button1_Click(object sender, System.EventArgs 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.";
    C1ZipFile zipFile = new C1ZipFile();
    zipFile.Create(@"c:\temp\strings.zip");
    // 方法1:OpenWriter を使用する。
    Stream stream = zipFile.Entries.OpenWriter("Shakespeare.txt", true);
    C1ZStreamWriter sw = new C1ZStreamWriter(stream);
    sw.Write(str);
    sw.Close();
    // 方法2:メモリストリームを使用する。
    stream = new MemoryStream();
    sw = new C1ZStreamWriter(stream);
    sw.Write(str);
```

```
    sw.Flush();
    stream.Position = 0;
    zipFile.Entries.Add(stream, "Shakespeare2.txt");
    stream.Close();
}
```

圧縮のレベルを設定する

圧縮ファイルのサイズを最小化するには、次のコードを使用して、**C1ZStreamWriter** のコンストラクタで圧縮レベルを設定します。

VisualBasic

```
Dim fn As String = Path.GetTempFileName()
Dim fs As New FileStream(fn, FileMode.Create)
Dim compressor As New C1ZStreamWriter(fs, CompressionLevelEnum.BestCompression)
```

C#

```
string fn = Path.GetTempFileName();
FileStream fs = new FileStream(fn, FileMode.Create);
C1ZStreamWriter compressor = new C1ZStreamWriter(fs,
CompressionLevelEnum.BestCompression);
```

このコードサンプルでは、圧縮レベルを **BestCompression** に設定しています。これは、圧縮時間が最長で、圧縮速度が最低になります。**C1ZStreamWriter** のコンストラクタでは、ほかに次の圧縮レベルオプションを指定できます。

- **BestSpeed** を指定すると、圧縮時間は最短、圧縮速度は最高になります。
- **DefaultCompression** を指定すると、圧縮時間と速度が中程度になります。
- **NoCompression** を指定すると、データは圧縮されません。

パスワードを使用して zip ファイル保護する

パスワードで保護された zip ファイルを作成するには、エントリを作成する前に、**Password** プロパティを空以外の文字列に設定します。エントリごとに独自のパスワードを設定できます。次に例を示します。

VisualBasic

```
C1Zip.Password = "password"
C1Zip.Entries.Add(someFile)
```

C#

```
C1Zip.Password = "password";
C1Zip.Entries.Add(someFile);
```

このエントリを後から抽出するには、エントリを追加するときに適用された値と同じ値を **Password** プロパティに設定する必要があります。次に例を示します。

VisualBasic

```
' 失敗します。パスワードが設定されていません。
C1Zip.Password = ""
C1Zip.Entries.Extract(someFile)
' 失敗します。パスワードが不正です。
C1Zip.Password = "pass"
C1Zip.Entries.Extract(someFile)
' 正しく実行されます。
C1Zip.Password = "password"
C1Zip.Entries.Extract(someFile)
```

C#

