
ComponentOne

Zip for .NET

Copyright © 1987-2011 ComponentOne LLC. All rights reserved.

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: info@ComponentOne.com

Web site: <http://www.componentone.com>

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

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

Warranty

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for \$25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

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

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

Table of Contents

ComponentOne Zip for .NET Overview	1
What's New in Zip for .NET	1
Revision History	1
What's New in 2010 v2	1
Installing Zip for .NET	2
Zip for .NET Setup Files	2
System Requirements	3
Installing Demonstration Versions	3
Uninstalling Zip for .NET	4
End-User License Agreement	4
Licensing FAQs	4
What is Licensing?	4
How does Licensing Work?	4
Common Scenarios	5
Common Scenarios in Mobile Applications	7
Troubleshooting	8
Technical Support	10
Redistributable Files	10
About this Documentation	11
Namespaces	11
Creating a .NET Project	12
Creating a Mobile Device Application	13
Adding a Reference to the C1Zip Assembly	14
Migrating a Zip for .NET Project to Visual Studio 2005	15
Key Features	17
Zip for .NET Fundamentals	18
High Level: C1ZipFile, C1ZipEntry and C1ZipEntryCollection Classes	19
Medium Level: C1ZStreamReader and C1ZStreamWriter Classes	20
Low Level: ZStream Class	21
ComponentOne Zip for Mobile Devices Fundamentals	21

Zip for .NET Samples	22
Zip for .NET Frequently Asked Questions	23
Zip for .NET Task-Based Help	23
Compressing Datasets	23
Compressing an Entire Folder into a Zip File	25
Creating a Zip File with Multiple Entries	25
Extracting Files from Zip Entry to Memory	26
Reading a Zipped File Using a StreamReader	27
Retrieving Images from a Zip File	28
Saving a String Variable to a Zip File	30
Setting the Level of Compression	31
Using Passwords to Protect Zip Files	32
Zip for .NET Tutorials	33
Compressing Data in Memory	33
Compressing Files	41
Compressed Serialization	50
Handling Zip Files	56

ComponentOne Zip for .NET Overview

Quickly compress data, saving disk space and network bandwidth, and easily manipulate and work with compressed data. **ComponentOne Zip™ for .NET** allows you to compress files from your applications, including system and hidden files, and add and delete compressed files and folders; you can even read and write to zip files stored in streams.

Zip for .NET classes are supported in all .NET development platforms including WinForms, ASP.NET, WPF, Mobile, and a special Silverlight version exists in ComponentOne Studio for Silverlight.


Getting Started

To get started, review the following topics:

- [Key Features](#) (page 17)
- [Zip for .NET Fundamentals](#) (page 18)
- [Samples](#) (page 22)

What's New in Zip for .NET

This documentation was last revised for 2010 v3 on October 25, 2010. There are no new features added to **Zip for .NET** for 2010 v3.

 **Tip:** A version history containing a list of new features, improvements, fixes, and changes for each product is available in HelpCentral at <http://helpcentral.componentone.com/VersionHistory.aspx>.

Revision History

The revision history provides recent enhancements to **Zip for .NET**.

What's New in 2010 v2

The following enhancement was made to **Zip for .NET** in the 2010 v2 release of the ComponentOne Studios

Support for Zip64 files has been added to **ComponentOne Zip for .NET**. This allows for entries longer than the usual 4GB limit and also allows for more entries per Zip file.

Class Members

The following members have been added to **Zip for .NET** in the 2010 v2 release:

Member	Description
C1ZipEntry.SizeCompressedLong Property	Gets the compressed size of the entry, in bytes, as a long integer.
C1ZStreamWriter.SizeCompressedLong Property	Gets the number of bytes in the stream (compressed bytes).
C1ZStreamReader.SizeCompressedLong Property	

C1ZipEntry.SizeUncompressedLong Property	Gets the original (uncompressed) size of the entry, in bytes, as a long integer.
C1ZipStreamReader.SizeUncompressedLong Property	Gets the number of bytes that were compressed into the stream (uncompressed bytes).
C1ZipStreamWriter.SizeUncompressedLong Property	
ZipProgressEventArgs.FileLengthLong Property	Gets the length of the file being compressed or expanded.
ZipProgressEventArgs.PositionLong Property	Gets the current position into the stream.
Add Method (String, String, DateTime)	Adds an entry to the current zip file.
Add Method (Stream, String, DateTime)	Adds a stream to the current zip file.
OpenWriter Method (String, DateTime, Boolean)	Opens a stream for writing an entry into the zip file.

Installing Zip for .NET

The following sections provide helpful information on installing **ComponentOne Zip for .NET**.

Zip for .NET Setup Files

The ComponentOne Studio for WinForms installation program will create the following directory C:\Program Files\ComponentOne\Studio for WinForms. Note that **Studio for WinForms** will be replaced by the corresponding platform, depending on which ComponentOne Studio you installed. The directory contains the following subdirectories:

bin	Contains copies of all ComponentOne binaries (DLLs, EXEs).
C1Zip or C1ZipCF (Mobile)	Contains files (at least a readme.txt) related to the ComponentOne Zip for .NET product.

The **ComponentOne Studio for WinForms Help Setup** program installs integrated Microsoft Help 2.0 and Microsoft Help Viewer help to the C:\Program Files\ComponentOne\Studio for WinForms directory in the following folders:

H2Help	Contains Microsoft Help 2.0 integrated documentation for all Studio components.
HelpViewer	Contains Microsoft Help Viewer Visual Studio 2010 integrated documentation for all Studio components.

Samples

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the **ComponentOne Samples** directory is slightly different on Windows XP and Windows 7/Vista machines:

Windows XP path: C:\Documents and Settings\\My Documents\ComponentOne Samples

Windows 7/Vista path: C:\Users\\Documents\ComponentOne Samples

The **ComponentOne Samples** folder contains the following subdirectories:

- Common** Contains support and data files that are used by many of the demo programs.
- C1Zip** Contains samples and tutorials for **Zip for .NET**.

Samples can be accessed from the **ComponentOne Sample Explorer**. To view samples, on your desktop, click the Start button and then click **All Programs | ComponentOne | Studio for WinForms (or the appropriate Studio) | Samples | Zip Samples**.

System Requirements

System requirements include the following:

- Operating Systems:** Windows® 2000
 Windows Server® 2003
 Windows Server 2008
 Windows XP SP2
 Windows Vista™
 Windows 7
- Environments:** .NET Framework 2.0 or later
 .NET Framework 3.5 or later (WPF)
 .NET Compact Framework 2.0 or later (Mobile)
 C# .NET
 Visual Studio® 2005 or later
 Visual Studio 2005 extensions for .NET Framework 2.0 November 2006 CTP (WPF)
 Visual Studio 2008 or later (WPF)
 Internet Explorer 6.0 or later, Firefox® 2.0 or later, or Safari® 2.0 or later (ASP.NET)
- Web Server (for ASP.NET)** Microsoft Internet Information Service (IIS) 5.0 or later
- Disc Drive:** CD or DVD-ROM drive if installing from CD

Installing Demonstration Versions

If you wish to try **ComponentOne Zip for .NET** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that registered versions will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

Uninstalling Zip for .NET

To uninstall **ComponentOne Zip for .NET**:

1. Open the Control Panel and select **Add or Remove Programs (Programs and Features** in Windows 7/Vista).
2. Select **ComponentOne Studio for WinForms** and click the **Remove** button. Note that depending on which ComponentOne Studio you installed, the name of the studio will appear under **Add or Remove Programs**.
3. Click **Yes** to remove the program.

