Zip for .NET

2018.04.12 更新

グレープシティ株式会社

Zip for .NET

目次

はじめに

データをすばやく圧縮し、ディスク領域とネットワーク帯域幅を節約して、圧縮データを容易に操作および処理します。**Zip for .NET** では、アプリケーションからシステムおよび隠しファイルを含むファイルを圧縮し、圧縮ファイルおよびフォルダを追加／削除できます。さらに、ストリームに保存された zip ファイルの読み取りおよび書き込みを行うことができます。

Zip for .NET のクラスは、WinForms、ASP.NET、WPF、Mobile を含むすべての .NET 開発プラットフォームでサポートされ、ComponentOne for Silverlight では特別な Silverlight バージョンが用意されています。

コンポーネントをプロジェクトに組み込む方法

コンポーネントの組み込み

Visual Studio では、ツールボックスにコンポーネントを追加しただけでは、プロジェクトにコンポーネントを追加したことにはなりません。プロジェクトの参照設定へ追加された時点でコンポーネントが組み込まれます。

以下のいずれかの操作を行うとプロジェクトへコンポーネントが組み込まれます。

- フォームにコンポーネントを配置する
- ソリューションエクスプローラ上で参照の追加を行う

プロジェクトに組み込まれているコンポーネントの一覧は、ソリューションエクスプローラで確認できます。また、各コンポーネントが使用している DLL もソリューションエクスプローラに登録される場合があります。詳細については、Visual Studio の製品ヘルプを参照してください。

本製品で使用しているコンポーネントの一覧を以下に示します。

ファイル	内容
C1.C1Zip.2.dll	本体アセンブリ
C1.C1Zip.4.dll	本体アセンブリ（※）

※ .NET Framework 4 以上でご利用いただけます。

コンポーネントのランタイムファイル

Zip for WinForms のランタイムファイルは、C:\Program Files\ComponentOne\WinForms\Bin\ フォルダにインストールされる次のファイルです。

ファイル	内容
C1.C1Zip.4.dll	本体アセンブリ（※）

※ .NET Framework 4 以上でご利用いただけます。

注意:本製品に含まれているファイルのうち、上記以外のファイルは配布できません。

主な特長

役立ちそうな **ZIP for .NET** の主な機能として、次の事項が挙げられます。

- **主な操作**

  ZIP for .NET を使用して、以下を実行できます。

  - グローバルな情報を zip ファイルで入手する。
  - コレクションオブジェクトを使用して、zip ファイルのコンテンツの詳細リストを取得する。
  - ファイル内のファイルを削除する。
  - ファイルとそのコンテンツの完全性をテストする。
  - ファイル内の個々のファイルのコメントを追加および取得する。
  - グローバル zip ファイルのコメントを取得および設定する。
  - 各ファイルの zip ファイルに保存されているパス情報を制御する。
  - zip システムと隠しファイル。
  - zip 対象ファイルに適用される圧縮量を制御する。
  - ファイルを圧縮解除するパスを指定する。
  - 最大限の安全を確保するために zip ファイルの一時コピーで作業する。
- **フォルダの圧縮および展開**

  Zip を使用して、フォルダ構造を維持しながら、簡単にフォルダを圧縮および展開できます。

- **ストリームとの間の読み取りおよび書き込み**

  Zip は、C1ZipFile.Open(Stream) メソッドを使用して、実際のファイルだけではなくストリームに保存された zip ファイルの読み取りおよび書き込みを行うことができます。これにより、実際のファイルではなく、アプリケーションリソースに埋め込まれた zip ファイルやデータベースフィールドに格納された zip ファイルを読み取ることができます。

- **ファイル情報の取得および設定**

  zip ファイルのコンテンツの詳細リストを含む zip ファイル情報を簡単に取得できます。zip ファイルの個別エントリに対するコメントの追加および取得、パス情報の制御、およびグローバル zip ファイルのコメントの取得および設定により、ファイル情報を制御することもできます。

- **ファイルの完全性と安全**

  Zip を使用して、zip ファイルとそのコンテンツの完全性をテストしたり、最大限の安全を確保するために zip ファイルの一時ファイルで作業できます。

- **ファイルの圧縮と圧縮ファイルの操作**

  システムおよび隠しファイルを含むファイルを圧縮したり、zip ファイル内のファイルの削除、zip 対象ファイルに適用される圧縮量の制御、およびファイルを圧縮解除するパスの指定を行うことにより簡単に圧縮ファイルを操作できます。

- **ZLIB の利点の取り込み**

  ZLIB は、移植可能な、プラットフォーム間で共通のデータ形式を備えた、汎用性のある、無損失のデータ圧縮ライブラリです。Unix 圧縮および GIF 画像形式の LZW 圧縮と異なり、ZLIB 圧縮法は基本的にデータを拡大することはなく（LZW では、極端な場合ファイルサイズが2倍から3倍になります）、メモリ使用量は入力データと無関係です。

- **Zip64のサポート**

  C1Zip は Zip64 ファイルをサポートするようになりました。この結果、通常の 4 GB 制限を超えたエントリが可能になり、zip ファイルあたりのエントリでもより大きなエントリが可能になりました。zip ファイルあたりの最大エントリ数は 2,147,483,647（int.MaxValue）です。非圧縮エントリの最大サイズは 9,223,372,036,854,775,807（long.MaxValue）です。

- **高速、低メモリ使用量の圧縮エンジン**

  低メモリ使用量の圧縮エンジンにより、以下を実行できます。

  - 新規または既存の zip ファイルにファイルまたはメモリバッファを zip する。
  - ディスクまたは直接メモリにファイルを解凍する。
  - 文字列またはバッファをメモリに完全に圧縮および展開する。
  - 最大限の柔軟性を発揮するストリームベースの圧縮と展開。
- **100%マネージド C# コード**

  C1.C1Zip.ZLib は Adler と Gailly の ZLIB コードの C# 実装であり、ストリームベースの圧縮と展開を処理するメインクラスの ZStream を含んでいます。ZLIB は実際のデータの圧縮と展開を処理します。

  ZLIB は、ほとんどすべてのコンピュータハードウェアおよびオペレーティングシステムで使用できるように、無償かつ汎用性のある、無損失のデータ圧縮ライブラリとして設計されています。ZLIB データ形式は、それ自体プラットフォーム間で移植可能です。Unix 圧縮および GIF 画像形式で使用される LZW 圧縮法と異なり、ZLIB で現在使用されている圧縮法は基本的にデータを拡大することはありません（LZW では、極端な場合ファイルサイズが2倍から3倍になることがあります）。また、ZLIB のメモリ使用量は入力データと無関係です.

- **単純かつ直感的なオブジェクトモデル**

  Zip for .NET には、以下の3つのメインクラスが含まれています。

  - **C1ZStreamWriter** は、通常のデータを入力として取り、それを圧縮し、それを基底にあるストリームに書き出すストリームオブジェクトです。データをファイル、メモリ、その他あらゆるタイプのストリームに圧縮できます。**BinaryWriter** オブジェクトを C1ZStreamWriterにアタッチして、バイト配列を処理せずに直接オブジェクトを書き込むことができます。
  - **C1ZStreamReader**は、圧縮ストリームを入力として取り、それを展開して、基底にある圧縮ストリームからデータを読み取るストリームオブジェクトです。**BinaryReader** オブジェクトを C1ZStreamReaderにアタッチして、バイト配列を処理せずに直接オブジェクトを読み取ることができます。
  - **C1ZipFile** は zip ファイルを処理するクラスです。zip ファイルを作成したり、開いたり、変更したりできます。C1ZipFile には、zip ファイルのエントリのコレクションを表す Entries プロパティが用意されています。

製品の概要

製品の概要

C1Zip ライブラリのクラスは、以下の3つに分類されます。

レベル	主なクラス	説明
High	**C1ZipFile**, **C1ZipEntry**, **C1ZipEntryCollection**	これらのクラスを使用し、Zip ファイルを作成する／開く／管理する操作を行います。Zip ファイルの内容を調べたり、その完全性をテストしたり、Zip ファイルに対してエントリを追加、削除、抽出したりできます。
Medium	**C1ZStreamReader**, **C1ZStreamWriter**	Use これらのクラスを使用し、通常の .NET ストリーム（メモリ、ファイル、ネットワークの各ストリームなど）に対してデータの圧縮／展開を行います。
Low	**ZStream**	これは **C1Zip** の最下位クラスです。広く普及しているデータ圧縮ライブラリである Zlib の 100% C# 実装で、Jean-loup Gailly と Mark Adler により書かれたものです。 ZStream は C1Zip の上位クラスが使用します。 C1ZipFile クラス、C1ZipEntry クラス、C1ZipEntryCollection クラス C1ZStreamReader クラスと C1ZStreamWriter クラス ZStream クラス.

C1ZipFile クラス、C1ZipEntry クラス、C1ZipEntryCollection クラス

これらは、**C1Zip** ライブラリの中で最高位クラスです。これらのクラスを使用すると、Zip ファイルを作成、管理できます。アプリケーションを格納するときに Zip ファイルを使用すると、以下の利点があります。

- 多くのファイルを1つに統合できるため、アプリケーションの配備がより簡単になります。
- 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できます。
- 形式はオープンスタンダードで、広く普及している多くのアプリケーションでサポートされています。.