```
// 失敗します。パスワードが設定されていません。
C1Zip.Password = "";
C1Zip.Entries.Extract(someFile);
// 失敗します。パスワードが不正です。
C1Zip.Password = "pass";
C1Zip.Entries.Extract(someFile);
// 正しく実行されます。
C1Zip.Password = "password";
C1Zip.Entries.Extract(someFile);
```

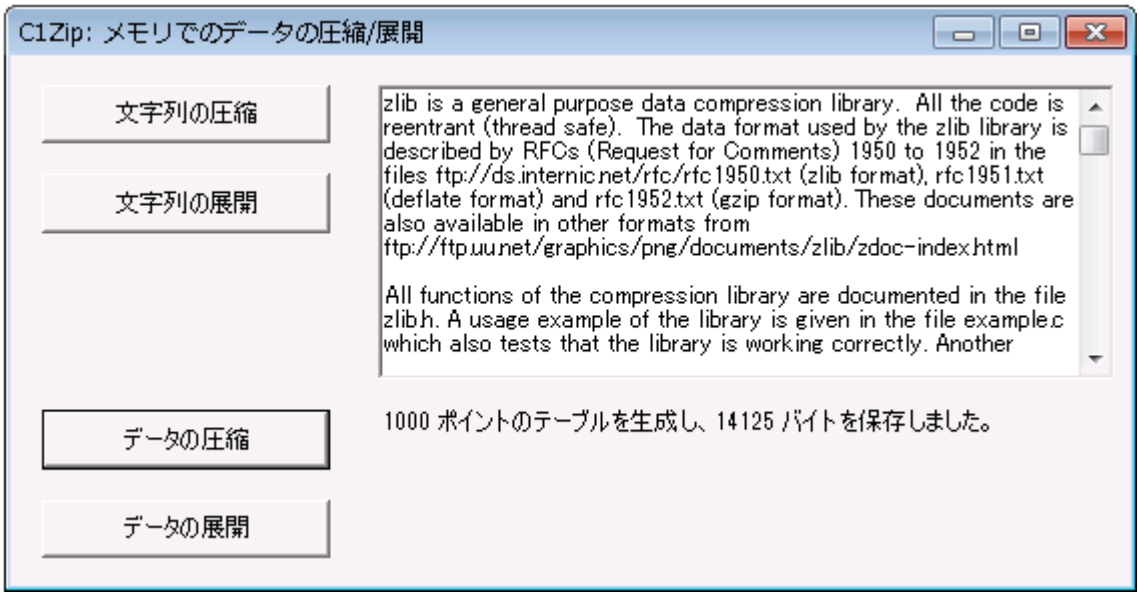
チュートリアル

次のトピックには、**C1Zip** ライブラリの主要な機能を具体的に説明するチュートリアルが含まれています。これらのチュートリアルでは、いくつかの単純なプロジェクトの作成を通して、それぞれの手順を詳しく説明しています。

チュートリアル	説明
メモリでのデータの圧縮	メモリで任意のデータを圧縮および展開する方法について説明します。この手法は、アプリケーションを実行している間、メモリストリームを維持する場合に役立ちます。ストリームを圧縮することで、アプリケーションに必要なメモリを減らすことができます。
ファイルの圧縮	個々のファイルを圧縮して、ファイルが占めるディスク容量と、ユーザーからのアクセス回数を減らす方法について説明します。これは zip ファイルではなく、個々の圧縮ファイルが対象であることに注意してください。zip ファイルについては最後のチュートリアルで扱います。
圧縮シリアライズ	Zip とWPF シリアライズを組み合わせて、オブジェクトを通常のサイズの数分の1のストリームに保存する方法について説明します。オブジェクトを XML ストリームにシリアライズすると、ディスク容量とネットワーク帯域幅を大幅に節約することができます。

メモリでのデータの圧縮

このチュートリアルでは、文字列や double などの基本データ型をメモリストリームに圧縮する方法、およびストリームからデータを読み取る際の展開方法について説明します。最終アプリケーションは、次の図のように表示されます。



手順1：メインフォームを作成します。

Visual Studio プロジェクトで新しい WPF/Silverlight プロジェクトを作成します。ツールボックスから、ドラッグ &ドロップ操作を実行するか、コンポーネントをダブルクリックして、次のコントロールをフォームに追加します。

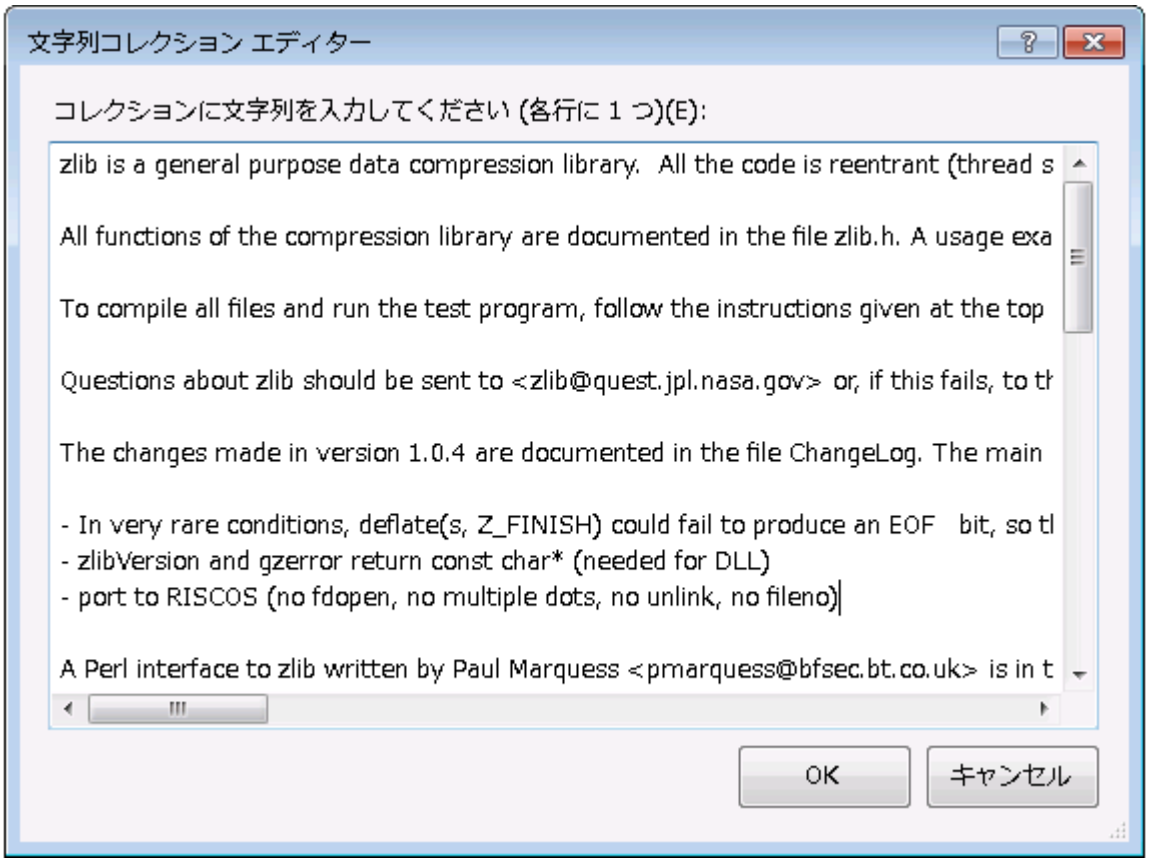
- フォームの左端に並べて4個の **Button** コントロール（前の図を参照）。[プロパティ]ウィンドウで、各 **Button** コントロールに次の変更を加えます。

Button	Button.Content プロパティ	Button.Name プロパティ	Button.IsEnabled プロパティ
1	文字列の圧縮	**btnCompressString**	True（デフォルト）

2	文字列の圧縮解除	**btnExpandString**	**False**
3	データの圧縮	**btnCompressData**	True（デフォルト）
4	データの圧縮解除	**btnExpandData**	**False**

[文字列の圧縮解除]ボタンと**[データの圧縮解除]**ボタンは、展開する圧縮データがある場合にのみ使用できます。

- フォームの右上に **RichTextBox**。**Height** プロパティを **250** に設定し、TextWrapping プロパティを Wrap に設定します。



- テキストボックスの下に **Label** コントロール。

手順2：C1.WPF.Zip/C1.Silverlight.Zip アセンブリに参照を追加します。

ソリューションエクスプローラウィンドウに移動し、**[すべてのファイルを表示]**ボタンをクリックします。**[参照]**を右クリックし、**[参照の追加]**メニューオプションを選択します。リストから C1.WPF.Zip/C1.Silverlight.Zip アセンブリを選択するか、ファイルを参照して C1.WPF.Zip.dll/C1.Silverlight.Zipdll ファイルを探します。

[MainPage.xaml.vb]タブ（C# では**[MainPage.xaml.cs]**タブ）を選択するか、**[表示]**→**[コード]**を選択して、コードエディタを開きます。ファイルの上部に、次のステートメントを追加します。

VisualBasic

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

C#

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

これで、C1.WPF.Zip/C1.Silverlight.Zip アセンブリで定義されているオブジェクトがプロジェクトから可視になり、タイピング量を大きく減らすことができます。

手順3：文字列を圧縮するコードを追加します。

[文字列の圧縮]コマンドボタンをダブルクリックし、**btnCompressString_Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private m_CompressedString As Byte()
Private Sub btnCompressString_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCompressString.Click
    ' 文字列を圧縮します。
    Dim ticks As Long = DateTime.Now.Ticks
    m_CompressedString = CompressString(textBox1.Text)
    ' 処理にかかる時間をユーザーに通知します。
    Dim ms As Integer
    ms = (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond
    Dim lenBefore As Integer = textBox1.Text.Length * 2
    Dim lenAfter As Integer = m_CompressedString.Length
    Dim msg As String
    msg = String.Format("Compressed from {0} bytes to " &  "{1} bytes in {2}
milliseconds.", lenBefore, lenAfter, ms)
    MessageBox.Show(msg, "Compressed", MessageBoxButtons.OK,
MessageBoxIcon.Information)
    ' これで、展開することができます。
    btnExpandString.Enabled = True
End Sub
```

C#

```
private byte[] _compressedString;
      private void btnCompressString_Click(object sender, RoutedEventArgs e)
        {
            // 文字列を圧縮します。
            long ticks = DateTime.Now.Ticks;
            richTextBox1.SelectAll();
            string text = richTextBox1.Selection.Text;
            _compressedString = CompressString(text);
            // 処理にかかる時間をユーザーに通知します。
            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);
```

```
            // これで、展開することができます。
            btnExpandString.IsEnabled = true;
        }
```

最初に重要な行は、**m_CompressedString** というメンバ変数の宣言です。これは、圧縮データ（バイト配列としてエンコードされる）を保持するために使用されます。次に重要な行は、ユーティリティ関数 **CompressString** の呼び出しです。これは、指定された文字列をバイト配列に圧縮します。これを後から展開して、元の文字列を復元できます。残りのコードは、圧縮プロセスにかかる時間を計算し、その値をダイアログボックスに表示するために使用されます。

lenBefore 変数は（文字列の長さ）×2で計算されています。これは、.NET 文字列が Unicode であり、各文字が実際は2バイトあるためです。

CompressString 関数を実装する次のコードを追加します。

VisualBasic

```
Public Function CompressString(ByVal str As String) As Byte()
    ' メモリストリームを開きます。
    Dim ms As MemoryStream = New MemoryStream()
    ' 圧縮プログラムストリームをメモリストリームにアタッチします。
    Dim sw As C1ZStreamWriter = New C1ZStreamWriter(ms)
    ' データを圧縮プログラムストリームに書き込みます。
    Dim writer As StreamWriter = New StreamWriter(sw)
    writer.Write(str)
    ' 保留中のデータをフラッシュします。
    writer.Flush()
    ' メモリバッファを返します。
    CompressString = ms.ToArray()
End Function
```