To uninstall **ComponentOne Zip for .NET** integrated help:

1. Open the Control Panel and select **Add or Remove Programs** (Programs and Features in Windows 7/Vista).
1. Select **ComponentOne Studio for WinForms Help** and click the **Remove** button. Note that depending on which ComponentOne Studio Help you installed, the name of the studio will appear under **Add or Remove Programs**.
2. Click **Yes** to remove the integrated help.

End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, frequently asked licensing questions, and the ComponentOne licensing model, is available online at <http://www.componentone.com/SuperPages/Licensing/>.

Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

Note: The **Compact Framework** components use slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the About Box of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog.)

When a licensed component is added to a form or Web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license
- A licenses.licx file that contains the licensed component strong name and version information

These files are automatically added to the project.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the App_Licenses.dll assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. Thus, the App_licenses.dll must always be deployed with the application.

The licenses.licx file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for runtime licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the licenses.licx file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's Toolbox, or select **Show All Files** from the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the licenses.licx file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a licenses.licx file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the licenses.licx file and things will then work as expected. (The component can be removed from the form after the licenses.licx file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the licenses.licx file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a LicenseProvider attribute to the component.

This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the licenses.licx file, and the base class will handle the licensing process as usual. No additional work is needed. For example:

```
[LicenseProvider(typeof(LicenseProvider))]  
class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid  
{  
    // ...  
}
```

- Add an instance of the base component to the form.

This will embed the licensing information into the licenses.licx file as in the previous scenario, and the base component will find it and use it. As before, the extra instance can be deleted after the licenses.licx file has been created.

Please note, that C1 licensing will not accept a run-time license for a derived control if the run-time license is embedded in the same assembly as the derived class definition, and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

Using licensed components in console applications

When building console applications, there are no forms to add components to, and therefore Visual Studio won't create a licenses.licx file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the licenses.licx file into the console application project.

Make sure the licenses.licx file is configured as an embedded resource. To do this, right-click the licenses.licx file in the Solution Explorer window and select **Properties**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

Using licensed components in Visual C++ applications

There is an issue in VC++ 2003 where the licenses.licx is ignored during the build process; therefore, the licensing information is not included in VC++ applications.

To fix this problem, extra steps must be taken to compile the licensing resources and link them to the project. Note the following:

1. Build the C++ project as usual. This should create an .exe file and also a licenses.licx file with licensing information in it.
2. Copy the licenses.licx file from the app directory to the target folder (Debug or Release).
3. Copy the C1Lc.exe utility and the licensed dlls to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use C1Lc.exe to compile the licenses.licx file. The command line should look like this:
`c1lc /target:MyApp.exe /complist:licenses.licx /i:C1.Win.C1FlexGrid.dll`
5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select **Properties**, and go to the **Linker/Command Line** option. Enter the following:
`/ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses`

6. Rebuild the executable to include the licensing information in the application.

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information, and there is no easy way to add it.

This can be avoided by adding the string "C1CheckForDesignLicenseAtRuntime" to the AssemblyConfiguration attribute of the assembly that contains or derives from ComponentOne controls. This attribute value directs the ComponentOne controls to use design-time licenses at run time.

For example:

```
#if AUTOMATED_TESTING
[assembly:Configuration("C1CheckForDesignLicenseAtRuntime")]
#endif
public class MyDerivedControl : C1LicensedControl
{
    // ...
}
```