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、それを既存の Zip ファイルに付加するか、このオブジェクトに指示して新しい空の Zip ファイルを作成できます。たとえば、以下のように記述します。

Visual Basic コードの書き方

Visual Basic

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

C# コードの書き方

C#

```
//C1ZipFile オブジェクトを作成します。
```

```
C1ZipFile myZip = new C1ZipFile();
//新規(空の)zip ファイルを作成します。
myZip.Create("New.zip");
//既存の zip ファイルを開きます。
myZip.Open("Old.zip");
```

C1ZipEntryCollection クラス

Zip ファイルを作成するか開いたら、**Entries** コレクションを使用し、Zip ファイルの内容を調べたり、エントリを追加／展開／削除したりできます。たとえば、以下のように記述します。

Visual Basic コードの書き方

Visual Basic

```
myZip.Entries.Add("MyData.txt")
myZip.Entries.Add("MyData.xml")
myZip.Entries.Add("MyData.doc")
Dim zipEntry As C1ZipEntry
ForEach zipEntry In  myZip.Entries
   Console.WriteLine(zipEntry.FileName)
Next zipEntry
```

C# コードの書き方

C#

```
myZip.Entries.Add("MyData.txt");
myZip.Entries.Add("MyData.xml");
myZip.Entries.Add("MyData.doc");
foreach (C1ZipEntry zipEntry in myZip.Entries)
 Console.WriteLine(zipEntry.FileName);
```

C1ZipEntry クラス

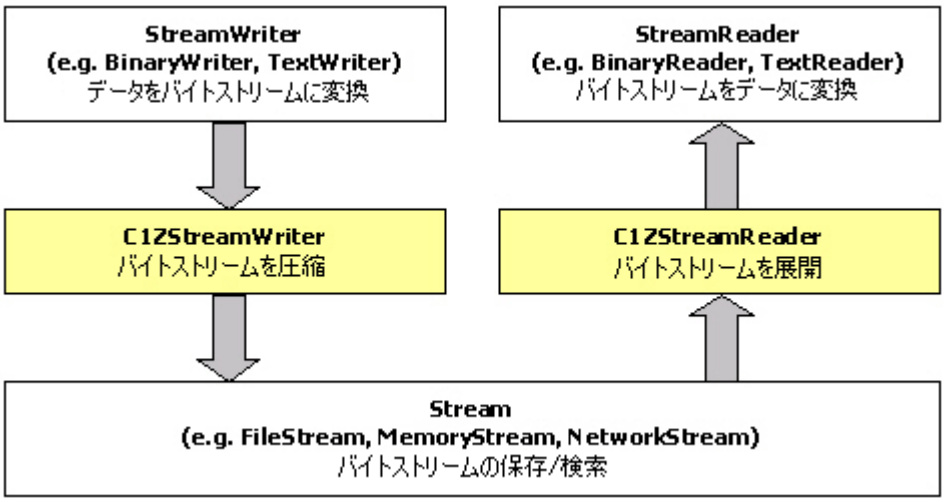
C1ZipEntry クラスは、元のファイルの名前、サイズ、圧縮サイズなどの各エントリを記述するプロパティとメソッドを提供します。また、ストリームオブジェクトを返す **OpenReader** メソッドも持っているため、先に展開せずにエントリの内容を読み取ることができます。

C1ZStreamReader クラスと C1ZStreamWriter クラス

これらのクラスでは、Zip ファイル内のデータはもちろん、あらゆる .NET ストリーム上でデータ圧縮を使用できます。

C1ZStreamReaderオブジェクトと **C1ZStreamWriter** オブジェクトを使用するには、これらのオブジェクトを通常のストリームに付加し、これらを介してデータの読み取りと書き込みを行います。データは基底のストリームに対してオンザフライで圧縮(または展開)されます。

この設計により、ネイティブの .NET ストリームとうまく統合できます。その仕組みは以下の図のようになります。



たとえば、以下のコードでは ADO.NET **DataTable** オブジェクトをストリームに保存してから、それを再度読み取っています。

Visual Basic コードの書き方

Visual Basic

```
' DataTable を圧縮ストリームに保存します。
Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter()
Dim fout As New FileStream("test.tmp", FileMode.Create)
bf.Serialize(fout, DataTableOut)
fout.Close()
' 圧縮データを読み取ります。
Dim fin As New FileStream("test.tmp", FileMode.Open)
Dim DataTableIn As DataTable = bf.Deserialize(fin)
```

C# コードの書き方

C#

```
//DataTable を圧縮ストリームに保存します。
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter bf = new
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter()
FileStream fout = new FileStream("test.tmp", FileMode.Create);
bf.Serialize(fout, DataTableOut);
fout.Close();
//圧縮データを読み取ります。
FileStream fin = new FileStream("test.tmp", FileMode.Open);
DataTable DataTableIn = (DataTable)bf.Deserialize(fin)
```

データ圧縮を追加するには、以下の2行のコードを追加するだけです。

Visual Basic コードの書き方

Visual Basic

```
'DataTable を圧縮ストリームに保存します。
Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter()
Dim fout As New FileStream("test.tmp", FileMode.Create)
Dim compressor As New C1ZStreamWriter(fout)
bf.Serialize(compressor, DataTableOut)
fout.Close()
```

```
'圧縮データを読み取ります。
Dim fin As New FileStream("test.tmp", FileMode.Open)
Dim decompressor As New C1ZStreamReader(fin)
Dim DataTableIn As DataTable = bf.Deserialize(decompressor)
```

C# コードの書き方

C#

```
//DataTable を圧縮ストリームに保存します。
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter bf = new
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
FileStream fout = new FileStream("test.tmp", FileMode.Create);
C1ZStreamWriter compressor = new C1ZStreamWriter(fout);
bf.Serialize(compressor, DataTableOut);
fout.Close();
//圧縮データを読み取ります。
FileStream fin = new FileStream("test.tmp", FileMode.Open);
C1ZStreamReader decompressor = new C1ZStreamReader(fin);
DataTable DataTableIn = (DataTable)bf.Deserialize(decompressor)
```

ZStream クラス

これは、**C1Zip** ライブラリの最下位クラスで、前述の上位クラスで広く使用されます。

大部分のユーザーは、**ZStream** を直接使用する必要はありません。このクラスは、**C1Zip** ライブラリの中で最も柔軟性が高いと同時に、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、オープンソースライブラリで、Jean-loup Gailly と Mark Adlerによって作成されました

タスク別ヘルプ

タスク別ヘルプは、Visual Studio のプログラミングに習熟していることを前提にしています。はじめて Zip for.Net チュートリアル を使用する場合、まず、[ZIP for .NET の使い方]を参照してください。

次のトピックでは、C1.C1Zip名前空間を使用する個々のタスクのソリューションを提供しています。また、各タスク別ヘルプトピックでは、新規 .NET プロジェクトが作成されていることを前提にしています。

データセットの圧縮

データを ZIP ファイルに圧縮するには、以下のコードを使用します。

Visual Basic コードの書き方

Visual Basic

```
Private Sub SaveDataSet(ds As DataSet)
  'ZIP ファイルを開くまたは作成します。
   Dim zip As New C1.C1Zip.C1ZipFile()
   zip.Open("c:\temp\dataset.zip")
   'データセットを ZIP ファイルに書き込みます。
    Dim s As Stream = zip.Entries.OpenWriter(ds.DataSetName, True)
    Try
     ds.WriteXml(s, XmlWriteMode.WriteSchema)
   Finally
     s.Dispose()
  End Try
End Sub
Private Sub CheckDataSet(ds As DataSet)
   'ZIP ファイルを開くまたは作成します。
    Dim zip As New C1.C1Zip.C1ZipFile()
    zip.Open("c:\temp\dataset.zip")
    'データセットを ZIP ファイルから読み込みます。
    Dim dsTest As New DataSet(ds.DataSetName)
    Dim s As Stream = zip.Entries(ds.DataSetName).OpenReader()
    Try
      dsTest.ReadXml(s)
    Finally
      s.Dispose()
    End Try
    'データセットが同じであるかどうか確認します。
    For i = 0 To ds.Tables.Count - 1
    Dim dt1 As DataTable = ds.Tables(i)
      Dim dt1 As DataTable = ds.Tables(i)
      Dim dt2 As DataTable = dsTest.Tables(i)
      Debug.Assert((dt1.TableName = dt2.TableName And dt1.Columns.Count =
dt2.Columns.Count And dt1.Rows.Count = dt2.Rows.Count))   Next i
End Sub
```

C# コードの書き方

C#

```
private void SaveDataSet(DataSet ds)
{
    //ZIP ファイルを開くまたは作成します。
    C1.C1Zip.C1ZipFile zip = new C1.C1Zip.C1ZipFile();
    zip.Open(@"c:\temp\dataset.zip");
   //データセットを ZIP ファイルに書き込みます。
    using (Stream s = zip.Entries.OpenWriter(ds.DataSetName, true))
   {
     ds.WriteXml(s, XmlWriteMode.WriteSchema);
   }
}
private void CheckDataSet(DataSet ds)
{
    //ZIP ファイルを開くまたは作成します。
    C1.C1Zip.C1ZipFile zip = new C1.C1Zip.C1ZipFile();
    zip.Open(@"c:\temp\dataset.zip");
    //データセットを ZIP ファイルから読み込みます。
    DataSet dsTest = new DataSet(ds.DataSetName);
    using (Stream s = zip.Entries[ds.DataSetName].OpenReader())
    {
      dsTest.ReadXml(s);
    }
    // データセットが同じであるかどうか確認します。
    for (int i = 0; i < ds.Tables.Count; i++)
    {
        DataTable dt1 = ds.Tables[i];
        DataTable dt2 = dsTest.Tables[i];
        System.Diagnostics.Debug.Assert(dt1.TableName == dt2.TableName
dt1.Columns.Count == dt2.Columns.Count  dt1.Rows.Count ==
dt2.Rows.Count)
    }
}
```