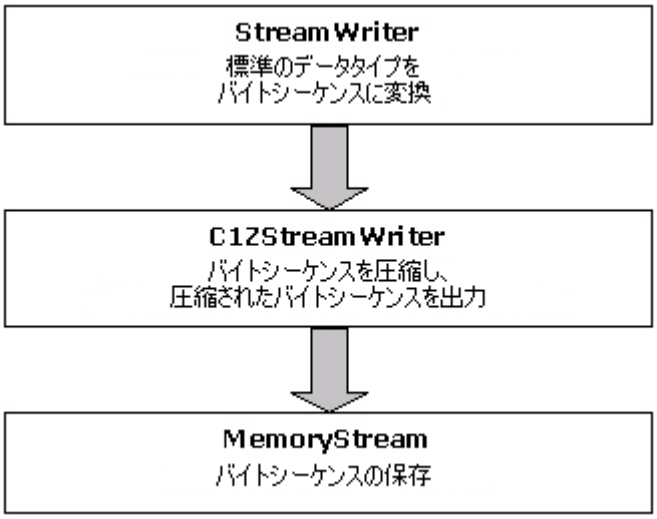
C#

```
public byte[] CompressString(string str)
{
    // メモリストリームを開きます。
    MemoryStream ms = new MemoryStream();
    // 圧縮プログラムストリームをメモリストリームにアタッチします。
    C1ZStreamWriter sw = new C1ZStreamWriter(ms);
    // データを圧縮プログラムストリームに書き込みます。
    StreamWriter writer = new StreamWriter(sw);
    writer.Write(str);
   // 保留中のデータをフラッシュします。
    writer.Flush();
    // メモリバッファを返します。
    return ms.ToArray();
}
```

この関数は、最初に新しいメモリストリームを作成します。このストリームにより、圧縮データを保持するメモリバッファが自動的に割り当てられます。

次に、**C1ZStreamWriter** オブジェクトを作成し、それを新しいメモリストリームにアタッチします。**C1ZStreamWriter** オブジェクトに書き込まれたデータはすべて、圧縮されてメモリストリームに書き込まれます。

C1ZStreamWriter オブジェクトは、バイトデータとバイト配列を書き込むための基本ストリームメソッドを提供するだけです。文字列や整数などの他の基本型を書き込むには、**StreamWriter** オブジェクトを **C1ZStreamWriter** にアタッチします。次の図に、この関数の動作を示します。



StreamWriter の設定が終了したら、後はその **Write** メソッドを呼び出して、文字列を圧縮メモリストリームに書き込むだけです。書き込みが完了したら、**Flush** メソッドも呼び出して、キャッシュされている入力をすべて書き出します。

最後に、**ToArray** メソッドを使用して、メモリストリームによって作成されたバイト配列を返します。

手順4：文字列を展開するコードを追加します。

文字列を展開するには、圧縮時に実行した手順を逆に実行する必要があります。**[文字列の圧縮解除]**ボタンをダブルクリックし、**btnExpandString_Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private Sub btnExpandString_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnExpandString.Click
    ' 文字列を展開します。
    Dim ticks As Long = DateTime.Now.Ticks
    TextBox1.Text = ExpandString(m_CompressedString)
    ' 処理にかかる時間をユーザーに通知します。
    Dim ms As Integer = (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond
    Dim lenBefore As Integer = m_CompressedString.Length
    Dim lenAfter As Integer = TextBox1.Text.Length * 2
    Dim msg As String
    msg = String.Format("Expanded from {0} bytes to {1} bytes " & "in {2}
milliseconds.", lenBefore, lenAfter, ms)
    MessageBox.Show(msg, "Expanded", MessageBoxButtons.OK,
MessageBoxIcon.Information)
End Sub
```

C#

```
  private void btnExpandString_Click(object sender, RoutedEventArgs e)
        {
            // 文字列を展開します。
```

```
            long ticks = DateTime.Now.Ticks;
            string text = ExpandString(_compressedString);
            richTextBox1.Selection.Text = text;
            // 処理にかかる時間をユーザーに通知します。
            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);
        }
```

重要な行は、バイト配列を受け取って元の文字列を返すユーティリティ関数 **ExpandString** の呼び出しです。次のように **ExpandString** 関数のコードを追加します。

VisualBasic

```
Public Function ExpandString(ByVal buffer As Byte()) As String
    ' バッファをメモリストリームにします。
    Dim ms As MemoryStream = New MemoryStream(buffer)
    ' 圧縮解除プログラムストリームをメモリストリームにアタッチします。
    Dim sr As C1ZStreamReader = New C1ZStreamReader(ms)
    ' 圧縮解除されたデータを読み取ります。
    Dim reader As StreamReader = New StreamReader(sr)
    ExpandString = reader.ReadToEnd()
End Function
```

C#

```
public string ExpandString(byte[] buffer)
{
    // バッファをメモリストリームにします。
    MemoryStream ms = new MemoryStream(buffer);
    // 圧縮解除プログラムストリームをメモリストリームにアタッチします。
    C1ZStreamReader sr = new C1ZStreamReader(ms);
    // 圧縮解除されたデータを読み取ります。
    StreamReader reader = new StreamReader(sr);
    return reader.ReadToEnd();
}
```

ここでプロジェクトを実行すると、文字列を圧縮/圧縮解除してみることができます。テキストボックス内のテキストを変更するか、テキストボックスに新しい内容を貼り付け、文字列を圧縮/展開して、どの程度圧縮されるかを確認できます。

手順5：バイナリデータを圧縮するコードを追加します。

文字列の圧縮と同様に、バイナリデータも簡単に圧縮できます。唯一の違いは、圧縮プログラムストリームに StreamWriter オブジェクトをアタッチする代わりに、BinaryWriter オブジェクトをアタッチするという点です。

[データの圧縮]ボタンをダブルクリックし、**btnCompressData_Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private m_CompressedData As Byte()
Private Sub btnCompressData_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCompressData.Click
    ' メモリストリームを開きます。
    Dim ms As MemoryStream = New MemoryStream()
    ' 圧縮プログラムストリームをメモリストリームにアタッチします。
    Dim sw As C1ZStreamWriter = New C1ZStreamWriter(ms)
    ' 圧縮プログラムストリームに BinaryWriter をアタッチします。
    Dim bw As BinaryWriter = New BinaryWriter(sw)
    ' 一連の数値をストリームに書き込みます。
    Dim i As Integer
    Dim count As Integer = 1000
    bw.Write(count)
    For i = 0 To count - 1
        Dim a As Double = i * Math.PI / 180.0
        bw.Write(i)
        bw.Write(a)
        bw.Write(Math.Sin(a))
        bw.Write(Math.Cos(a))
    Next i
    ' 保留中の出力をフラッシュします。
    bw.Flush()
    ' 圧縮データを保存します。
    m_CompressedData = ms.ToArray()
    ' 完了。
    Dim msg As String
    msg =String.Format("Generated table with {0} points," & " saved into {1} bytes",
count, m_CompressedData.Length)
    Label1.Text = msg
    ' これで、展開することができます。
    btnExpandData.Enabled = True
End Sub
```

C#

```
private byte[] _compressedData;
private void btnCompressData_Click(object sender, RoutedEventArgs e)
        {
            // メモリストリームを開きます。
            MemoryStream ms = new MemoryStream();
            // 圧縮プログラムストリームをメモリストリームにアタッチします。
            C1ZStreamWriter sw = new C1ZStreamWriter(ms);
            // 圧縮プログラムストリームに BinaryWriter をアタッチします。
            BinaryWriter bw = new BinaryWriter(sw);
            // 一連の数値をストリームに書き込みます。
            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));
            }
            // 保留中の出力をフラッシュします。
            bw.Flush();
            // 圧縮データを保存します。
            _compressedData = ms.ToArray();
            // 完了。
            string msg;
            msg = string.Format("Generated table with {0} points," +
saved into {1} bytes, count, _compressedData.Length);
            label1.Content = msg;
            // これで、展開することができます。
            btnExpandData.IsEnabled = true;
        }
```