Note that the AssemblyConfiguration string may contain additional text before or after the given string, so the AssemblyConfiguration attribute can be used for other purposes as well. For example:

```
[assembly:Configuration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

Common Scenarios in Mobile Applications

The following topics describe some of the licensing scenarios you may encounter when working with Mobile components.

Updating or renewing a license

If you renew your subscription, the new license must be installed.

If the Mobile controls are licensed through a Studio Enterprise subscription, then open the **About Box** of either an ASP.NET control or a .NET Windows forms control and update the license by clicking the **License** button and entering your serial number.

If the Mobile controls are licensed through a Studio for Mobile Devices subscription, then open the **About Box** of the 1.x version of a Mobile control or run the setup again to enter your serial number. Presently, the 2.x versions of the Mobile controls do not have the option to license or register from the **About Box**; therefore, it is necessary to license through another component or the setup.

Licensing 2.x Mobile Controls through the Setup

To enter the serial number (license) through the Studio for Mobile Devices 2.0 setup, follow these steps:

1. Run the ComponentOne Studio for Mobile Devices 2.0 setup.
2. Follow the instructions in the setup, and enter the serial number when prompted.

There are cases where the setup may not prompt you. If you have a valid license installed already, or if you are installing a version of the controls that has already been installed (Maintenance Mode), you will not be prompted to enter your serial number. If you need to enter your serial number for an install, and the install is running in Maintenance Mode, you will need to uninstall first. Once you uninstall, run the install again.

If you are still not prompted for a serial number by the time you get to the install screen, you must have a valid license in the registry on your machine. If you still need to install a new serial number, remove the old license and

then run the setup again. There is a utility available that will remove the old license for you, which can be found below.

License Remover

The license remover will search for the following licenses on your system: Studio Enterprise, Studio for Mobile Devices, and individual Studio Mobile product licenses. Any licenses that are found will be populated into a list so that you can select the ones that you would like to remove. To remove a Studio for Mobile Devices license with the license remover, follow these steps:

1. Unzip and run the C1LicBomb.exe.
2. You will see an app with a list of the installed licenses on your machine. If you see both Studio Enterprise and Studio for Mobile devices listed, select both for removal, otherwise select the one that you see. If the new serial number that you wish to enter for Studio for Mobile Devices is not a Studio Enterprise serial number, you will need to enter your Studio Enterprise serial number again, but this can be done through any of the About Boxes for C1 controls with the exception of the Mobile Studio About Boxes.
3. Select any other licenses you wish to remove, then click the **Remove Selected Licenses** button.
4. You will be asked if you are sure that you want to delete each key; click **OK** when you get the dialog box.
5. There will also be a dialog box to let you know that a particular key was removed successfully; click **OK** for that one also.
6. Now you can close the program and run your Studio for Mobile Devices setup again to enter your new serial number.

Follow through with the setup, enter your serial number when prompted, and ComponentOne Studio for Mobile Devices will be licensed on your machine.

Updating a project after renewing a license

Once you have installed a new license, each project must be updated in order to use the new control; rebuilding the control is not sufficient. It is necessary for the control to regenerate the license embedded in its **SupportInfo** property. To do this, it is necessary to force Visual Studio to update the control properties stored in the form. The easiest way to do this is to simply modify a property. The simplest choice is to toggle a Boolean property such as the **Visible** property, and then toggle it back to its original value. This results in no changes or side effects in the control configuration, but it forces the IDE to update **SupportInfo** and embed the new run-time license.

Instantiating a Mobile control at run time

The Mobile controls behave the same way as other ComponentOne controls when they are created at run time and not placed on the form at design time. Because the IDE does not have an opportunity to obtain a run-time license on its own in this case, it is necessary to force the IDE to include a run-time license. To accomplish this, include at least one instance of the control on a form in the assembly that is instantiated BEFORE the dynamically created instance is created. Once a control of the same type is licensed, all others on the form are accepted as licensed.

Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.

If this happens, there may be a problem with the licenses.licx file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

If that fails follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and open it. If prompted, continue to open the file.
4. Change the version number of each component to the appropriate value. If the component does not appear in the file, obtain the appropriate data from another licenses.licx file or follow the alternate procedure following.
5. Save the file, and then close the licenses.licx tab.
6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

Alternatively, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and delete it.
4. Close the project and reopen it.
5. Open the main form and add an instance of each licensed control.
6. Check the Solution Explorer window, there should be a licenses.licx file there.
7. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

For ASP.NET 2.x applications, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Find the licenses.licx file and right-click it.
3. Select the **Build Runtime Licenses** from its context menu (this will rebuild the App_Licenses.licx file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

I have a licensed version of a ComponentOne product on my Web server but the components still behave as unlicensed.

There is no need to install any licenses on machines used as servers and not used for development.

The components must be licensed on the development machine, therefore the licensing information will be saved into the executable (.exe or .dll) when the project is built. After that, the application can be deployed on any machine, including Web servers.

For ASP.NET 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the Web server.

If your ASP.NET application uses WinForms user controls with constituent licensed controls, the runtime license is embedded in the WinForms user control assembly. In this case, you must be sure to rebuild and update the user control whenever the licensed embedded controls are updated.

I downloaded a new build of a component that I have purchased, and now I'm getting the splash screen when I build my projects.

Make sure that the serial number is still valid. If you licensed the component over a year ago, your subscription may have expired. In this case, you have two options:

Option 1 – Renew your subscription to get a new serial number.

If you choose this option, you will receive a new serial number that you can use to license the new components (from the installation utility or directly from the About Box).

The new subscription will entitle you to a full year of upgrades and to download the latest maintenance builds directly from <http://prerelease.componentone.com/>.

Option 2 – Continue to use the components you have.

Subscriptions expire, products do not. You can continue to use the components you received or downloaded while your subscription was valid.

Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at <http://www.componentone.com/SuperProducts/SupportServices/>.

Some methods for obtaining technical support include:

- **Online Support via [HelpCentral](#)**
ComponentOne HelpCentral provides customers with a comprehensive set of technical resources in the form of [FAQs](#), [samples](#), [Version Release History](#), [Articles](#), searchable [Knowledge Base](#), searchable [Online Help](#) and more. We recommend this as the first place to look for answers to your technical questions.
- **Online Support via our Incident Submission Form**
This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This email will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.
- **Peer-to-Peer Product Forums and Newsgroups**
ComponentOne peer-to-peer product [forums and newsgroups](#) are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums and newsgroups, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.
- **Installation Issues**
Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.
- **Documentation**
Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. If you have suggestions on how we can improve our documentation, please email the [Documentation team](#). Please note that e-mail sent to the [Documentation team](#) is for documentation feedback only. [Technical Support](#) and [Sales](#) issues should be sent directly to their respective departments.

Note: You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Redistributable Files

ComponentOne Zip for .NET is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following Redistributable

Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.C1Zip.2.dll
- C1.C1Zip.4.dll
- C1.CF.C1Zip.2.dll (Zip for Mobile Devices)
- C1.CF.C1Zip.3.dll (Zip for Mobile Devices)

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

About this Documentation

Acknowledgements

Microsoft, Windows, Windows Vista, Windows Server, and Visual Studio are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Firefox is a registered trademark of the Mozilla Foundation.

Safari is a registered trademark of Apple Inc.

ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

Corporate Headquarters

ComponentOne LLC

201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA
412.681.4343
412.681.4384 (Fax)

<http://www.componentone.com/>

ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

Namespaces

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

The namespaces in the **C1Zip** assembly are:

- C1.C1Zip
- C1.C1Zip.ZLib

The following code fragment shows how to declare a C1ZipFile component using the fully qualified name for this class:

- Visual Basic

```
Dim zip As C1.C1Zip.C1ZipFile
```
- C#

```
C1.C1Zip.C1ZipFile zip;
```

Namespaces address a problem sometimes known as *namespace pollution*, in which the developer of a class library is hampered by the use of similar names in another library. These conflicts with existing components are sometimes called *name collisions*.

For example, if you create a new class named **C1ZipFile**, you can use it inside your project without qualification. However, the **C1Zip** assembly also implements a class called **C1ZipFile**. So, if you want to use the C1ZipFile class in the same project, you must use a fully qualified reference to make the reference unique. If the reference is not unique, Visual Studio .NET produces an error stating that the name is ambiguous. The following code snippet demonstrates how to declare these objects:

- Visual Basic

```
' Define a new C1ZipFile object (custom C1ZipFile class).  
Dim MyC1ZipFile as C1ZipFile  
  
' Define a new C1Zip.C1ZipFile object.  
Dim ZipFile as C1.C1Zip.C1ZipFile
```

- C#

```
// Define a new C1ZipFile object (custom C1ZipFile class).  
MyC1ZipFile as C1ZipFile;  
  
// Define a new C1Zip.C1ZipFile object.  
ZipFile as C1.C1Zip.C1ZipFile;
```

Fully qualified names are object references that are prefixed with the name of the namespace where the object is defined. You can use objects defined in other projects if you create a reference to the class (by choosing **Add Reference** from the **Project** menu) and then use the fully qualified name for the object in your code.

Fully qualified names prevent naming conflicts because the compiler can always determine which object is being used. However, the names themselves can get long and cumbersome. To get around this, you can use the **Imports** statement (**using** in C#) to define an alias — an abbreviated name you can use in place of a fully qualified name. For example, the following code snippet creates aliases for two fully qualified names, and uses these aliases to define two objects:

- Visual Basic