すべてのフォルダを Zip ファイルに圧縮する

AddFolder メソッドを使用して、フォルダの構成を変更せずに、フォルダ全体を一つの ZIP ファイルに圧縮/展開できます。

例えば、[c:\temp] フォルダの内容をサブフォルダも含めて圧縮するには、以下のコードを使用します。

Visual Basic コードの書き方

Visual Basic

```
C1Zip.Entries.AddFolder("c:\temp\", "*.*", True)
```

C# コードの書き方

C#

```
C1Zip.Entries.AddFolder(@"c:\temp\", "*.*", true);
```

元のフォルダ構成を保ったままフォルダを展開するには、以下のコードを使用します。

Visual Basic コードの書き方

Visual Basic

```
C1Zip.Entries.ExtractFolder("c:\temp\")
```

C# コードの書き方

C#

```
C1Zip.Entries.ExtractFolder(@"c:\temp\");
```

複数のエントリを持つ Zip ファイルの作成

複数の XML ファイルを直接 ZIP ファイルに書き込むストリームオブジェクトを開くには、**OpenWriter** メソッドを使用します。

zip ファイルにデータを追加するために使用できるストリームが戻り値です。エントリは、ストリームが閉じられるまで追加されません。
以下のコードを、**Click** イベントに追加します。

Visual Basic コードの書き方

Visual Basic

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
  Dim zip As New C1ZipFile()
  zip.Create("c:\temp\test.zip")
  Dim s As Stream = zip.Entries.OpenWriter("entry1", True)
  Dim sw As New StreamWriter(s)
  sw.WriteLine("Hello world")
  '希望するだけ書き込みます。
   sw.Close()
   s = zip.Entries.OpenWriter("entry2", True)
   sw = New StreamWriter(s)
   sw.WriteLine("Hello again")
   '希望するだけ書き込みます。
   sw.Close()
End Sub
```

C# コードの書き方

C#

```
private void button1_Click(object sender, System.EventArgs e)
{
    C1ZipFile zip = new C1ZipFile();
    zip.Create(@"c:\temp\test.zip");
    Stream s = zip.Entries.OpenWriter("entry1", true);
    StreamWriter sw = new StreamWriter(s);
    sw.WriteLine("Hello world");
      // 希望するだけ書き込みます。
    sw.Close();
    s = zip.Entries.OpenWriter("entry2", true);
    sw = new StreamWriter(s);
    sw.WriteLine("Hello again");
      // 希望するだけ書き込みます。
```

```
    sw.Close();
}
```

エントリをファイルに保存せずに読み込むには、**OpenReader** メソッドを使用します。

> **注意**: OpenWriter メソッドは、**C1ZipEntryCollection** クラスのメンバですが、OpenReader メソッドは、**C1ZipEntry** クラスのメンバです。

Zip エントリからファイルをメモリ上に展開する

ZIP ファイルからメモリ（例 Byte 配列）にファイルを抽出するには、以下の関数を使用します。

Visual Basic コードの書き方

Visual Basic

```
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)
     'エントリの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()eturn ms.ToArray()
End Function
```

C# コードの書き方

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];
       // エントリの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);
        }
    }
      // C# で using ステートメントを使用しているため、Close メソッドを呼び出す必要はありません。
       // しかし、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込む

StreamReader を使用して ZIP ファイルを読み込むには、以下のコードを追加します。

Visual Basic コードの書き方

Visual Basic

```
 'ZIP ファイルを開きます。
Dim zip As New C1ZipFile()
zip.Open("c:\temp\myzipfile.zip")
 '任意のエントリで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()
 ' StreamReader をストリームで開きます。
Dim sr As New StreamReader(s)
 ' StreamReader を使用して閉じます.
```

C# コードの書き方

C#

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

Zip ファイルからイメージを検索する

ZIP ファイルから直接画像を取り出すには、以下のコードを **Form_Load** イベントに追加して複数の画像ファイルを ZIP ファイ

ルに圧縮します。

Visual Basic コードの書き方

Visual Basic

```
 'リソースディレクトリに画像一覧を作成し、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"))
     ' 画像をZIPファイルとリストボックスに追加します。
    Dim f As String
    For Each f In  Directory.GetFiles(s)
        Dim fname As String = f.ToLower()
         'zip ファイルをスキップします。
        If fname.EndsWith("zip") Then
          GoTo ContinueForEach1
        End If
         ' 画像一覧をリストボックスに追加します。
        ListBox1.Items.Add(Path.GetFileName(fname))
         ' 画像をZIPファイルに追加します。
        zip.Entries.Add(fname)
        ContinueForEach1:
    Next f
End Sub
```

C# コードの書き方

C#

```
 //リソースディレクトリに画像一覧を作成し、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");
    //画像をZIPファイルとリストボックスに追加します。
    foreach (string f in Directory.GetFiles(s))
    {
        string fname = f.ToLower();
    // ZIPファイルをスキップします。
        if (fname.EndsWith("zip")) continue;
    //画像一覧をリストボックスに追加します。
        listBox1.Items.Add(Path.GetFileName(fname));
    //画像をZIPファイルに追加します。
     zip.Entries.Add(fname);
```

```
    }
}
```

画像を選択し、画像データ格納しているストリームを取得し（**OpenReader** メソッド）、画像を読み込む（Image.**FromStream** メソッド）には、以下のコードを **SelectedIndexChanged** イベントに追加します。

Visual Basic コードの書き方

Visual Basic

```
 ' 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)
    ' 圧縮ストリームから画像を直接読み込みます。
    Dim s As Stream = zip.Entries(item).OpenReader()
    Try
        pictureBox1.Image = CType(Image.FromStream(s), Image)
    Catch
    End Try
     ' ストリームを終了します。
    s.Close()
End Sub
```

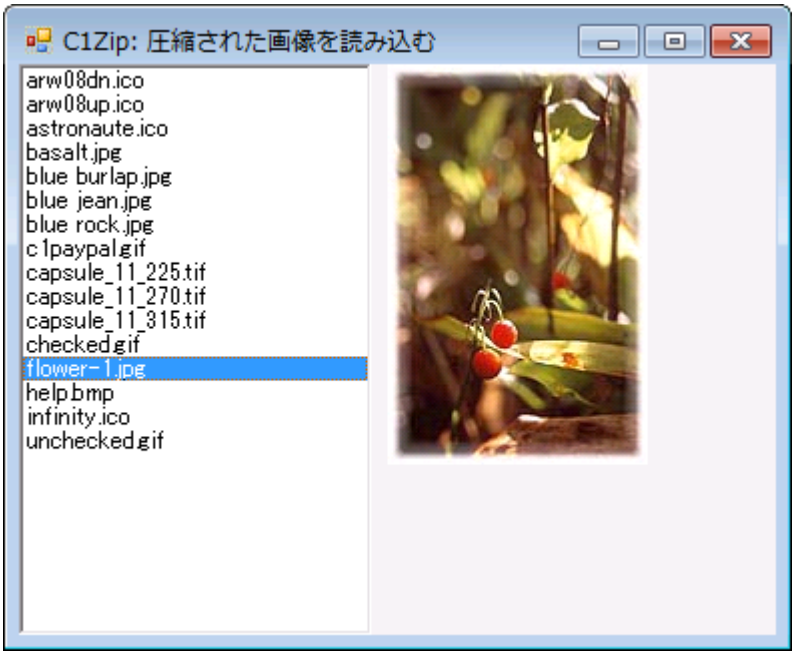
C# コードの書き方

C#

```
// 選択された画像を表示します。
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs eventArgs
e)
{
     // 選択された項目を取得します。
    string item = (string)listBox1.SelectedItem
     // 圧縮ストリームから画像を直接読み込みます。
    Stream s = zip.Entries[item].OpenReader()
    try
    {
        pictureBox1.Image = (Image)Image.FromStream(s)
    }
    catch {}
     // ストリームを終了します。
    s.Close();
}
```

コードの結果

下図には、ICO、GIF、TIFF、BMP や JPG など複数の種類の画像は表示されます。



Zip ファイルに文字列変数を保存する

文字列を ZIP ファイルに保存するには、以下のいずれかのメソッドを使用します。

- **OpenWriter**メソッド
  OpenWriterメソッドを使用してストリームを取得します。文字列を書き込んで閉じます。ストリームに書き込むとデータが圧縮され、ストリームを閉じる時このストリーム全体が 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んで、このストリームを ZIP ファイルに追加します。

注意：**MemoryStream** は、OpenWriter メソッドより複雑ですが、とても使いやすいメソッドです。

上記二つのメソッドを以下のコードに示します。そのコードを、**Button_Click**イベントに追加します。

Visual Basic コードの書き方

Visual Basic

```
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: OpenWrite
    Dim stream As Stream = zipFile.Entries.OpenWriter("Shakespeare.txt", True)
    Dim sw As New StreamWriter(stream)
    sw.Write(str)
    sw.Close()
     ' 方法 2: MemoryStream
    Dim Stream As New MemoryStream()
    Dim sw As New StreamWriter(Stream)
    sw.Write(str)
    sw.Flush()
    Stream.Position = 0
```