このコードは、最初に **m_CompressedData** というメンバ変数を宣言します。これは、圧縮データ(バイト配列としてエンコードされる)を保持するために使用されます。

MemoryStream、**C1ZStreamWriter**、**BinaryWriter** の各オブジェクトを前と同様に設定します。ただし、ここでは、**StreamWriter** の代わりに **BinaryWriter** を使用します。

次に、**Write** メソッドを使用して、データをストリームに書き込みます。**BinaryWriter** オブジェクトは、すべての基本オブジェクト型をストリームに書き込むことができるように、このメソッドをオーバーロードしています。最後に、前と同様に **Flush** メソッドを使用して、キャッシュされているデータがあればすべて圧縮ストリームに書き出します。

手順6:バイナリデータを展開するコードを追加します。

圧縮されたバイナリデータを展開するには、通常のストリームと同様に、圧縮解除プログラムストリームを設定し、データを読み取ります。

[データの圧縮解除]コマンドボタンに次の **Click** イベントハンドラコードを追加します。

VisualBasic

```
Private Sub btnExpandData_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnExpandData.Click
    ' 保存されたデータのメモリストリームを開きます。
    Dim ms As MemoryStream = New MemoryStream(m_CompressedData)
    ' 圧縮解除プログラムストリームをメモリストリームにアタッチします。
    Dim sr As C1ZStreamReader = New C1ZStreamReader(ms)
    ' 圧縮解除されたデータを読み取ります。
    Dim i As Integer
    Dim br As BinaryReader = New BinaryReader(sr)
    Dim count As Integer = br.ReadInt32()
    For i = 0 To count - 1
        Dim deg As Integer = br.ReadInt32()
        Dim rad As Double = br.ReadDouble()
```

```
        Dim sin As Double = br.ReadDouble()
        Dim cos As Double = br.ReadDouble()
    Next i
    ' 処理が完了したことをユーザーに通知します。
    Dim msg As String
    msg = String.Format("Read table with {0} points " & "from stream with {1}
bytes.", count, m_CompressedData.Length)
    Label1.Text = msg
End Sub
```

C#

```
private void btnExpandData_Click(object sender, RoutedEventArgs e)
{
    // 保存されたデータのメモリストリームを開きます。
    MemoryStream ms = new MemoryStream(_compressedData);
    // 圧縮解除プログラムストリームをメモリストリームにアタッチします。
    C1ZStreamReader sr = new C1ZStreamReader(ms);
    // 圧縮解除されたデータを読み取ります。
   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();
    }
    // 処理が完了したことをユーザーに通知します。
   string  msg;
    msg = string .Format("Read table with {0} points " +
        "from stream with {1} bytes.", count, _compressedData.Length);
    label1.Content = msg;
}
```

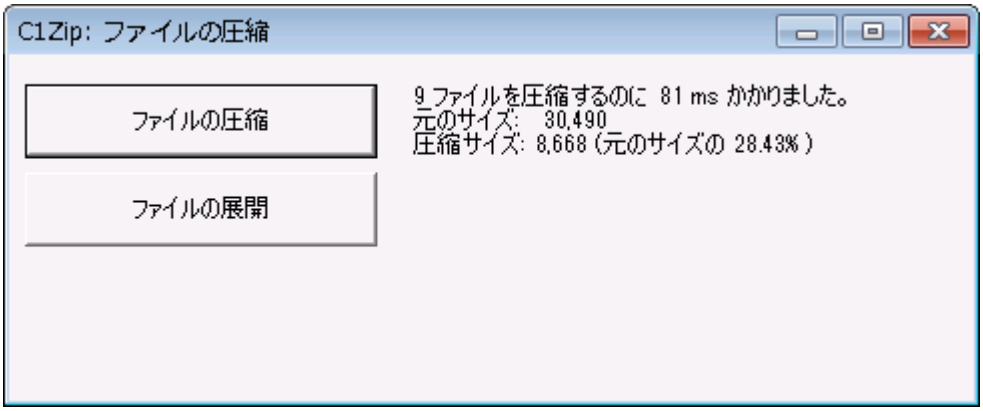
このコードでは、データは読み取られますが、画面には表示されません。コードをデバッグモードでステップ実行すると、読み取られたデータが書き込まれたデータと同じであることを確認できます。

プロジェクトを実行し、[データの圧縮]/[データの圧縮解除]ボタンをクリックすると、データが 14,125 バイトの配列に保存されることがわかります。このデータを通常のストリームに保存する場合は、28,004 バイト(4 + 1000 * (4 + 8 * 3))が必要です。このため、元のサイズの約半分の大きさに圧縮されたことになります。

これで**「メモリでのデータの圧縮」**チュートリアルは終了です。

ファイルの圧縮

このチュートリアルでは、個々のファイルを圧縮および展開する方法について説明します。これらは zip ファイルではなく、ディスク上の圧縮ストリームが対象であることに注意してください。最終アプリケーションは、次の図のように表示されます。



手順 1:メインフォームを作成します。

Visual Studio のツールボックスから、新しい WPF/Silverlight プロジェクトを開始し、フォームに次のコントロールを追加します。

- フォームの左端に並べて4個の **Button** コントロール(前の図を参照)。[プロパティ]ウィンドウで、次の変更を加えます。

Button	**Button.Content プロパティ**	**Button.Name プロパティ**	Button.IsEnabled プロパティ
1	ファイルの圧縮	**btnOpenSourceFile**	True(デフォルト)
2	圧縮ファイルの保存	**btnSaveCompressedFile**	**False**

[**展開ファイル**]の保存]ボタンは、保存する圧縮ファイルを開くまで使用できません。

- ボタンの右に **Label** コントロール。このコントロールには、圧縮/展開プロセスの統計値が表示されます。

手順 2: C1.WPF.Zip/C1.Silverlight.Zip アセンブリに参照を追加します。

ソリューションエクスプローラウィンドウに移動し、**[すべてのファイルを表示]**ボタンをクリックします。**[参照]**を右クリックし、**[参照の追加]**メニューオプションを選択します。リストから C1.WPF.Zip/C1.Silverlight.Zip アセンブリを選択するか、ファイルを参照して C1.WPF.Zip.dll/C1.Silverlight.Zip.dll ファイルを探します。

[MainPage.xaml.vb]タブ(C# では**[MainPage.xaml.cs]**タブ)を選択するか、**[表示]**→**[コード]**を選択して、コードエディタを開きます。ファイルの上部に、次のステートメントを追加します。

VisualBasic

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

C#

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

これで、**C1.WPF.Zip,/C1.Silverlight.Zip** アセンブリと **System.IO** アセンブリで定義されているオブジェクトがプロジェクトから可視になり、タイピング量を大きく減らすことができます。

手順 3:圧縮ファイルと展開後ファイルのディレクトリ名を定義します。

フォームのコードエディタで、次の定数を定義します。

VisualBasic

```
Private Const DIR_COMP = "\compressed"
Private Const DIR_EXP = "\expanded"
```

C#

```
private const string DIR_COMP = @"\compressed";
private const string DIR_EXP = @"\expanded";
```

これらは、圧縮ファイルと展開後ファイルを保存するディレクトリの名前です(ディスク上のチュートリアルアプリケーションが配置されているディレクトリからの相対ディレクトリ)。