```
Imports C1File = C1.C1Zip.C1ZipFile  
Imports MyFile = MyProject.C1ZipFile  
Dim f1 As C1File  
Dim f2 As MyFile
```

- C#

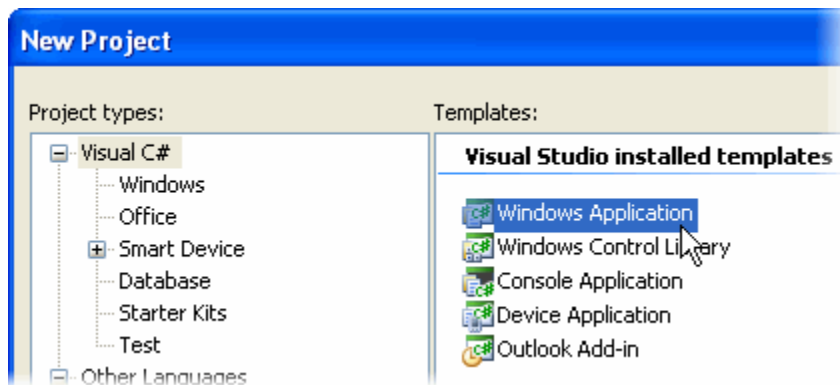
```
using C1File = C1.C1Zip.C1ZipFile;  
using MyFile = MyProject.C1ZipFile;  
C1File f1;  
MyFile f2;
```

If you use the **Imports** statement without an alias, you can use all the names in that namespace without qualification provided they are unique to the project.

Creating a .NET Project

To create a new .NET project, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio .NET, select **New** and click **Project**. The **New Project** dialog box opens.
2. Under **Project types**, choose either **Visual Basic Projects** or **Visual C# Projects**, and select **Windows Application** from the list of **Templates** in the right pane.



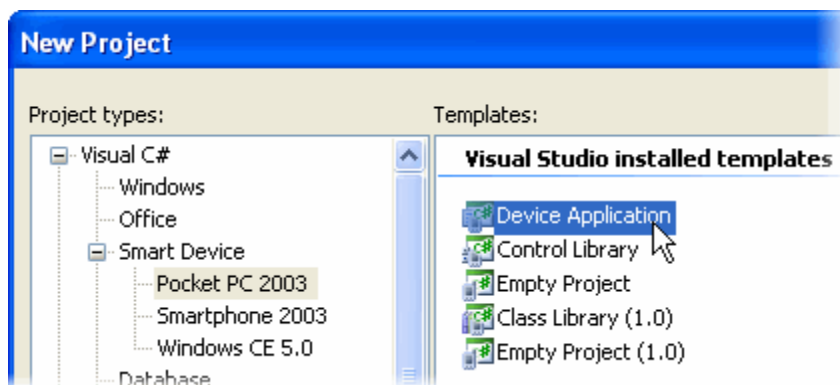
3. Enter a name for your new application in the **Name** field or you can use the default name, WindowsApplication1. Click on the **Browse** button to the right of the **Location** field to specify the location of your new project and click **OK**.

A new Microsoft Visual Studio .NET project is created in the specified location. In addition, a new Form1 is displayed in the Designer view.

Creating a Mobile Device Application

To create a new .NET 2.0 device application project, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio .NET, select **New | Project**. The **New Project** dialog box opens.
2. Under **Project Types**, expand the **Visual Basic** or **Visual C#** node. Note that one of these options may be located under **Other Languages**.
3. Expand the **Smart Device** node and select one of the smart devices listed.
4. Select **Device Application** from the list of **Templates** in the right pane.

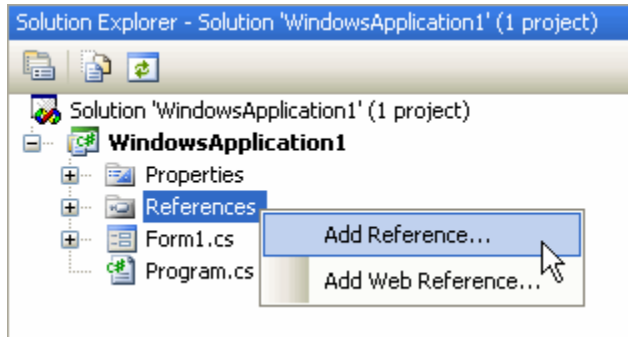


5. Enter a name in the **Name** textbox and click **OK**. A new project is created, and a new Form1 is displayed in the Designer view.
6. Reference the C1Zip assembly in your project. For information, see [Adding a Reference to the C1Zip Assembly](#) (page 14).

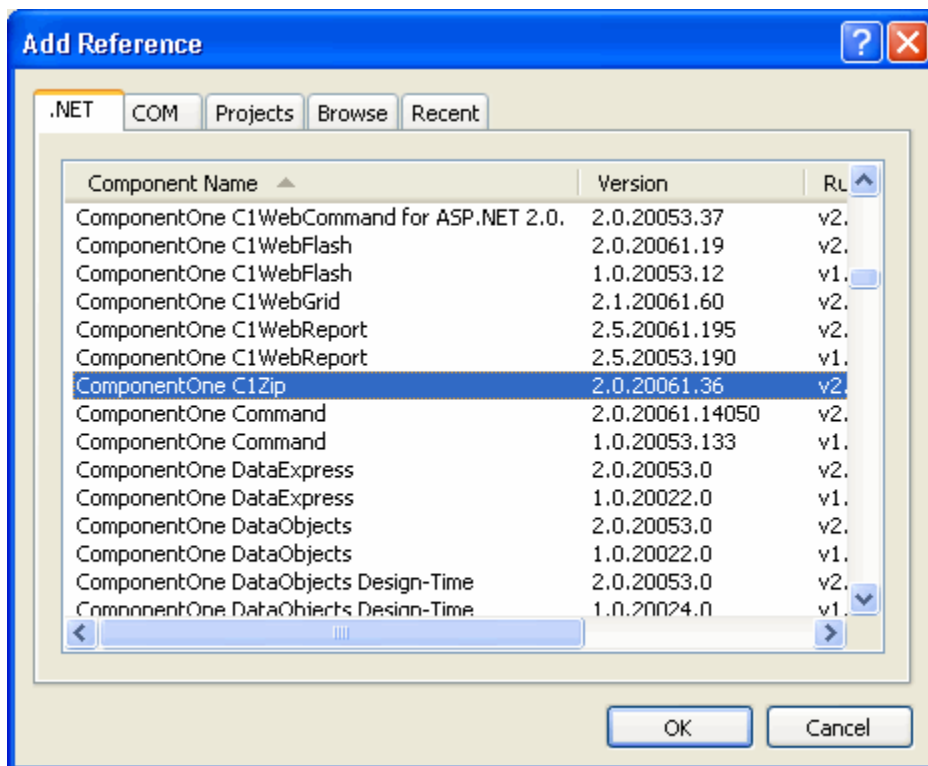
Adding a Reference to the C1Zip Assembly

When you install **ComponentOne Zip for .NET**, a new component does not appear in the Visual Studio Toolbox. Therefore, to use **Zip for .NET**, you must reference the **C1Zip** assembly in your project. To use **Zip for .NET**:

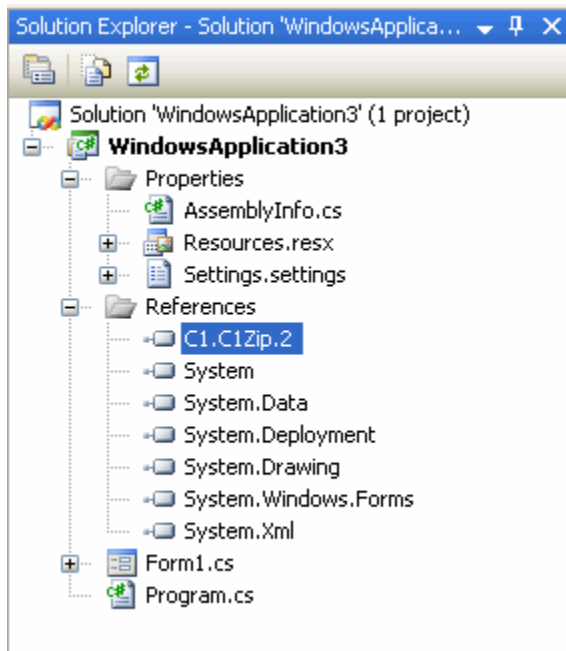
1. Select the **Add Reference** option from the **Project** menu of your project.



2. The **Add Reference** dialog box appears. Select the **C1.C1Zip** assembly from the list, or browse to find the C1.C1Zip.2.dll or C1.C1Zip.4.dll file. Double-click the component and select **OK**.



3. In the Solution Explorer, notice the C1Zip assembly has been referenced in your project.



4. Select the **Form1.vb** tab (**Form1.cs** in C#) or go to **View | Code** to open the Code Editor. At the top of the file, add the following **Imports** statements (**using** in C#):

- Visual Basic

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

- C#

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

Note: This makes the objects defined in the C1Zip and System.IO assemblies visible to the project and saves a lot of typing.

Migrating a Zip for .NET Project to Visual Studio 2005

To migrate a project using ComponentOne components to Visual Studio 2005, there are two main steps that must be performed. First, you must convert your project to Visual Studio 2005, which includes removing any references to a previous assembly and adding a reference to the new assembly. Second, you must update the project References to use the new C1.C1Zip.2.dll.

To convert the project:

1. Open Visual Studio 2005 and select **File | Open | Project/Solution**.
2. Locate the **.sln** file for the project that you wish to convert to Visual Studio 2005. Select it and click **Open**. The **Visual Studio Conversion Wizard** appears.
3. Click **Next**.
4. Select **Yes, create a backup before converting** to create a backup of your current project and click **Next**.
5. Click **Finish** to convert your project to Visual Studio 2005. The **Conversion Complete** window appears.
6. Click Show the conversion log when the wizard is closed if you want to view the conversion log.

7. Click **Close**. The project opens. Now you must remove references to any of the previous ComponentOne .dlls and add references to the new ones.
8. Go to the Solution Explorer (**View | Solution Explorer**), select the project, and click the **Show All Files** button.

Note: The **Show All Files** button does not appear in the Solution Explorer toolbar if the Solution project node is selected.

9. Expand the **References** node, right-click C1.C1Zip and select **Remove**.
10. Right-click the **References** node and select **Add Reference**.
11. Locate and select **C1.C1Zip.2.dll**. Click **OK** to add it to the project.

Key Features