```
    zipFile.Entries.Add(Stream, "Shakespeare2.txt"
    Stream.Close()
End Sub
```

C# コードの書き方

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
    Stream stream = zipFile.Entries.OpenWriter("Shakespeare.txt", true);
    StreamWriter sw = new StreamWriter(stream);
    sw.Write(str)
    sw.Close()
     // 方法 2: MemoryStream
    Stream stream = new MemoryStream()
    StreamWriter sw = new StreamWriter(stream);
    sw.Write(str);
    sw.Flush();
    stream.Position = 0;
    zipFile.Entries.Add(stream, "Shakespeare2.txt");
    stream.Close()
}
```

圧縮レベルの設定

圧縮されたファイルのサイズを小さくするには、以下のコードを使用して、C1ZStreamWriter のコンストラクタに圧縮レベルを設定します。

Visual Basic コードの書き方

Visual Basic

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

C# コードの書き方

C#

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

注意: 上記のサンプルコードは、圧縮レベルを **BestCompression** に設定しています。ファイルサイズは最小になりますが、処理にかかる時間は長くなります。圧縮レベルを設定する他のオプションは以下のとおりです:

- **BestSpeed** は、圧縮時間を最小化します。
- **DefaultCompression** は、標準の圧縮時間と速度です。
- **NoCompression** は、圧縮しません。

Zip ファイルの保護のためにパスワードを使用する

Zip ファイルをパスワードで保護するには、エントリを作成する前に **Password**プロパティに任意の文字列を設定します。以下のように各エントリは、それぞれ独自のパスワードを設定できます。

Visual Basic コードの書き方

Visual Basic

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

C# コードの書き方

C#

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

このエントリを抽出するには、Password プロパティにエントリを追加したときと同じ値を設定する必要があります。例えば:

Visual Basic コードの書き方

Visual Basic

```
' パスワードを設定しないい場合は、ファイルが開きません。
C1Zip.Password = ""
C1Zip.Entries.Extract(someFile)
' 不正なパスワードを設定する場合は、ファイルが開きません
C1Zip.Password = "pass"
C1Zip.Entries.Extract(someFile)
 'ファイルが開きます。
C1Zip.Password = "password"
C1Zip.Entries.Extract(someFile);
```

C# コードの書き方

C#

```
// パスワードを設定しない場合は、ファイルが開きません。
C1Zip.Password = ""
C1Zip.Entries.Extract(someFile)
// 不正なパスワードを設定する場合は、ファイルが開きません。
C1Zip.Password = "pass"
C1Zip.Entries.Extract(someFile)
// ファイルが開きます。
C1Zip.Password = "password"
C1Zip.Entries.Extract(someFile)
```

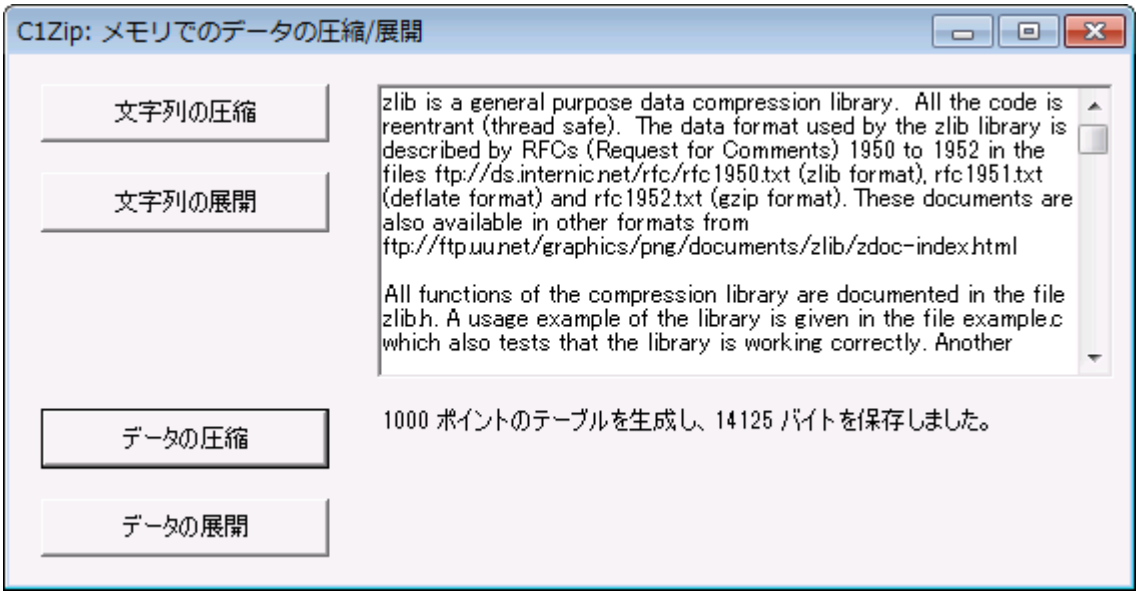
Zip for.Netチュートリアル

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 .NET 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。
Zip ファイルの操作	zip アーカイブにあるファイルを開いたり、ファイルを検査、追加、および削除する方法を示します。アプリケーションストレージの目的で zip 形式を使用することには、いくつかの利点があります。zip 形式は十分な解説文書があるオープンスタンダードで、空間効率に優れています。

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。

新しい Visual Studio プロジェクトを開始します。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。

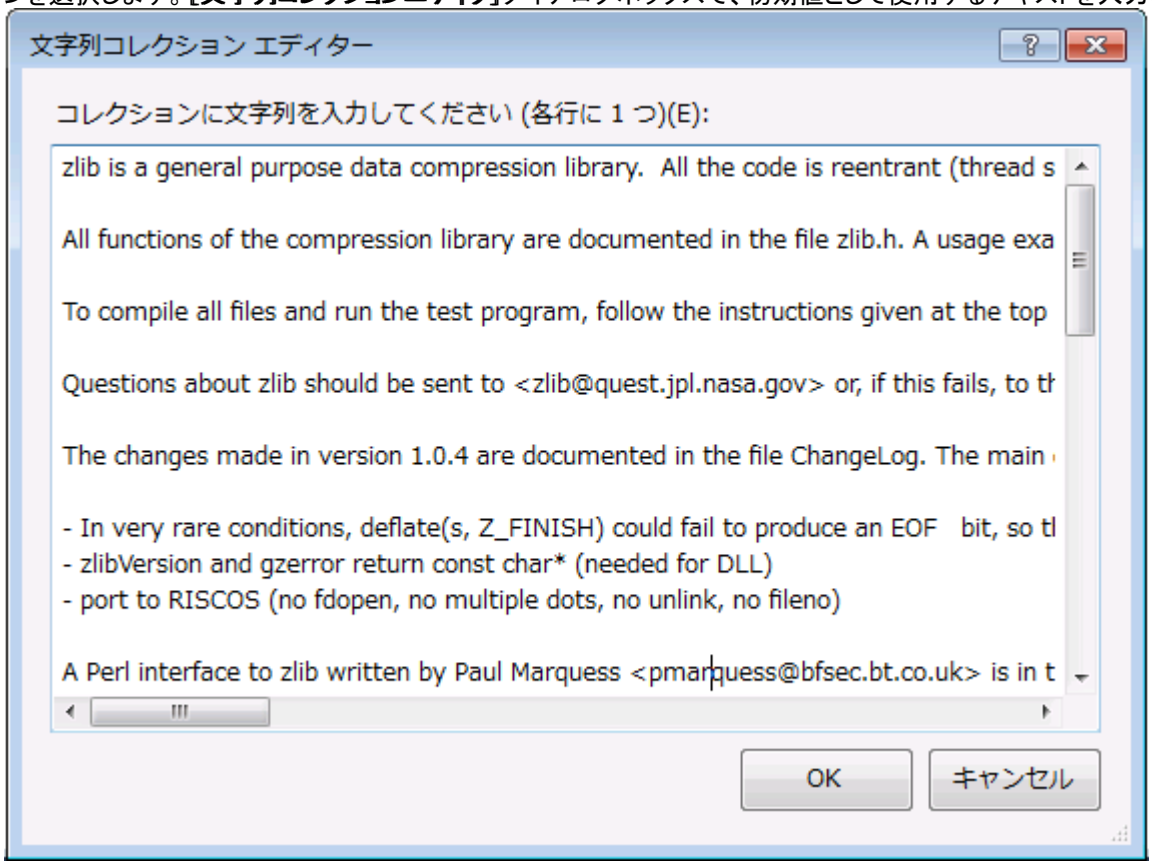
- フォームの左端に並べて4個の **Button** コントロール（前の図を参照）。[プロパティ]ウィンドウで、各 **Button** コント

ロールに次の変更を加えます。

ボタン	Button.Text プロパティ	Button.Name プロパティ	Button.Enabled プロパティ
1	文字列の圧縮	**btnCompressString**	True（デフォルト）
2	文字列の展開	**btnExpandString**	**False**
3	データの圧縮	**btnCompressData**	True（デフォルト）
4	データの展開	**btnExpandData**	**False**

メモ: **[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 **TextBox**。**MultiLine** プロパティを "**True**" に設定します。**Lines** プロパティの横にある**省略符**ボタンを選択します。**[文字列コレクションエディタ]**ダイアログボックスで、初期値として使用するテキストを入力します。



- テキストボックスの下に **Label** コントロール。

手順2：C1Zip 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.C1Zip アセンブリを選択するか、ファイルを参照して C1.C1Zip.2.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。

Visual Basic でコードを書く場合

Visual Basic

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

C# でコードを書く場合

C#

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

これで、C1Zip 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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