手順 4:ファイルを圧縮するコードを追加します。

[**ファイルの圧縮**]コマンドボタンの **Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private Sub btnCompress_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCompress.Click
    ' アプリケーションディレクトリを取得します。
    Dim appPath As String = Application.ExecutablePath
    Dim i As Integer = appPath.IndexOf("\bin\")
    If i > 0 Then appPath = appPath.Substring(0, i)
    ' 圧縮ファイルのディレクトリを作成します。
    If (Directory.Exists(appPath + DIR_COMP)) Then
        Directory.Delete(appPath + DIR_COMP, True)
    End If
    Directory.CreateDirectory(appPath + DIR_COMP)
    ' 圧縮統計値を収集する準備をします。
    Dim count As Long
    Dim size As Long
    Dim sizeCompressed As Long
    Dim ticks As Long = DateTime.Now.Ticks
    ' アプリケーションディレクトリにあるすべてのファイルを圧縮ディレクトリに圧縮します。
    Dim files As String() = Directory.GetFiles(appPath)
    Dim srcFile As String
    For Each srcFile In files
        Dim dstFile As String
        dstFile = appPath + DIR_COMP + "\" + Path.GetFileName(srcFile) + ".cmp"
        ' ファイルを圧縮します。
        CompressFile(dstFile, srcFile)
        ' 統計値を更新します。
        count = count + 1
```

```
        size = size + New FileInfo(srcFile).Length
        sizeCompressed = sizeCompressed + New FileInfo(dstFile).Length
    Next srcFile
    ' 統計値を表示します。
    Dim msg As String = String.Format("Compressed {0} files in {1} ms." & vbCrLf &
"Original size:   {2:#,###}" & vbCrLf & "Compressed size: {3:#,###} ({4:0.00}% of
original)", count, (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond, size,
sizeCompressed, (sizeCompressed / size) * 100.0)
    Label1.Text = msg
    ' これで、展開することができます。
    btnExpand.Enabled = True
End Sub
```

C#

```
private void btnCompress_Click(object sender, EventArgs e)
{
    // アプリケーションディレクトリを取得します。
    string appPath = Application.ExecutablePath;
    int i = appPath.IndexOf(@"\bin\");
    if (i > 0) appPath = appPath.Substring(0, i);
    // 圧縮ファイルのディレクトリを作成します。
    if ((Directory.Exists(appPath + DIR_COMP)))
        Directory.Delete(appPath + DIR_COMP, true);
    Directory.CreateDirectory(appPath + DIR_COMP);
    // 圧縮統計値を収集する準備をします。
    long count = 0;
    long size = 0;
    long sizeCompressed = 0;
    long ticks = DateTime.Now.Ticks;
    // アプリケーションディレクトリにあるすべてのファイルを圧縮ディレクトリに圧縮します。
    foreach (string srcFile in Directory.GetFiles(appPath))
    {
        string dstFile = appPath + DIR_COMP + Path.GetFileName(srcFile) + ".cmp";
        // ファイルを圧縮します。
        CompressFile(dstFile, srcFile);
        // 統計値を更新します。
        count++;
        size += new FileInfo(srcFile).Length;
        sizeCompressed += new FileInfo(dstFile).Length;
    }
    // 統計値を表示します。
    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,
(sizeCompressed / size) * 100.0);
    label1.Text = msg;
    // これで、展開することができます。
    btnExpand.Enabled = true;
}
```

重要な行は、ユーティリティ関数 **CompressFile** の呼び出しです。ここで、選択されたファイルをそれぞれ圧縮します。圧縮ファイルは、アプリケーションフォルダにある ¥compressed ディレクトリに保存されます。ファイル名は、元のファイルの名前に拡張子 CMP が付加された名前になります。

次のように **CompressFile** 関数のコードを追加します。

VisualBasic

```
Private Function CompressFile( dstFile As String, srcFile As String) As Boolean
    ' ファイルを圧縮する準備をします。
    Dim retval As Boolean = True
    Dim srcStream As FileStream = Nothing
    Dim dstStream As FileStream = Nothing
    Try
        ' ファイルを開きます。
        srcStream = New FileStream(srcFile, FileMode.Open, FileAccess.Read)
        dstStream = New FileStream(dstFile, FileMode.Create, FileAccess.Write)
        ' 書き込み先のファイルの圧縮プログラムストリームを開きます。
        Dim sw As C1ZStreamWriter = New C1ZStreamWriter(dstStream)
        ' ソースを圧縮プログラムストリームにコピーします。
        StreamCopy(sw, srcStream)
    Catch
        ' 例外？ 呼び出し元に処理が失敗したことを通知します。
       retval = False
    Finally
        ' 必ずストリームを閉じます。
        If Not (srcStream Is Nothing) Then srcStream.Close()
        If Not (dstStream Is Nothing) Then dstStream.Close()
    End Try
    ' 完了。
    CompressFile = False
End Function
```

C#

```
private bool CompressFile(string  dstFile, string  srcFile)
{
    // ファイルを圧縮する準備をします。
    bool retval = true;
    FileStream srcStream = null;
    FileStream dstStream = null;
    try
    {
        // ファイルを開きます。
        srcStream = new FileStream(srcFile, FileMode.Open, FileAccess.Read);
        dstStream = new FileStream(dstFile, FileMode.Create, FileAccess.Write);
        // 書き込み先のファイルの圧縮プログラムストリームを開きます。
        C1ZStreamWriter sw = new C1ZStreamWriter(dstStream);
        // ソースを圧縮プログラムストリームにコピーします。
        StreamCopy(sw, srcStream);
    }
```

```
    catch
    {
        // 例外？ 呼び出し元に処理が失敗したことを通知します。
        retval = false;
    }
    finally
    {
        // 必ずストリームを閉じます。
        if (srcStream != null) srcStream.Close();
        if (dstStream != null) dstStream.Close();
    }
    // 完了。
    return false;
}
```

この関数は、最初に新しいファイルストリームを2つ作成します。1つはソースファイル用、もう1つは圧縮ファイル用です。次に、C1ZStreamWriter オブジェクトを作成し、それを出力先ストリームにアタッチします。次に、**StreamCopy** 関数を呼び出して、ソースファイルからデータを転送し、圧縮プログラムストリームに書き込みます。

StreamCopy 関数は、ストリーム間でバイトを単純にコピーします。次にコードを示します。

VisualBasic

```
Private Sub StreamCopy(dstStream As Stream, srcStream As Stream)
    Dim buffer(32768) As Byte
    Dim read As Integer
    Do
        read = srcStream.Read(buffer, 0, buffer.Length)
        dstStream.Write(buffer, 0, read)
    Loop While read > 0
    dstStream.Flush()
End Sub
```

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();
}
```

この関数は、コピーの完了時に **Flush** メソッドを呼び出して、キャッシュされているデータがあればすべて書き出します。この動作は、圧縮ストリームを処理する際は特に重要です。圧縮ストリームでは、圧縮率を向上させるために大量のデータがキャッシュされるためです。

手順 5:ファイルを展開するコードを追加します。

[**ファイルの展開**]コマンドボタンの **Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private Sub btnExpand_Click(sender As Object, e As EventArgs) Handles btnExpand.Click
    ' アプリケーションディレクトリを取得します。
    Dim appPath As String = Application.ExecutablePath
    Dim i As Integer      = appPath.IndexOf("\bin\")
    If i > 0 Then appPath = appPath.Substring(0, i)
    ' 展開後ファイルのディレクトリを作成します。
    If Directory.Exists(appPath + DIR_EXP) Then
        Directory.Delete(appPath + DIR_EXP, True)
    End If
    Directory.CreateDirectory(appPath + DIR_EXP)
    ' 圧縮統計値を収集する準備をします。
    Dim count As Long
    Dim size As Long
    Dim sizeExpanded As Long
    Dim ticks As Long = DateTime.Now.Ticks
    ' compressed ディレクトリにあるファイルをすべて expanded ディレクトリに展開します。
    Dim srcFile As String
    Dim files As String()
    files = Directory.GetFiles(appPath + DIR_COMP)
    For Each srcFile In files
        ' ファイルを展開します。
        Dim dstFile As String = appPath + DIR_EXP + "\" + Path.GetFileName(srcFile)
        dstFile = dstFile.Replace(".cmp", "")
        ExpandFile(dstFile, srcFile)
        ' 統計値を更新します。
        count = count + 1
        size = size + New FileInfo(srcFile).Length
        sizeExpanded = sizeExpanded + New FileInfo(dstFile).Length
    Next srcFile
    ' 統計値を表示します。
    Dim msg As String
    msg = String.Format("Expanded {0} files in {1} ms." & vbCrLf & "Original size:
{2:#,###}" & vbCrLf & "Expanded size: {3:#,###} ({4:0.00} x size of compressed)",
count, (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond, size,
sizeExpanded, sizeExpanded / size)
    Label1.Text = msg
End Sub
```

C#