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

- **Major operations**

With Zip for .NET, you can:

- Obtain global information on a zip file.
- Obtain a detailed list of a zip file's contents via a collection object.
- Delete files from within a zip file.
- Test the integrity of a zip file and its contents.
- Add and retrieve comments for individual files in the zip file.
- Get and set the global zip file comment.
- Control what path information is stored in the zip file for each file.
- Zip system and hidden files.
- Control the amount of compression applied to files being zipped.
- Specify the path where files will be uncompressed to.
- Work on temporary copies of zip files, for maximum safety.

- **Compress and Expand Folders**

With Zip you can easily compress and expand folders while preserving folder structure.

- **Read and Write to Streams**

Zip can read and write to zip files stored in streams in addition to actual files using the `C1ZipFile.Open(Stream)` method. This allows reading zip files embedded in application resources or stored in database fields instead of in actual files.

- **Get and Set File Information**

Easily obtain zip file information including a detailed list of the zip file's contents. You can also control file information by adding and retrieving comments and control path information for individual entries in the zip file, and getting and setting the global zip file comment.

- **File Integrity and Safety**

With Zip you can test the integrity of a zip file and its contents and work on temporary copies of zip files to ensure maximum safety.

- **Compress and Manipulate Compressed Files**

Compress files, including system and hidden files, and easily manipulate compressed files by deleting files from within a zip file, controlling the amount of compression applied to files being zipped, and specifying the path where files will be uncompressed to.

- **Incorporates the ZLIB Advantage**

ZLIB is a general-purpose, lossless data-compression library with a portable, cross-platform data format. Unlike Unix compress and GIF image format LZW compression, the ZLIB compression method essentially never expands the data (LZW doubles or triples the file size in extreme cases) and has a memory footprint independent of input data.

- **Support for Zip64**

C1Zip now supports Zip64 files. This allows for entries longer than the usual 4 gig limit and also allows more entries per zip file. The maximum number of entries per ZIP file is 2,147,483,647 (int.MaxValue). The maximum uncompressed entry size is 9,223,372,036,854,775,807 (long.MaxValue).

- **Fast, low memory usage compression engine**

The low memory compression engine lets you:

- Zip files or memory buffers into new or existing zip files.
- Unzip files to disk or directly to memory.
- Compress and decompress strings or buffers completely in memory.
- Stream-based compression and decompression for maximum flexibility.

- **100% managed C# code**

C1.C1Zip.ZLib is a C# implementation of Adler and Gailly's ZLIB code and contains a main class ZStream that handles stream-based compression and decompression. ZLIB handles the actual data compression and decompression.

ZLIB is designed to be a free, general-purpose, lossless data-compression library for use on virtually any computer hardware and operating system. The ZLIB data format is itself portable across platforms. Unlike the LZW compression method used in Unix compress and in the GIF image format, the compression method currently used in ZLIB essentially never expands the data. (LZW can double or triple the file size in extreme cases.) ZLIB's memory footprint is also independent of the input data.

- **Simple, intuitive object model**

Zip for .NET contains three main classes:

- C1ZStreamWriter is a stream object that takes regular data on input, compresses it, and writes it out to an underlying stream. You can compress data to files, memory, or any other type of stream. You can attach a **BinaryWriter** object to C1ZStreamWriter to write objects directly, without dealing with byte arrays.
- C1ZStreamReader is a stream object that takes a compressed stream as input and decompresses it as it reads data from the underlying compressed stream. You can attach a **BinaryReader** object to C1ZStreamReader to read objects directly, without dealing with byte arrays.
- C1ZipFile is a class that deals with zip files. It allows you to create, open, and modify zip files. C1ZipFile has a Entries property that represents a collection of entries in a zip file.

Zip for .NET Fundamentals

The classes in the **C1Zip** library are divided into three levels:

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

High Level: C1ZipFile, C1ZipEntry and C1ZipEntryCollection Classes

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

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

C1ZipFile Class

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

- Visual Basic