C#

```
private byte[] m_CompressedString;
private void btnCompressString_Click(object sender, EventArgs e)
{
     // 文字列を圧縮します。
     long ticks = 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。
    int ms = (int)((DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond);
    int lenBefore = textBox1.Text.Length * 2;
    int lenAfter = m_CompressedString.Length;
    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;
}
```

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

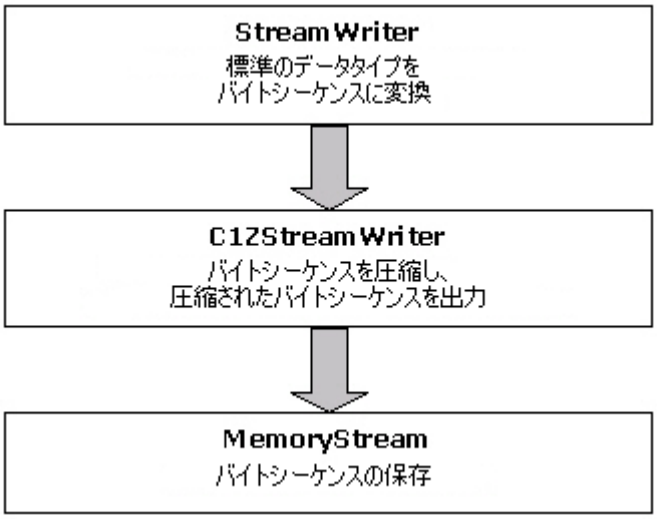
C#

```
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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

C#

```
private void btnExpandString_Click(object sender, EventArgs e)
```

```
{

    // 文字列を展開します。
    long ticks = 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。
    int ms = (int)((DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond);
    int lenBefore = m_CompressedString.Length;
    int lenAfter = textBox1.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", MessageBoxButtons.OK,
MessageBoxIcon.Information);
}
```

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。
    C1ZStr
    // 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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

C#

```
private void btnCompressData_Click(object sender, EventArgs 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。
    m_CompressedData = ms.ToArray();
      // 完了。
    string  msg;
    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;
}
```

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

C#

```
private void btnExpandData_Click(object sender, EventArgs 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(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。
    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, m_CompressedData.Length);
    label1.Text = msg;
}
```

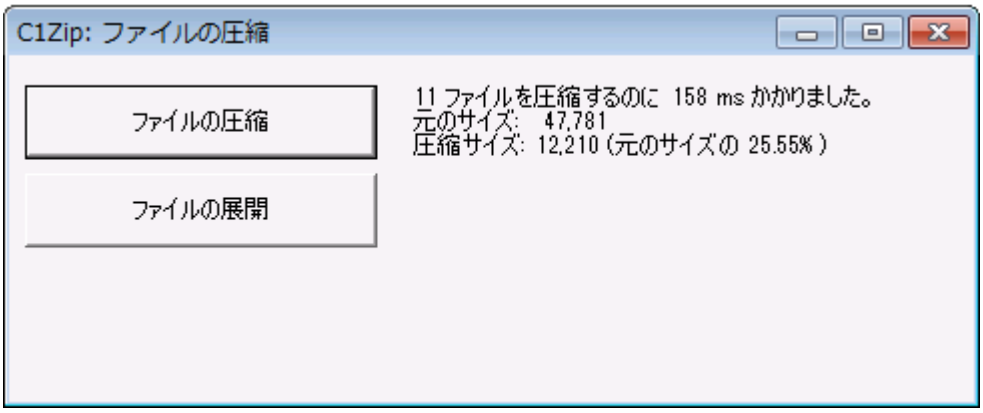
このコードでは、データは読み取られますが、画面には表示されません。コードをデバッグモードでステップ実行すると、読み取られたデータが書き込まれたデータと同じであることを確認できます。

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。Zip ファイルについては、「Zip ファイルの操作」チュートリアルで扱います。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。

新しい Visual Studio プロジェクトを開始し、ツールボックスから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2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。

ボタン	Button.Text プロパティ	Button.Name プロパティ	Button.Enabled プロパティ
1	ファイルの圧縮	**btnCompress**	True(デフォルト)
2	ファイルの展開	**btnExpand**	**False**

  [ファイルの展開]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。

- 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:C1Zip 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.C1Zip アセンブリを選択するか、ファイルを参照して C1.C1Zip.2.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。

Visual Basic でコードを書く場合

Visual Basic

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

C# でコードを書く場合

C#

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

これで、**C1Zip** 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。

Visual Basic でコードを書く場合

Visual Basic

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

C# でコードを書く場合

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。

最後に、両方のストリームを閉じます。**Finally** ステートメントを使用することで、関数の実行中に例外が発生した場合でも、両方のストリームが正しく閉じられます。

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

C#

```
private bool ExpandFile(string dstFile, string srcFile)
{
     // 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。

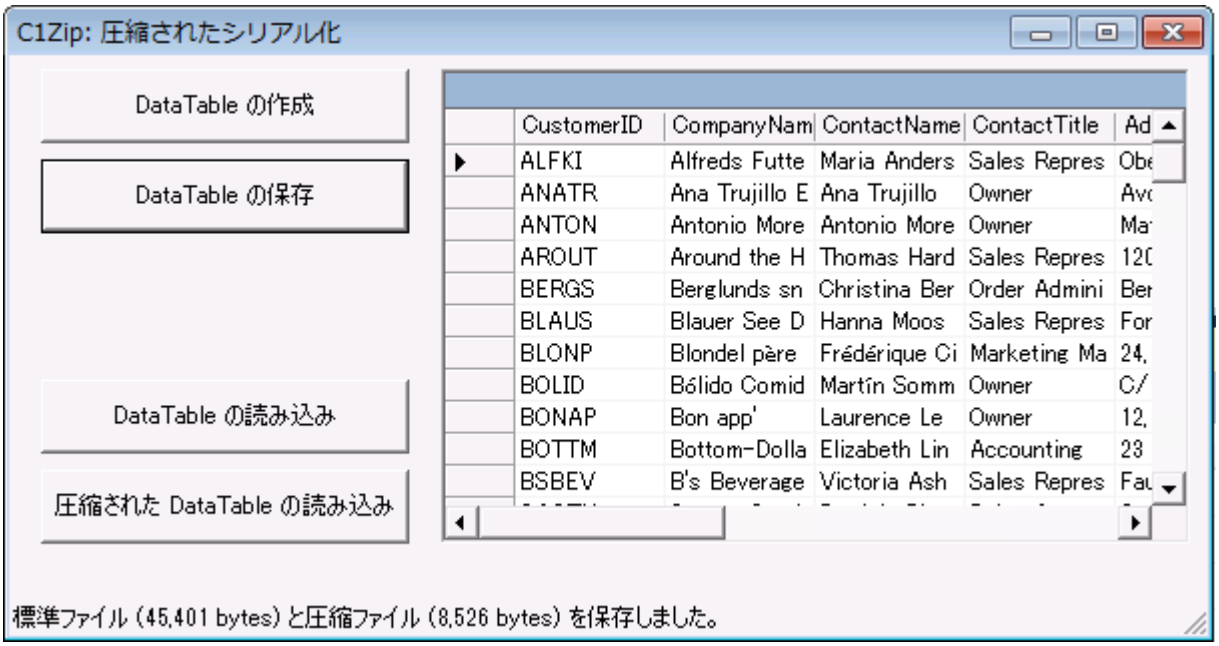
これで「ファイルの圧縮」チュートリアルは終了です。

圧縮されたシリアル化

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。

この例では、NorthWind データベースを使用して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

新しい Visual Studio プロジェクトを開始し、ツールボックスから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、各 **Button** コントロールに次の変更を加えます。

ボタン	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.C1Zip アセンブリを選択するか、ファイルを参照して C1.C1Zip.2.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。

Visual Basic でコードを書く場合

```
Visual Basic
```

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

C# でコードを書く場合

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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Const FN_REGULAR = "\DataTable.regular"
Private Const FN_COMPRESSED = "\DataTable.compressed"
Private Const MDBFILE = " C:\Users\Documents\ComponentOne Samples\Common\C1NWIND.MDB"
```

C# でコードを書く場合

C#

```
private const string FN_REGULAR = @"\DataTable.regular";
private const string FN_COMPRESSED = @"\DataTable.compressed";
private const string MDBFILE = @"C:\Users\Documents\ComponentOne
Samples\Common\C1NWIND.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

[**データテーブルの作成**]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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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

[**データテーブルのロード**]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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。