```
private void btnExpand_Click(object sender, EventArgs e)
{
    // アプリケーションディレクトリを取得します。
    string appPath = Application.ExecutablePath;
```

```
    int i = appPath.IndexOf(@"\bin\");
    if (i > 0) appPath = appPath.Substring(0, i);
    // 展開後ファイルのディレクトリを作成します。
    if (Directory.Exists(appPath + DIR_EXP))
        Directory.Delete(appPath + DIR_EXP, true);
    Directory.CreateDirectory(appPath + DIR_EXP);
    // 圧縮統計値を収集する準備をします。
    long count = 0;
    long size = 0;
    long sizeExpanded = 0;
    long ticks = DateTime.Now.Ticks;
    // compressed ディレクトリにあるファイルをすべて expanded ディレクトリに展開します。
    foreach (string srcFile in Directory.GetFiles(appPath + DIR_COMP))
    {
        // ファイルを展開します。
        string  dstFile = appPath + DIR_EXP + @"\" + Path.GetFileName(srcFile);
        dstFile = dstFile.Replace(".cmp", "");
        ExpandFile(dstFile, srcFile);
        // 統計値を更新します。
        count++;
        size += new FileInfo(srcFile).Length;
        sizeExpanded += new FileInfo(dstFile).Length;
    }
    // 統計値を表示します。
    string msg = string .Format("Expanded {0} files in {1} ms.\r\n" + "Original size:
{2:#,###}\r\n" + "Expanded size: {3:#,###} ({4:0.00} x size of compressed)", count,
(DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond, size, sizeExpanded,
sizeExpanded / size);
    label1.Text = msg;
}
```

重要な行は、ユーティリティ関数 **ExpandFile** の呼び出しです。ここで、前に圧縮されたファイルを展開します。展開後ファイルは、アプリケーションフォルダにある ¥expanded ディレクトリに保存されます。ファイル名は、元のファイルの名前から拡張子 CMP が除去された名前になります。

次に、ExpandFile 関数のコードを示します。

VisualBasic

```
Private Function ExpandFile(dstFile As String, srcFile As String) As Boolean
    ' ファイルを展開する準備をします。
    Dim retval As Boolean = True
    Dim srcStream As FileStream = Nothing
    Dim dstStream As FileStream = Nothing
    Try
        ' ファイルを開きます。
        srcStream = New FileStream(srcFile, FileMode.Open, FileAccess.Read)
        dstStream = New FileStream(dstFile, FileMode.Create, FileAccess.Write)
        ' 圧縮ソースのエキスパンダストリームを開きます。
        Dim sr As C1ZstreamReader = New C1ZStreamReader(srcStream)
        ' エキスパンダストリームを出力先のファイルにコピーします。
        StreamCopy(dstStream, sr)
```

```
    Catch
        ' 例外？ 呼び出し元に処理が失敗したことを通知します。
        retval = False
    Finally
        ' 必ずストリームを閉じます。
        If Not (srcStream Is Nothing) Then srcStream.Close()
        If Not (dstStream Is Nothing) Then dstStream.Close()
    End Try
  ' 完了。
  ExpandFile = retval
End Sub
```

C#

```
{
    // ファイルを展開する準備をします。
    bool retval = true;
    FileStream srcStream = null;
    FileStream dstStream = null;
    try
    {
        // ファイルを開きます。
        srcStream = new FileStream(srcFile, FileMode.Open, FileAccess.Read);
        dstStream = new FileStream(dstFile, FileMode.Create, FileAccess.Write);
        // 圧縮ソースのエキスパンダストリームを開きます。
        C1ZStreamReader sr = new C1ZStreamReader(srcStream);
        // エキスパンダストリームを出力先のファイルにコピーします。
        StreamCopy(dstStream, sr);
    }
    catch
    {
        // 例外？ 呼び出し元に処理が失敗したことを通知します。
        retval = false;
    }
    finally
    {
        // 必ずストリームを閉じます。
        if (srcStream != null) srcStream.Close();
        if (dstStream != null) dstStream.Close();
    }
    // 完了。
    return retval;
}
```

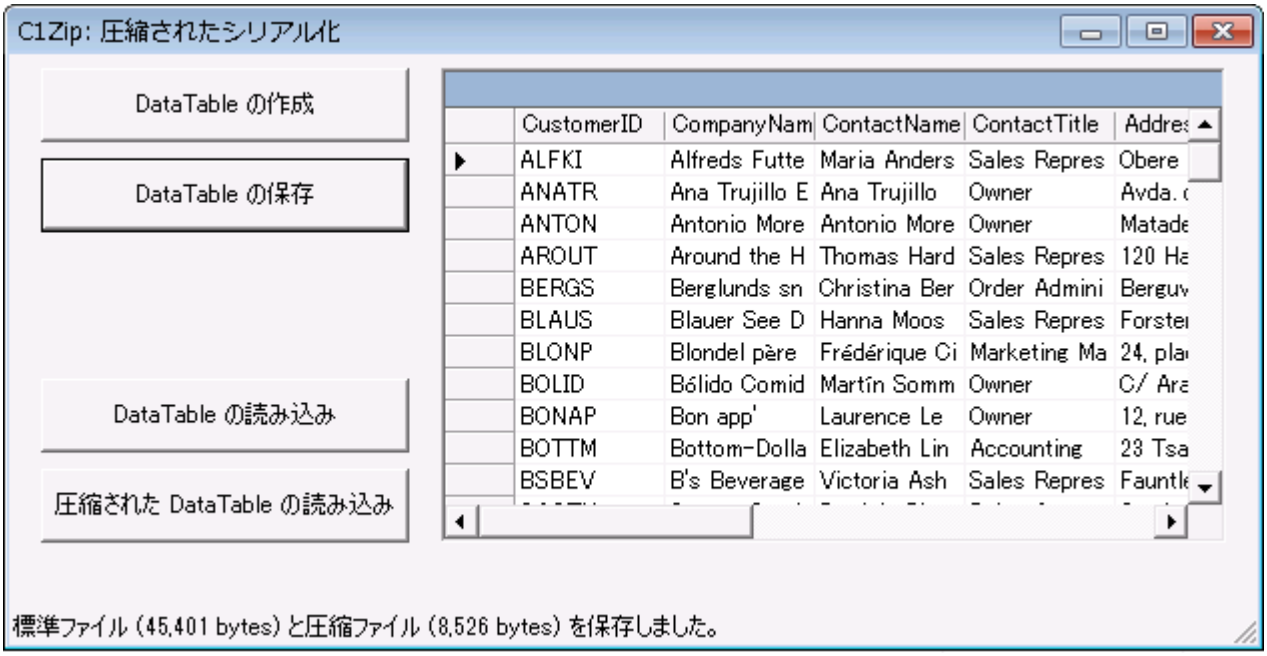
この関数は CompressFile に似ていますが、**C1ZStreamWriter** を出力先ストリームにアタッチする代わりに、**C1ZStreamReader** をソースストリームにアタッチする点が異なります。

これで「**ファイルの圧縮**」チュートリアルは終了です。

圧縮シリアライズ

このチュートリアルでは、オブジェクトを圧縮ファイルにシリアライズする方法、およびそのオブジェクトをメモリにロードする方法について説明します。

このサンプルは、人物リストのデータテーブルを作成します。テーブルは、通常のストリームと圧縮ストリームに保存(シリアライズ)されます。最後に、どちらかのストリームからデータがロードされます。最終アプリケーションは、次の図のように表示されます。



手順 1:メインフォームを作成します。

Visual Studio のツールボックスから、新しい WPF/Silverlight プロジェクトを開始し、フォームに次のコントロールを追加します。

フォームの左端に並べて5個の Button コントロール(前の図を参照)。[プロパティ]ウィンドウで、各 Button コントロールに次の変更を加えます。

- フォームの左端に並べて4個の **Button** コントロール(前の図を参照)。[プロパティ]ウィンドウで、各 **Button** コントロールに次の変更を加えます。

Button	**Button.Text プロパティ**	Button.Name プロパティ	**Button.Enabled プロパティ**
1	DataTable の作成	**btnCreate**	True(デフォルト)
2	DataTable の保存	**btnSave**	**False**
3	DataTable の読み込み	**btnLoad**	**False**
4	圧縮された DataTable の読み込み	**btnLoadCompressed**	**False**

  保存ボタンとロードボタンは、データテーブルを作成または保存するまで使用できません。

- フォームの右に **DataGridView** コントロール。
- フォームの下部にドッキングされる **ToolStripStatusLabel** コントロール。このコントロールを追加するには、最初に **StatusStrip** コントロールをフォームに追加します。次に、**[ToolStripStatusLabel の追加]**ドロップダウン矢印をクリックし、**[StatusLabel]**を選択します。**ToolStripStatusLabel** コントロールが表示され、フォームの下部にドッキングされます。

手順 2:参照と Imports ステートメントを追加します。

ソリューションエクスプローラウィンドウに移動し、**[すべてのファイルを表示]**ボタンをクリックします。[参照]を右クリックし、**[参**

照の追加]メニューオプションを選択します。リストから C1.WPF.Zip/C1.Silverlight.Zip アセンブリを選択するか、ファイルを参照して C1.WPF.Zip.4.dll/C1.Silverlight.Zip.dll ファイルを探します。

[Form1.vb]タブ（C# では**[Form1.cs]**タブ）を選択するか、**[表示]**→**[コード]**を選択して、コードエディタを開きます。ファイルの上部に、次のステートメントを追加します。

VisualBasic