```
' Create a C1ZipFile object.
Dim myZip As New C1ZipFile()

' Create a new (empty) zip file.
myZip.Create("New.zip")

' Open an existing zip file.
myZip.Open("Old.zip")
```

- C#

```
// Create a C1ZipFile object.
C1ZipFile myZip = new C1ZipFile();

// Create a new (empty) zip file.
myZip.Create("New.zip");

// Open an existing zip file.
myZip.Open("Old.zip");
```

C1ZipEntryCollection Class

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

- Visual Basic

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

- 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 Class

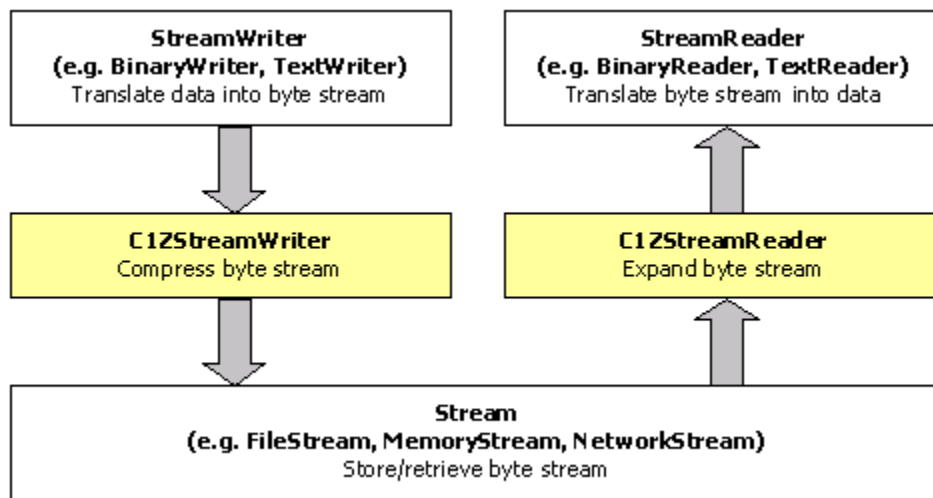
The C1ZipEntry class exposes properties and methods that describe each entry, including its original file name, size, compressed size, and so on. It also has a OpenReader method that returns a stream object, so you can read the entry contents without expanding it first.

Medium Level: C1ZStreamReader and C1ZStreamWriter Classes

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

To use C1ZStreamReader and C1ZStreamWriter objects, attach them to regular streams and read or write the data through them. The data is compressed (or expanded) on the fly into (or out of) the underlying stream.

This design allows great integration with native .NET streams. The diagram below illustrates how it works:



For example, the code below saves an ADO.NET **DataTable** object into a stream and then reads it back:

- Visual Basic

```
' Save the DataTable into a compressed stream.
Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter()
Dim fout As New FileStream("test.tmp", FileMode.Create)
bf.Serialize(fout, DataTableOut)

fout.Close()
' Read the compressed data.
Dim fin As New FileStream("test.tmp", FileMode.Open)
Dim DataTableIn As DataTable = bf.Deserialize(fin)
```

- C#

```
// Save the DataTable into compressed stream.
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();

// Read the compressed data.
FileStream fin = new FileStream("test.tmp", FileMode.Open);
DataTable DataTableIn = (DataTable)bf.Deserialize(fin);
```

To add data compression, you would simply add two lines of code:

- Visual Basic

```
' Save the DataTable into a compressed stream.
```

```

Dim bf As New Runtime.Serialization.Formatters.Binary.BinaryFormatter()
Dim fout As New FileStream("test.tmp", FileMode.Create)
Dim compressor As New ClZStreamWriter(fout)
bf.Serialize(compressor, DataTableOut)
fout.Close()

' Read the compressed data.
Dim fin As New FileStream("test.tmp", FileMode.Open)
Dim decompressor As New ClZStreamReader(fin)
Dim DataTableIn As DataTable = bf.Deserialize(decompressor)

```

- C#

```

// Save the DataTable into a compressed stream.
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter bf = new
System.Runtime.Serialization.Formatters.Binary.BinaryFormatter();
FileStream fout = new FileStream("test.tmp", FileMode.Create);
ClZStreamWriter compressor = new ClZStreamWriter(fout);
bf.Serialize(compressor, DataTableOut);
fout.Close();

// Read compressed data.
FileStream fin = new FileStream("test.tmp", FileMode.Open);
ClZStreamReader decompressor = new ClZStreamReader(fin);
DataTable DataTableIn = (DataTable)bf.Deserialize(decompressor);

```

Low Level: ZStream Class

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

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

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

ComponentOne Zip for Mobile Devices Fundamentals

The mobile version of **ComponentOne Zip for .NET** allows you to quickly develop Microsoft.NET Compact Framework-based applications for your mobile devices, such as personal digital assistants (PDA's), mobile phones, and more. It implements several classes that allow you to add data compression to your applications.

ComponentOne Zip for Mobile supports all members of the object model that are supported by the .NET Compact Framework.


Note that the code examples provided in this documentation are examples for using **ComponentOne Zip for .NET**. If you would like to see samples for the mobile version, they are installed with **ComponentOne Zip for Mobile Devices**, or you can get them online at <http://helpcentral.componentone.com/Samples.aspx>.

Zip for .NET Samples


Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other development tools included with ComponentOne Studio Enterprise.

Samples can be accessed from the **ComponentOne Sample Explorer**. To view samples, on the desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WinForms (or Studio for ASP.NET) | Samples | Zip Samples**.


Click one of the following links to view a list of **C1Zip** samples:

- [Visual Basic Samples](#) 

Sample	Description
Files	Shows how to compress and expand individual files. This sample calls the C1.C1Zip namespace.
Memory	Shows how to compress and expand memory streams. This sample calls the C1.C1Zip namespace.
Serialization	Shows how to add compression to the .NET serialization support. This sample calls the C1.C1Zip namespace.
ZipFileDemo	Shows how to implement a simple zip utility. This sample uses the C1.C1Zip namespace.
ZipFileSimple	Shows how to implement a simple zip utility. This sample lets you create, open, and edit standard zip files. Note that this is a stripped-down version of the ZipFileDemo sample.

- [C# Samples](#) 

Sample	Description
Files	Shows how to compress and expand individual files. This sample calls the C1.C1Zip namespace.
InMemoryCompression	Shows how to create and use an in-memory archive. This sample calls the C1.C1Zip namespace.
Memory	Shows how to compress and expand memory streams. This sample calls the C1.C1Zip namespace.
SelfExtract	Self-extracting application. This sample calls the C1.C1Zip namespace.
Serialization	Shows how to add compression to the .NET serialization support. This sample calls the C1.C1Zip namespace.
ZipFileDemo	Shows how to implement a simple zip utility. This sample uses the C1.C1Zip namespace.
ZipImages	Shows how to retrieve images directly from a zip file. This sample calls the C1.C1Zip namespace.

- [Zip for Mobile Devices Samples \(Visual Basic and C#\)](#) 

Sample	Description
DataSet	Add compression to the .NET serialization support. This sample calls the C1.C1Zip namespace.
LongString	Compress and expand strings. This sample calls the C1.C1Zip namespace.
ZipFile	Implement a simple zip utility. This sample calls the C1.C1Zip namespace.

Zip for .NET Frequently Asked Questions

Here are some frequently asked questions (FAQs) about **ComponentOne Zip for .NET**:

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

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

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

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

For more FAQs about **C1Zip**, visit [HelpCentral](#).

Zip for .NET Task-Based Help

The task-based help section assumes that you are familiar with programming in the Visual Studio.NET environment. If you are a novice to **C1Zip**, see the [Zip for .NET Tutorials](#) (page 33) first.

Each task-based help topic provides a solution for specific tasks referencing the C1.C1Zip namespace. For additional information, please see [Adding a Reference to the C1Zip Assembly](#) (page 14). Each topic also assumes that you have created a new .NET project. For more information on this topic, please see [Creating a .NET 2.0 Project](#).

Compressing Datasets

To write a dataset into a compressed zip file, use the following code:

- Visual Basic


```
Private Sub SaveDataSet(ds As DataSet)
    ' Open/create zip file.
```

```

Dim zip As New C1.C1Zip.C1ZipFile()
zip.Open("c:\temp\dataset.zip")