[**圧縮データテーブルのロード**]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。

Visual Basic でコードを書く場合

Visual Basic

```
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# でコードを書く場合

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。

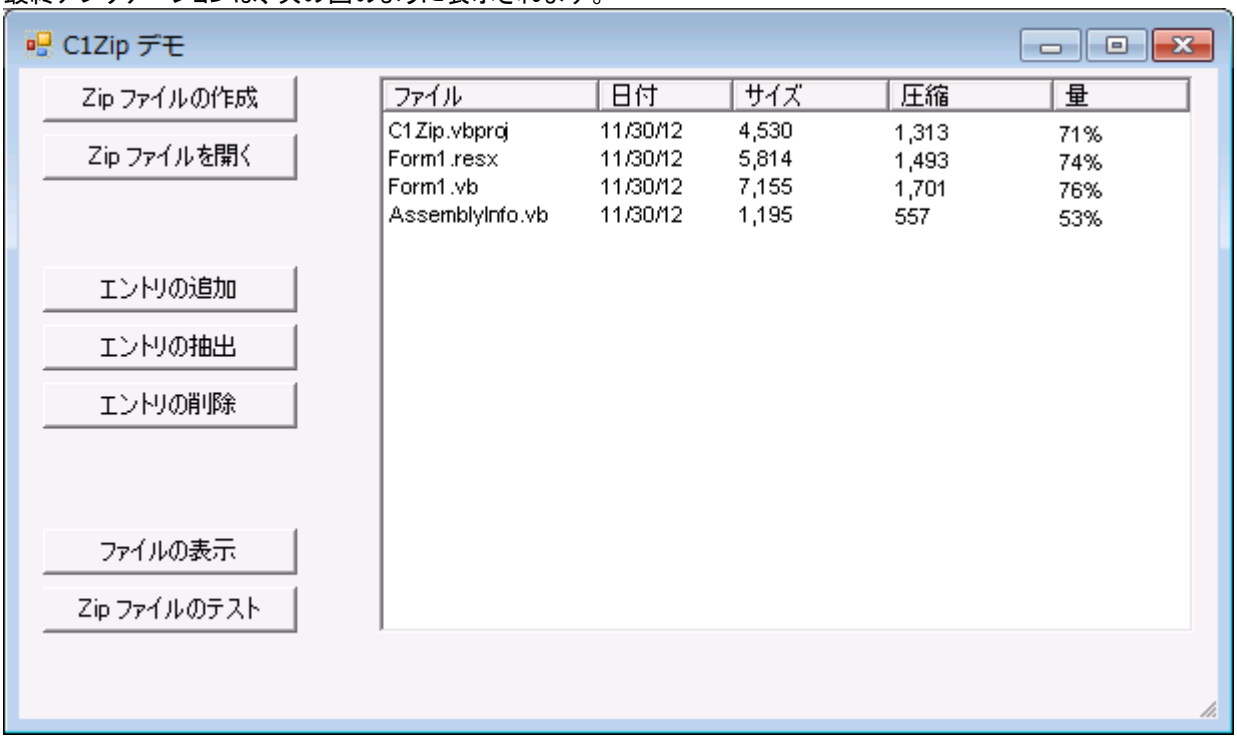
これで、「圧縮シリアライズ」チュートリアルは終了です。

Zip ファイルの操作

このチュートリアルでは、次のような zip ファイルの操作方法について説明します。

- zip ファイルの作成。
- 既存の zip ファイルを開く。
- zip ファイルへのエントリの追加。
- zip ファイルからのエントリの抽出。
- zip ファイルからのエントリの削除。
- メモリストリームへのエントリの展開。
- zip ファイルの整合性のテスト。

最終アプリケーションは、次の図のように表示されます。



メモ: これは、**C1ZipFile** オブジェクトの主要な機能に注目して設計された極めて単純なアプリケーションです。配布ディスクには、より高度なバージョンも用意されています。そこでは、ドラッグ &ドロップ操作、フォルダの zip 圧縮、圧縮レベルの設定などの高度な機能がサポートされています。

手順1:メインフォームを作成します。

新しい Visual Studio プロジェクトを開始します。

1. ツールボックスから、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7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。

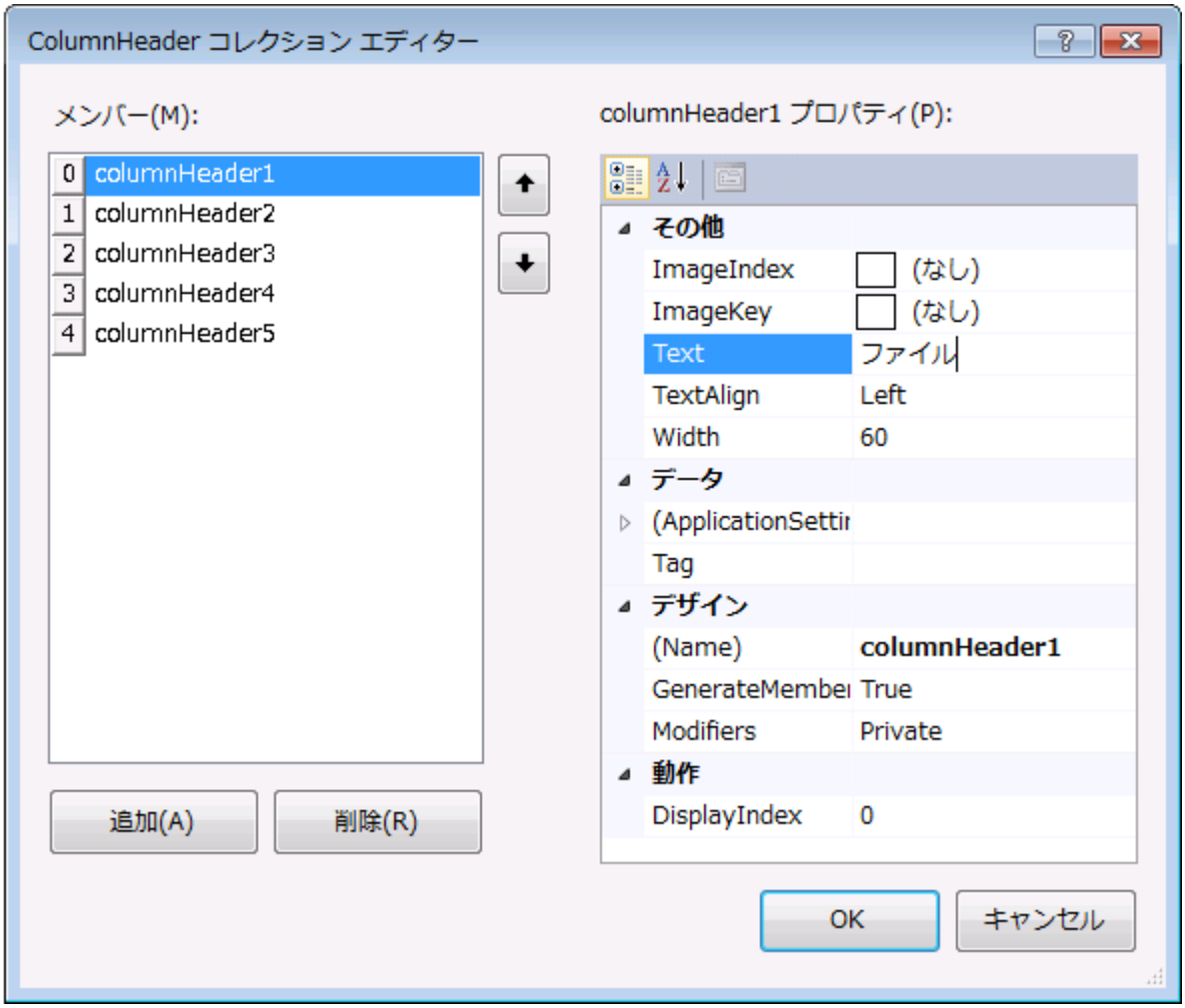
Button	Button.Text Property	Button.Name Property
1	Create Zip File...	**btnNew**
2	Open Zip File...	**btnOpen**
3	Add Files...	**btnAdd**
4	Extract Files	**btnExtract**
5	Remove Files	**btnRemove**
6	View File	**btnView**
7	Test Zip File	**btnTest**

   - フォームの右側をカバーする ListView コントロール。プロパティウィンドウで、Columns プロパティの横にある省略符ボタンをクリックします。[ColumnHeader コレクションエディタ]ダイアログボックスが表示されます。

メモ: [ColumnHeader コレクションエディタ]には、[ListView タスク]メニューの[列の編集]をクリックしてアクセスすることもできます。[ListView タスク]メニューにアクセスするには、ListView コントロールの右上隅にあるスマートタグ をクリックします。

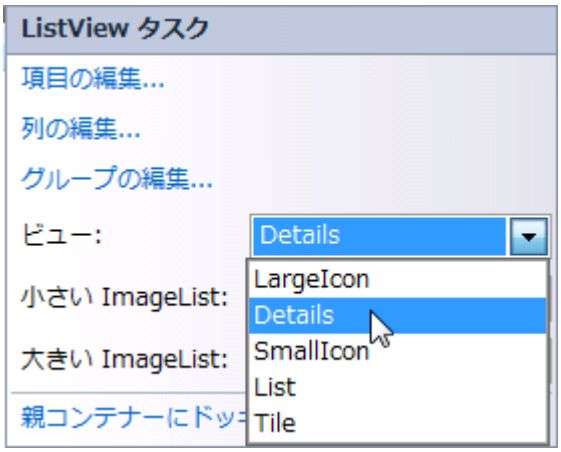
2. [追加]ボタンをクリックし、5つの ColumnHeader を追加したら、各列の Text プロパティをそれぞれ[ファイル]、[日付]、[サイズ]、[圧縮]、および[量]に設定します。

[ColumnHeader コレクションエディタ]は次のようになります。



3. [OK]を選択して、エディタボックスを閉じます。

列ヘッダを表示するには、[プロパティ]ウィンドウで[表示]プロパティを[詳細]に設定するか、[ListView タスク]メニューの[表示]ドロップダウンボックスから[詳細]を選択します。



手順2:C1Zip 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、[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.C1Zip アセンブリを選択するか、ファイルを参照して C1.C1Zip.2.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、次のディレクティブを追加します。

Visual Basic でコードを書く場合

Visual Basic
```
Imports System.IO
Imports C1.C1Zip
```

C# でコードを書く場合

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

これで、C1Zip 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に表示されます。

手順3:C1ZipFile オブジェクトを宣言します。

コードエディタに切り替えて、次のデータメンバ宣言を入力するかコピーします。

Visual Basic でコードを書く場合

Visual Basic
```
Private m_Zip As C1ZipFile
```

C# でコードを書く場合

C#
```
private m_Zip As C1ZipFile;
```

これは、このアプリケーションのメインオブジェクトです。このオブジェクトは、zip ファイルの処理に使用されるメソッドを実装しています。

手順4:C1ZipFile オブジェクトを初期化するコードを追加します。

Form_Load イベントに次のコードを追加します。

Visual Basic でコードを書く場合

Visual Basic
```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
     ' C1ZipFile メンバを作成します。
    m_Zip = New C1ZipFile()
End Sub
```

C# でコードを書く場合

C#
```
private void Form1_Load(object sender, EventArgs e)
{
  // C1ZipFile メンバを作成します。
    m_Zip = new C1ZipFile();
```