```
Imports System.IO
Imports System.Data.OleDb
Imports System.Runtime.Serialization.Formatters.Binary
Imports C1.C1Zip
```

C#

```
using System.IO;
using System.Data.OleDb;
using System.Runtime.Serialization.Formatters.Binary;
using C1.C1Zip;
```

これで、プロジェクト内で使用されるクラスの名前空間が宣言されます。

手順 3：定数を宣言します。

フォームのコードエディタで、フォーム実装の本体に、次の行を入力するかコピーします。

VisualBasic

```
Private Const FN_REGULAR = "\DataTable.regular"
Private Const FN_COMPRESSED = "\DataTable.compressed"
Private Const MDBFILE = "C:\Users\<ユーザー名>\Documents\ComponentOne
Samples\Common\NWIND.MDB"
```

C#

```
private const string FN_REGULAR = @"\DataTable.regular";
private const string FN_COMPRESSED = @"\DataTable.compressed";
private const string MDBFILE = @"C:\Users\<ユーザー名>\Documents\ComponentOne
Samples\Common\NWIND.MDB";
```

これらの定数は、データテーブルへの入力時に使用されるデータベースの名前、およびデータのシリアライズに使用されるファイルの名前を定義します。

手順 4：データテーブルを作成するコードを追加します。

[DataTable の作成]ボタンの **Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private Sub btnCreate_Click(sender As Object,e As EventArgs) Handles btnCreate.Click
    ' テーブルを開きます。
    Dim conn As String
    conn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & MDBFILE & ";"
    Dim rs As String = "select * from customers"
    ' ステータスを表示します。
    Cursor = Cursors.WaitCursor
    ToolStripStatusLabel1.Text = "Loading data from mdb file..."
    ' データをロードします。
    Dim da As OleDbDataAdapter = New OleDbDataAdapter(rs, conn)
    Dim ds As DataSet = New DataSet()
    Try
        da.Fill(ds)
    Catch
        MessageBox.Show("Could not load data from " + MDBFILE)
    End Try
    ' ステータスを表示します。
    Cursor = Cursors.Default
    ToolStripStatusLabel1.Text = "Loaded " & ds.Tables(0).Rows.Count & " records from
mdb file."
    ' グリッドに連結します。
    DataGridView1.DataSource = ds.Tables(0)
    ' 保存ボタンを有効化します。
    btnSave.Enabled = True
End Sub
```

C#

```
private void btnCreate_Click(object sender,EventArgs e)
{
    // テーブルを開きます。
    string conn =  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + MDBFILE + ";";
    string rs = "select * from customers";
    // ステータスを表示します。
    Cursor = Cursors.WaitCursor;
    toolStripStatusLabel1.Text = "Loading data from mdb file...";
    // データをロードします。
    OleDbDataAdapter da = new OleDbDataAdapter(rs, conn);
    DataSet ds = new DataSet();
    try
    {
        da.Fill(ds);
    }
    catch
    {
        MessageBox.Show("Could not load data from " + MDBFILE);
    }
    // ステータスを表示します。
    Cursor = Cursors.Default;
    toolStripStatusLabel1.Text = "Loaded " + ds.Tables[0].Rows.Count +  " records
```

```
from mdb file.";
    // グリッドに連結します。
    dataGridView1.DataSource = ds.Tables[0];
    // 保存ボタンを有効化します。
    btnSave.Enabled = true;
}
```

この関数は、標準の ADO.NET オブジェクトとメソッドを使用して、**DataTable** オブジェクトを作成し、このオブジェクトにデータを入力します。このオブジェクトは、次に **DataGrid** コントロールに連結されます。

手順 5:データテーブルを保存するコードを追加します。

[データテーブルの保存]ボタンの **Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private Sub btnSave_Click(sender As Object, e As EventArgs) Handles btnSave.Click
    ' グリッドからデータテーブルを取得します。
    Dim dt As DataTable = DataGridView1.DataSource
    ' ステータスを表示します。
    Cursor = Cursors.WaitCursor
    ToolStripStatusLabel1.Text = "Serializing data to regular file..."
    ' データセットを通常のファイルにシリアライズします。
    Dim fn As String = Application.StartupPath + FN_REGULAR
    Dim fs As FileStream = New FileStream(fn, FileMode.Create)
    Dim bf As BinaryFormatter = New BinaryFormatter()
    bf.Serialize(fs, dt)
    Dim lenRegular As Long = fs.Length
    fs.Close()
    ' ステータスを表示します。
    Cursor = Cursors.WaitCursor
    ToolStripStatusLabel1.Text = "Serializing data to compressed file..."
    ' データセットを圧縮ファイルにシリアライズします。
    fn = Application.StartupPath & FN_COMPRESSED
    fs = New FileStream(fn, FileMode.Create)
    Dim compressor As C1ZStreamWriter = New C1ZStreamWriter(fs)
    bf = New BinaryFormatter()
    bf.Serialize(compressor, dt)
    Dim lenCompressed As Long = fs.Length
    fs.Close()
    ' ステータスを表示します。
    Cursor = Cursors.Default
    ToolStripStatusLabel1.Text = string.Format( "Saved to regular file ({0:#,###}
bytes) and " & "compressed file ({1:#,###} bytes)", lenRegular, lenCompressed)
    ' ロードボタンを有効化します。
    btnLoad.Enabled = True
    btnLoadCompressed.Enabled = True
End Sub
```

C#