' Write the dataset into the zip file.
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)

    ' Open/create zip file.
    Dim zip As New C1.C1Zip.C1ZipFile()
    zip.Open("c:\temp\dataset.zip")

    ' Read the dataset from the zip file.
    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

    ' Check that the datasets are equivalent.
    Dim i As Integer
    For i = 0 To ds.Tables.Count - 1
        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#**

```

private void SaveDataSet(DataSet ds)
{
    // Open/create zip file.
    C1.C1Zip.C1ZipFile zip = new C1.C1Zip.C1ZipFile();
    zip.Open(@"c:\temp\dataset.zip");

    // Write the dataset into the zip file.
    using (Stream s = zip.Entries.OpenWriter(ds.DataSetName, true))
    {
        ds.WriteXml(s, XmlWriteMode.WriteSchema);
    }
}

private void CheckDataSet(DataSet ds)
{
    // Open/create zip file.
    C1.C1Zip.C1ZipFile zip = new C1.C1Zip.C1ZipFile();
    zip.Open(@"c:\temp\dataset.zip");
}

```

```

// Read the dataset from the zip file.
DataSet dsTest = new DataSet(ds.DataSetName);
using (Stream s = zip.Entries[ds.DataSetName].OpenReader())
{
    dsTest.ReadXml(s);
}

// Check that the datasets are equivalent.
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);
}
}

```

Compressing an Entire Folder into a Zip File

To compress an entire folder into a zip file, preserving the folder structure for later expansion, use the `AddFolder` method.

For example, to compress the contents of the "c:\temp" folder, including all subfolders, use the following code:

- Visual Basic

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

- C#

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

To extract the folder later, including the original folder structure, use the following code:

- Visual Basic

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

- C#

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

Creating a Zip File with Multiple Entries

To stream multiple XML files directly into the zip file, use the `OpenWriter` method. That returns a stream that you can write to, and when the stream is closed it is added to the zip file.

Add the following code to the **Click** event:

- 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")

    ' Continue writing as much as you want...
    sw.Close()
    s = zip.Entries.OpenWriter("entry2", True)

```

```

sw = New StreamWriter(s)
sw.WriteLine("Hello again")

' Continue writing as much as you want...
sw.Close()
End Sub

```

- 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");

    // Continue writing as much as you want...
    sw.Close();
    s = zip.Entries.OpenWriter("entry2", true);
    sw = new StreamWriter(s);
    sw.WriteLine("Hello again");

    // Continue writing as much as you want...
    sw.Close();
}

```

To read an entry without saving it to a file, use the `OpenReader` method on the entry object.

Note: `OpenWriter` is a member of the `C1ZipEntryCollection` class, while `OpenReader` is a member of the `C1ZipEntry` class.

Extracting Files from Zip Entry to Memory

To extract a file from a zip to memory variable (for instance, a `Byte` array), use the following function:

- Visual Basic

```

Private Function GetDataFromZipFile(zipFileName As String, entryName As
String) As Byte()

    ' Get the entry from the zip file.
    Dim zip As New C1ZipFile()
    zip.Open(zipFileName)
    Dim ze As C1ZipEntry = zip.Entries(entryName)

    ' Copy the entry data into a memory stream.
    Dim ms As New MemoryStream()
    Dim buf(1000) As Byte
    Dim s As Stream = ze.OpenReader()
    Try
        While True
            Dim read As Integer = s.Read(buf, 0, buf.Length)
            If read = 0 Then
                Exit While
            End If
            ms.Write(buf, 0, read)
        End While
    Finally

```

```

        s.Dispose()
    End Try
    s.Close()

    ' Return result.
    Return ms.ToArray()
End Function

```

- C#

```

private byte[] GetDataFromZipFile(string zipFileName, string entryName)
{
    // Get the entry from the zip file.
    C1ZipFile zip = new C1ZipFile();
    zip.Open(zipFileName);
    C1ZipEntry ze = zip.Entries[entryName];

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

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

```

Reading a Zipped File Using a StreamReader

To read a zipped file using a **StreamReader**, add the following code:

- Visual Basic

```

' Open a zip file.
Dim zip As New C1ZipFile()
zip.Open("c:\temp\myzipfile.zip")

' Open an input stream on any entry.
Dim ze As C1ZipEntry = zip.Entries("someFile.cs")
Dim s As Stream = ze.OpenReader()

' Open the StreamReader on the stream.
Dim sr As New StreamReader(s)

' Use the StreamReader, then close it.

```

- C#

```

// Open a zip file.

```

```

C1ZipFile zip = new C1ZipFile();
zip.Open(@"c:\temp\myzipfile.zip");

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

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

// Use the StreamReader, then close it.

```

Retrieving Images from a Zip File

To retrieve images directly from a zip file, first add the following code to the **Form_Load** event to compress several image files into a zip file:

- Visual Basic

```

' Build a list of images in resource directory, and add them all to a zip
file.
Dim zip As New C1ZipFile()
Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

    ' Get the application directory.
    Dim s As String = Application.ExecutablePath
    s = s.Substring(0, s.IndexOf("\bin")) + "\resources"

    ' Create the zip file.
    zip.Create((s + "\images.zip"))

    ' Populate the zip file and list.
    Dim f As String
    For Each f In Directory.GetFiles(s)
        Dim fname As String = f.ToLower()

        ' Skip self.
        If fname.EndsWith("zip") Then
            GoTo ContinueForEach1
        End If

        ' Add to the list.
        ListBox1.Items.Add(Path.GetFileName(fname))

        ' Add to the zip file.
        zip.Entries.Add(fname)
        ContinueForEach1:
    Next f
End Sub

```

- C#

```

// Build a list of images in resource directory, and add them all to a zip
file.
C1ZipFile zip = new C1ZipFile();
private void Form1_Load(object sender, System.EventArgs e)
{

```

```

// Get the application directory.
string s = Application.ExecutablePath;
s = s.Substring(0, s.IndexOf(@"\bin")) + @"resources";

// Create the zip file.
zip.Create(s + @"\images.zip");

// Populate the zip file and list.
foreach (string f in Directory.GetFiles(s))
{
    string fname = f.ToLower();

    // Skip self.
    if (fname.EndsWith("zip")) continue;

    // Add to the list.
    listBox1.Items.Add(Path.GetFileName(fname));

    // Add to the zip file.
    zip.Entries.Add(fname);
}
}

```

To allow you to select an image, retrieve a stream with the image data (OpenReader method), and load the image (Image.FromStream method), add the following code to the **SelectedIndexChanged** event:

- Visual Basic

```

' Show selected image.
Private Sub listBox1_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles listBox1.SelectedIndexChanged

    ' Get the selected item.
    Dim item As String = CStr(listBox1.SelectedItem)

    ' Load the image directly from a compressed stream.
    Dim s As Stream = zip.Entries(item).OpenReader()
    Try
        pictureBox1.Image = CType(Image.FromStream(s), Image)
    Catch
    End Try

    ' Done with stream.
    s.Close()
End Sub