```
}
```

このコードは、新しい C1ZipFile オブジェクトを作成し、それを m_Zip メンバに代入します。この C1ZipFile オブジェクトはまだ使用できないことに注意してください。それには、最初に Open メソッドを使用して既存の zip ファイルにアタッチするか、Create メソッドを使用して新しい zip ファイルにアタッチする必要があります。

手順5:zip ファイルを作成するコードを追加します。

[Zip ファイルの作成]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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub btnNew_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnNew.Click
     ' ファイルを開くダイアログを表示します。
    Dim fo As SaveFileDialog = New SaveFileDialog()
    fo.FileName = "*.zip"
    If fo.ShowDialog() <> Windows.Forms.DialogResult.OK Then
        Exit Sub
    End If
     ' zip ファイルを開きます。
    Try
        m_Zip.Create(fo.FileName)
    Catch
        MessageBox.Show("Can't create ZIP file, please try again.", "C1Zip")
    End Try
     ' UI を更新します。
    UpdateDisplay()
End Sub
```

C# でコードを書く場合

C#

```
private void btnNew_Click(object sender, EventArgs e)
{
    // ファイルを開くダイアログを表示します。
    SaveFileDialog fo = new SaveFileDialog();
    fo.FileName = "*.zip";
    if (fo.ShowDialog() != DialogResult.OK ) return;
     // zip ファイルを開きます。
    try
    {
        m_Zip.Create(fo.FileName);
    }
    catch
    {
        MessageBox.Show("Can't create ZIP file, please try again.", "C1Zip");
    }
     // UI を更新します。
    UpdateDisplay();
}
```

手順6:zip ファイルを開くコードを追加します。

[Zip ファイルを開く]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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub btnOpen_Click(sender As Object, e As EventArgs) Handles btnOpen.Click
     ' ファイルを開くダイアログを表示します。
    Dim fo As OpenFileDialog = New OpenFileDialog()
    fo.FileName = "*.zip"
    If fo.ShowDialog() <> Windows.Forms.DialogResult.OK Then Exit Sub
    Try
        m_Zip.Open(fo.FileName)
    Catch
        MessageBox.Show("Invalid ZIP file, please try again.")
    End Try
      ' UI を更新します。
    UpdateDisplay()
End Sub
```

C# でコードを書く場合

C#

```
private void btnOpen_Click(object sender, EventArgs e)
{
     // ファイルを開くダイアログを表示します。
    OpenFileDialog fo = new OpenFileDialog();
    fo.FileName = "*.zip";
    if (fo.ShowDialog() != DialogResult.OK ) return;
    try
    {
        m_Zip.Open(fo.FileName);
    }
    catch
    {
        MessageBox.Show("Invalid ZIP file, please try again.");
    }
     // UI を更新します。
    UpdateDisplay();
}
```

重要な行は、C1ZipFile オブジェクトの Open メソッドの呼び出しです。ここで、オブジェクトを既存の zip ファイルにアタッチします。**Try/Catch** ステートメントは、主に、ユーザーが zip ファイルでないファイルを選択した場合に対処するために使用されます。

ファイルが開いたら、**UpdateDisplay** ユーティリティ関数を呼び出して、zip ファイルの内容を表示します。

手順7: 表示を更新するコードを追加します。

次に、zip ファイルの内容を表示する **UpdateDisplay** ユーティリティ関数を追加します。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub UpdateDisplay()
      ' ListView コントロールを更新して、zip ファイルの内容を表示します。
    With ListView1
          ' 既存の項目を削除します。
        .Items.Clear()
         ' 各エントリを追加します。
        Dim ze As C1ZipEntry
        For Each ze In m_Zip.Entries
            ' 圧縮量を計算します。
            Dim pct As Double = 0
            If ze.SizeUncompressed > 0 Then
                pct = 1 - ze.SizeCompressed / ze.SizeUncompressed
            End If
             ' ListView 項目を構築します。
            Dim items(4) As String
            items(0) = ze.FileName
            items(1) = Format(ze.Date, "MM/dd/yy")
            items(2) = Format(ze.SizeUncompressed, "#,##0")
            items(3) = Format(ze.SizeCompressed, "#,##0")
            items(4) = Format(pct, "00 %")
            Dim lvi As ListViewItem = New ListViewItem(items)
              ' ZipEntry を項目タグに保存します。
            lvi.Tag = ze
              ' 項目を ListView に追加します。
            .Items.Add(lvi)
        Next ze
         ' UI を更新します。
        Dim hasEntries As Boolean = (.Items.Count > 0)
        btnExtract.Enabled = hasEntries
        btnRemove.Enabled = hasEntries
        btnTest.Enabled = hasEntries
    End With
End Sub
```

C# でコードを書く場合

C#

```
private void UpdateDisplay()
{
     // 既存の項目を削除します。
    listView1.Items.Clear();
     // 各エントリを追加します。
    foreach (C1ZipEntry ze in m_Zip.Entries)
    {
        // 圧縮量を計算します。
        double pct = 0;
        if (ze.SizeUncompressed > 0)
        {
            pct = 1 - (((double)ze.SizeCompressed) / ((double)ze.SizeUncompressed));
        }
        // ListView 項目を構築します。
```

```
        ListViewItem lvi = new ListViewItem(new string[] { ze.FileName,
Microsoft.VisualBasic.Strings.Format(ze.Date, "MM/dd/yy"),
Microsoft.VisualBasic.Strings.Format(ze.SizeUncompressed, "#,##0"),
Microsoft.VisualBasic.Strings.Format(ze.SizeCompressed, "#,##0"),
Microsoft.VisualBasic.Strings.Format(pct, "00 %") });
        // ZipEntry を項目タグに保存します。
        lvi.Tag = ze;
        // 項目を ListView に追加します。
        listView1.Items.Add(lvi);
    }
    // UI を更新します。
    bool hasEntries = (listView1.Items.Count > 0);
    btnExtract.Enabled = hasEntries;
    btnRemove.Enabled = hasEntries;
    btnTest.Enabled = hasEntries;
}
```

重要な行は、最初の C1ZipEntry オブジェクトの宣言と、そのオブジェクトを **ForEach** ループ内で使用して zip ファイル内のエントリ(m_Zip.Entries)を処理するコードです。

各エントリに対して、C1ZipEntry オブジェクトから抽出された情報を格納する **ListViewItem** を作成し、新しいエントリを **ListView** コントロールに追加します。C1ZipEntry オブジェクト自体は ListView 項目の **Tag** プロパティに保存されます。

最後に、抽出、削除、またはテスト可能なエントリがあるかどうかに応じて、コマンドボタンを有効/無効にします。

手順8:zip ファイルの整合性をテストするコードを追加します。

[**Zip ファイルのテスト**]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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub btnTest_Click(sender As Object, e As EventArgs) Handles btnTest.Click
     ' 各エントリをテストします。
    Dim ze As C1ZipEntry
    For Each ze In m_Zip.Entries
        If Not ze.CheckCRC32() Then
            MessageBox.Show("** Entry " & ze.FileName & " has errors.", "C1Zip",
MessageBoxButtons.OK, MessageBoxIcon.Error)
            Exit Sub
        End If
    Next
      ' ここまで到達すれば問題はありません。
    MessageBox.Show("All entries passed CRC check", "C1Zip")
End Sub
```

C# でコードを書く場合

C#

```
private void btnTest_Click(object sender, EventArgs e)
{
      // 各エントリをテストします。
    foreach (C1ZipEntry ze in m_Zip.Entries)
    {
```