```
private void btnSave_Click(object sender, EventArgs e)
{
    // グリッドからデータテーブルを取得します。
     DataTable dt = (DataTable)dataGridView1.DataSource;
    // ステータスを表示します。
    Cursor = Cursors.WaitCursor;
    toolStripStatusLabel1.Text = "Serializing data to regular file...";
    // データセットを通常のファイルにシリアライズします。
    string fn = Application.StartupPath + FN_REGULAR;
    FileStream fs = new FileStream(fn, FileMode.Create);
    BinaryFormatter bf = new BinaryFormatter();
    bf.Serialize(fs, dt);
    long lenRegular = fs.Length;
    fs.Close();
    // ステータスを表示します。
    Cursor = Cursors.WaitCursor;
    toolStripStatusLabel1.Text = "Serializing data to compressed file...";
    // データセットを圧縮ファイルにシリアライズします。
    fn = Application.StartupPath + FN_COMPRESSED;
    fs = new FileStream(fn, FileMode.Create);
    C1ZStreamWriter compressor = new C1ZStreamWriter(fs);
    bf = new BinaryFormatter();
    bf.Serialize(compressor, dt);
    long lenCompressed = fs.Length;
    fs.Close();
    // ステータスを表示します。
    Cursor = Cursors.Default;
    toolStripStatusLabel1.Text = string.Format("Saved to regular file ({0:#,###}
bytes) and " + "compressed file ({1:#,###} bytes)", lenRegular, lenCompressed);
    // ロードボタンを有効化します。
    btnLoad.Enabled = true;
    btnLoadCompressed.Enabled = true;
}
```

最初のコードセットは、**DataTable** を通常のファイルにシリアライズします。2番目のコードセットは、**DataTable** を圧縮ファイルにシリアライズします。1行のコードを追加するだけで、データを圧縮することができます。

どちらの場合も、シリアライズは **BinaryFormatter** オブジェクトによって実行されます。唯一の違いは、最初のコードセットでは **Serialize** メソッドが通常のファイルストリームをパラメータとして呼び出され、2番目のコードセットでは代わりに **C1ZStreamWriter** が使用されるという点です。

手順 6:通常のファイルからデータテーブルをロードするコードを追加します。

[DataTable の読み込み]ボタンの **Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private Sub btnLoad_Click(sender As Object, e As EventArgs) Handles btnLoad.Click
    ' グリッドをクリアし、ステータスを表示します。
    Cursor = Cursors.WaitCursor
    DataGridView1.DataSource = Nothing
    ToolStripStatusLabel1.Text = "Loading from regular file..."
```

```
    ' 通常のファイルからシリアライズ解除します。
    Dim fn As String = Application.StartupPath & FN_REGULAR
    Dim fs As FileStream = New FileStream(fn, FileMode.Open)
    Dim ticks As Long = DateTime.Now.Ticks
    Dim bf As BinaryFormatter = New BinaryFormatter()
    Dim dt As DataTable = bf.Deserialize(fs)
    Dim ms As Long = (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond
    fs.Close()
    ' 結果を表示します。
    Cursor = Cursors.Default
    DataGridView1.DataSource = dt
    ToolStripStatusLabel1.Text = "Loaded from regular file in " & ms.ToString() & "
ms."
End Sub
```

C#

```
private void btnLoad_Click(object sender, EventArgs e)
{
    // グリッドをクリアし、ステータスを表示します。
    Cursor = Cursors.WaitCursor;
    dataGridView1.DataSource = null;
    toolStripStatusLabel1.Text = "Loading from regular file...";
    // 通常のファイルからシリアライズ解除します。
    string fn = Application.StartupPath + FN_REGULAR;
    FileStream fs = new FileStream(fn, FileMode.Open);
    long ticks = DateTime.Now.Ticks;
    BinaryFormatter bf = new BinaryFormatter();
    DataTable dt = (DataTable)bf.Deserialize(fs);
    long ms = (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond;
    fs.Close();
    // 結果を表示します。
    Cursor = Cursors.Default;
    dataGridView1.DataSource = dt;
    toolStripStatusLabel1.Text = "Loaded from regular file in " + ms.ToString() + "
ms.";
}
```

最初に重要な行は新しい **BinaryFormatter** オブジェクトの作成、次に重要な行はその **Deserialize** メソッドの呼び出しです。この **Deserialize** メソッドはパラメータを1つ受け取ります。それは、オブジェクトが定義されたストリームです。この場合、ストリームは通常のファイルストリームです。

手順 7: 圧縮ファイルからデータテーブルをロードするコードを追加します。

[圧縮された DataTable の読み込み]ボタンの **Click** イベントを処理する次のコードを追加します。

VisualBasic

```
Private Sub btnLoadCompressed_Click(sender As Object, e As EventArgs) Handles
btnLoadCompressed.Click
```

```
    ' グリッドをクリアし、ステータスを表示します。
    Cursor = Cursors.WaitCursor
    DataGridView1.DataSource = Nothing
    ToolStripStatusLabel1.Text = "Loading from compressed file..."
    ' 圧縮ファイルからシリアライズ解除します。
    Dim fn As String = Application.StartupPath + FN_COMPRESSED
    Dim fs As FileStream = New FileStream(fn, FileMode.Open)
    Dim ticks As Long   = DateTime.Now.Ticks
    Dim decompressor As C1ZstreamReader
    decompressor = New C1ZStreamReader(fs)
    Dim bf As BinaryFormatter = New BinaryFormatter()
    Dim dt As DataTable = bf.Deserialize(decompressor)
    Dim ms As Long = (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond
    fs.Close()
    ' 結果を表示します。
    Cursor = Cursors.Default
    DataGridView1.DataSource = dt
    ToolStripStatusLabel1.Text = "Loaded from compressed file in " & ms.ToString() &
" ms."
End Sub
```

C#

```
private void btnLoadCompressed_Click(object sender, EventArgs e)
{
    // グリッドをクリアし、ステータスを表示します。
    Cursor = Cursors.WaitCursor;
    dataGridView1.DataSource = null;
    toolStripStatusLabel1.Text = "Loading from compressed file...";
    // 圧縮ファイルからシリアライズ解除します。
    string fn = Application.StartupPath + FN_COMPRESSED;
    FileStream fs = new FileStream(fn, FileMode.Open);
    long ticks = DateTime.Now.Ticks;
    C1ZStreamReader decompressor;
    decompressor = new C1ZStreamReader(fs);
    BinaryFormatter bf = new BinaryFormatter();
    DataTable dt = (DataTable)bf.Deserialize(decompressor);
    long ms = (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond;
    fs.Close();
    // 結果を表示します。
    Cursor = Cursors.Default;
    dataGridView1.DataSource = dt;
    toolStripStatusLabel1.Text = "Loaded from compressed file in " + ms.ToString() +
" ms.";
}
```

重要な行は、通常のファイルからデータをシリアライズ解除するコードと同じです。唯一の違いは、**Deserialize** メソッドに通常のファイルストリームを渡す代わりに、**C1ZStreamReader** オブジェクトを使用すると点です。

これで、「**圧縮シリアライズ**」チュートリアルは終了です。