```
        if (!ze.CheckCRC32())
        {
            MessageBox.Show("** Entry " + ze.FileName + " has errors.", "C1Zip",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
    }
     // ここまで到達すれば問題はありません。
    MessageBox.Show("All entries passed CRC check", "C1Zip");
}
```

重要な行は、C1ZipEntry 変数の宣言と、その変数を使用して zip ファイル内のエントリをループするコードです。次に、各エントリに対して **CheckCRC32** メソッドを呼び出して、ファイルに保存されているバイトデータの実際のチェックサムがエントリヘッダに保存されているチェックサムと一致するかどうかを確認します。この関数は、値が一致する場合は true を返します。値が一致しない場合は、false を返して、エントリが破損していることを示します。

手順9:エントリの内容を表示するコードを追加します。

[**ファイルの表示**]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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub btnView_Click(sender As Object, e As EventArgs) Handles btnView.Click
      ' 最初に選択されたディレクトリではない項目を取得します。
    Dim ze As C1ZipEntry = Nothing
    Dim lvi As ListViewItem
    For Each lvi In ListView1.SelectedItems
        Dim zeItem As C1ZipEntry = lvi.Tag
        If (zeItem.Attributes And FileAttributes.Directory) = 0 Then
            ze = zeItem
            Exit For
        End If
    Next lvi
      ' 項目を取得したことを確認します。
    If ze Is Nothing Then
        MessageBox.Show("Sorry, no files to show...", "C1Zip")
        Exit Sub
    End If
     ' エントリの内容を文字列に読み取ります。
    Dim entry As Stream = ze.OpenReader()
    Dim sr As StreamReader = New StreamReader(entry)
    Dim entryText As String = sr.ReadToEnd()
    entry.Close()
     ' エントリが MessageBox に対して長すぎないようにします。
    If entryText.Length > 16000 Then
        entryText = entryText.Substring(0, 16000)
    End If
      ' メッセージボックスにエントリを表示します。
    MessageBox.Show(entryText, ze.FileName)
End Sub
```

C# でコードを書く場合

C#

```
private void btnView_Click(object sender, EventArgs e)
{
     // 最初に選択されたディレクトリではない項目を取得します。
    C1ZipEntry ze = null;
    foreach (ListViewItem lvi in listView1.SelectedItems)
    {
        C1ZipEntry zeItem = lvi.Tag as C1ZipEntry;
        if ((zeItem.Attributes == 0) && (FileAttributes.Directory == 0))
        {
            ze = zeItem;
            break;
        }
    }
     // 項目を取得したことを確認します。
    if (ze == null )
    {
        MessageBox.Show("Sorry, no files to show...", "C1Zip");
        return;
    }
     // エントリの内容を文字列に読み取ります。
    Stream entry = ze.OpenReader();
    StreamReader sr = new StreamReader(entry);
    string  entryText = sr.ReadToEnd();
    entry.Close();
     // エントリが MessageBox に対して長すぎないようにします。
    if (entryText.Length > 16000 )
    {
        entryText = entryText.Substring(0, 16000);
    }
     // メッセージボックスにエントリを表示します。
    MessageBox.Show(entryText, ze.FileName);
}
```

この関数は、最初に選択されたサブディレクトリではない zip エントリを選択します。有効なエントリが見つからない場合、関数は終了します。

次に、**OpenReader** メソッドを呼び出して、エントリの内容の読み取りに使用できるストリームを取得します。この手順には、一時ファイルも余分な遅延もありません。データは、ストリームから読み取られると同時に展開されます。

次に、**StreamReader** オブジェクトの **ReadToEnd** メソッドを使用して、エントリの内容を文字列に直接読み取った後、ストリームを閉じます。

最後に、文字列を切り詰めてメッセージボックスに表示します。

この関数は、短いテキストファイルの内容を表示することを目的としています。バイナリファイルの内容を表示しようとすると、数文字しか表示されません。

手順 10：zip ファイルにエントリを追加するコードを追加します。

[**ファイルの追加**]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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
```

```
     ' 追加するファイルのリストを取得します。
    Dim fo As OpenFileDialog = New OpenFileDialog()
    fo.Multiselect = True
    fo.FileName = "*.*"
    If fo.ShowDialog <> Windows.Forms.DialogResult.OK Then Exit Sub
     ' リストにファイルを追加します。
    Dim file As String
    For Each file In fo.FileNames()
        m_Zip.Entries.Add(file)
    Next file
      ' 完了。
    UpdateDisplay()
End Sub
```

C# でコードを書く場合

C#

```
private void btnAdd_Click(object sender, EventArgs e)
{
     // 追加するファイルのリストを取得します。
    OpenFileDialog fo = new OpenFileDialog();
    fo.Multiselect = true;
    fo.FileName = "*.*";
    if (fo.ShowDialog() == DialogResult.OK)
    {
        // リストにファイルを追加します。
        foreach (string file in fo.FileNames)
        {
            m_Zip.Entries.Add(file);
        }
          // 完了。
        UpdateDisplay();
    }
}
```

zip ファイルに新しいエントリを追加することは簡単です。重要な行は、C1ZipFile オブジェクトの **Add** メソッドの呼び出しだけです。ここで、追加するファイルの完全名を文字列で渡します。これで、やることはすべてです。

手順 11：zip エントリを抽出するコードを追加します。

[**ファイルの抽出**]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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub btnExtract_Click(sender As Object, e As EventArgs) Handles
btnExtract.Click
      ' いくつかのエントリを選択していることを確認します。
    Dim cnt As Integer = ListView1.SelectedIndices.Count
    If cnt = 0 Then
        MessageBox.Show("Sorry, no files to extract...", "C1Zip")
        Exit Sub
    End If
```

```
     ' ユーザーに確認します。
    Dim dr As DialogResult
    Dim msg As String
    msg = "Please confirm that you want to extract " + cnt.ToString() + " entries."
    dr = MessageBox.Show(msg, "C1Zip", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question)
    If dr <> Windows.Forms.DialogResult.OK Then Exit Sub
     ' 選択されたすべてのエントリを抽出します。
    Dim lvi As ListViewItem
    Dim ze As C1ZipEntry
    For Each lvi In ListView1.SelectedItems
        ze = lvi.Tag
        If ze.SizeCompressed > 0 Then
             m_Zip.Entries.Extract(ze.FileName)
        End If
    Next lvi
      ' 完了。
    UpdateDisplay()
End Sub
```

C# でコードを書く場合

C#

```
private void btnExtract_Click(object sender, EventArgs e)
{
     // いくつかのエントリを選択していることを確認します。
    int cnt = listView1.SelectedIndices.Count;
    if (cnt == 0)
    {
        MessageBox.Show("Sorry, no files to extract...", "C1Zip");
        return;
    }
     // ユーザーに確認します。
    DialogResult dr;
    string  msg;
    msg = "Please confirm that you want to extract " + cnt.ToString() + " entries.";
    dr = MessageBox.Show(msg, "C1Zip", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question);
    if ( dr != DialogResult.OK )  return;
     // 選択されたすべてのエントリを抽出します。
    C1ZipEntry ze;
    foreach (ListViewItem lvi in ListView1.SelectedItems)
    {
        ze = (C1ZipEntry)lvi.Tag;
        if ( ze.SizeCompressed > 0 )
        {
            m_Zip.Entries.Extract(ze.FileName);
        }
    }
      // 完了。
    UpdateDisplay();
}
```

最も重要な行は、C1ZipFile オブジェクトの **Entries** プロパティの **Extract** メソッドの呼び出しです。パラメータは、エントリのファイル名を含む文字列です。これは、エントリの **FileName** プロパティに保存されています。).

デフォルトでは、Extract メソッドはエントリを展開し、それを zip ファイルのディレクトリに保存します。このメソッドには別のバージョンもあります。そこでは、出力ディレクトリと展開ファイルのファイル名を指定できます。詳細については、「**C1.C1Zip 名前空間** 」セクションを参照してください。

手順 12：zip ファイルからエントリを削除するコードを追加します。

[**ファイルの削除**]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。

Visual Basic でコードを書く場合

Visual Basic

```
Private Sub btnRemove_Click(sender As Object, e As EventArgs) Handles btnRemove.Click
      ' いくつかのエントリを選択していることを確認します。
    Dim cnt As Integer = ListView1.SelectedIndices.Count
    If cnt = 0 Then
        MessageBox.Show("Oops, no files to remove...", "C1Zip")
        Exit Sub
    End If
     ' ユーザーに確認します。
    Dim dr As DialogResult
    Dim msg As String
    msg = "Please confirm that you want to delete " + cnt.ToString() + " entries."
    dr = MessageBox.Show(msg, "C1Zip", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question)
    If dr <> Windows.Forms.DialogResult.OK Then Exit Sub
      ' 選択されたすべてのエントリを削除します。
    Dim lvi As ListViewItem
    Dim ze As C1ZipEntry
    For Each lvi In ListView1.SelectedItems
        ze = lvi.Tag
        m_Zip.Entries.Remove(ze.FileName)
    Next lvi
      ' 完了。
    UpdateDisplay()
End Sub
```

C# でコードを書く場合

C#

```
private void btnRemove_Click(object sender, EventArgs e)  {
     // いくつかのエントリを選択していることを確認します。
    int cnt = listView1.SelectedIndices.Count;
    if (cnt == 0 )
    {
        MessageBox.Show("Oops, no files to remove...", "C1Zip");
        return;
    }
     // ユーザーに確認します。
     DialogResult dr;
    string  msg;
```

```
    msg = "Please confirm that you want to delete " + cnt.ToString() + " entries.";
    dr = MessageBox.Show(msg, "C1Zip", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question);
    if (dr != DialogResult.OK) return;
     // 選択されたすべてのエントリを削除します。
    foreach (ListViewItem lvi in listView1.SelectedItems)
    {
        C1ZipEntry ze = (C1ZipEntry)lvi.Tag;
        if (ze.SizeCompressed > 0)
        {
            m_Zip.Entries.Remove(ze.FileName);
        }
    }
      // 完了。
    UpdateDisplay();
}
```

最も重要な行は、C1ZipFile オブジェクトの Entries プロパティの **Remove** メソッドの呼び出しです。パラメータは、エントリのファイル名を含む文字列です。これは、エントリの FileName プロパティに保存されています。

これで「**Zip ファイルの処理**」チュートリアルは終了です。

よくある質問と回答

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

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。
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。