```

- C#

```

// Show selected image.
private void listBox1_SelectedIndexChanged(object sender, System.EventArgs
e)
{

    // Get the selected item.
    string item = (string)listBox1.SelectedItem;

    // Load the image directly from a compressed stream.
    Stream s = zip.Entries[item].OpenReader();
    try
    {

```

```

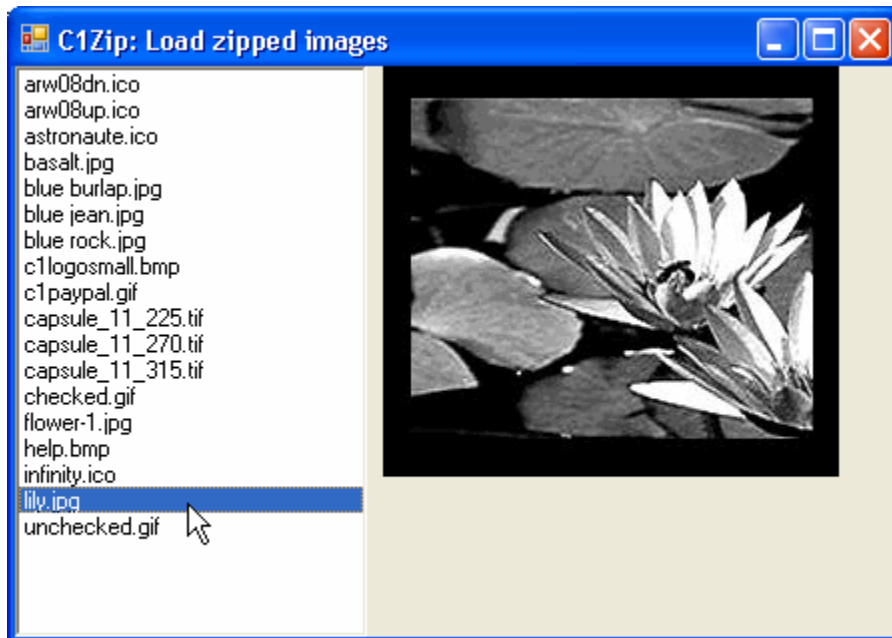
        pictureBox1.Image = (Image) Image.FromStream(s);
    }
    catch {}

    // Done with stream.
    s.Close();
}

```

This topic illustrates the following:

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



Saving a String Variable to a Zip File

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

- **C1ZipEntryCollection.OpenWriter** method

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

- **MemoryStream** method

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

The following code shows both methods. Add the code to the **Button_Click** event:

- **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")

' Method 1: OpenWriter.
Dim stream As Stream = zipFile.Entries.OpenWriter("Shakespeare.txt",
True)
Dim sw As New StreamWriter(stream)
sw.Write(str)
sw.Close()

' Method 2: Memory Stream.
stream = New MemoryStream()
sw = New StreamWriter(stream)
sw.Write(str)
sw.Flush()
stream.Position = 0
zipFile.Entries.Add(stream, "Shakespeare2.txt")
stream.Close()
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");

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

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

```

Setting the Level of Compression

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

- 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#

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

Note that the code sample above sets the compression level to **BestCompression**, which has the highest compression time and the lowest speed. Other compression level options on the C1ZStreamWriter's constructor include the following:

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

Using Passwords to Protect Zip Files

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

- Visual Basic

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

- C#

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

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

- Visual Basic

```
' Will fail, password not set.
C1Zip.Password = ""
C1Zip.Entries.Extract(someFile)

' Will fail, wrong password.
C1Zip.Password = "pass"
C1Zip.Entries.Extract(someFile)

' Will succeed.
C1Zip.Password = "password"
C1Zip.Entries.Extract(someFile)
```

- C#

```
// Will fail, password not set.
C1Zip.Password = "";
C1Zip.Entries.Extract(someFile);

// Will fail, wrong password.
C1Zip.Password = "pass";
C1Zip.Entries.Extract(someFile);

// Will succeed.
C1Zip.Password = "password";
C1Zip.Entries.Extract(someFile);
```

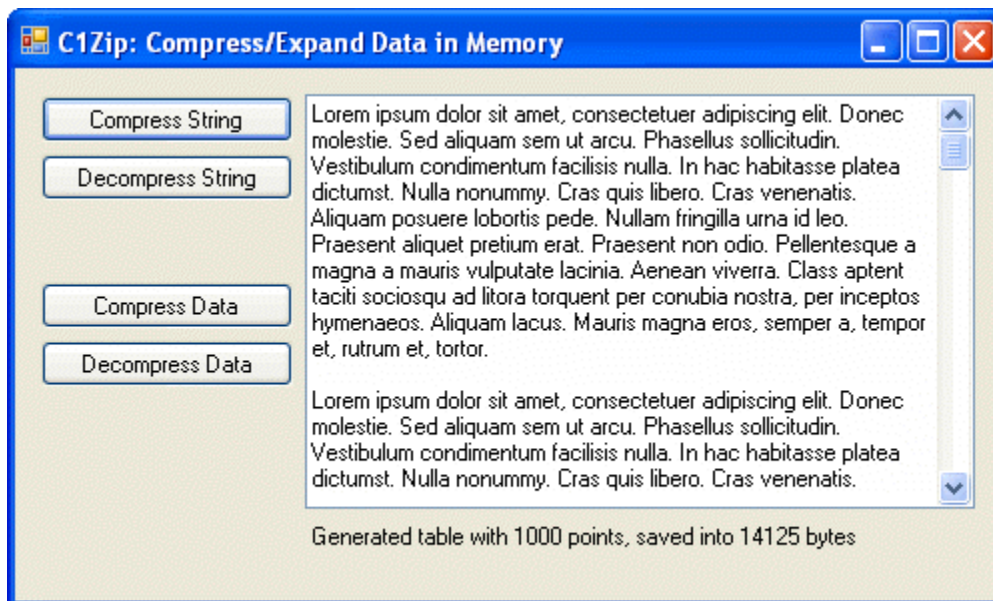
Zip for .NET Tutorials

The following topics contain tutorials that illustrate some of the main features in the **C1Zip** library. The tutorials walk you through the creation of several simple projects, describing each step in detail. See the included [Zip for .NET Samples](#) (page 22) for additional examples that can be used as a reference.

Tutorial	Description
Compressing Data in Memory (page 33)	Shows how you can compress and expand arbitrary data in memory. This technique is useful in any situation where you keep memory streams while the application is running. By compressing the streams, you can reduce the memory requirements for your application.
Compressing Files (page 41)	Shows how you can compress individual files so they take up less disk space and are less exposed to access by users. Note that these are not zip files, just individual compressed files. The last tutorial covers zip files.
Compressed Serialization (page 50)	Shows how you can combine ComponentOne Zip with .NET serialization to save objects into streams that are a fraction of the regular size. If you are serializing objects into XML streams, the savings in disk space and network bandwidth can be huge.
Handling Zip Files (page 56)	Shows how you can open, inspect, add, and remove files in a zip archive. Using the zip format for your application storage needs has several advantages: it is an open-standard, well-documented, space-efficient format.

Compressing Data in Memory

This tutorial shows how you can compress basic data types such as strings and doubles into memory streams, and expand the data back when you read it from the streams. Here is what the final application will look like:



Step 1: Create the main form.

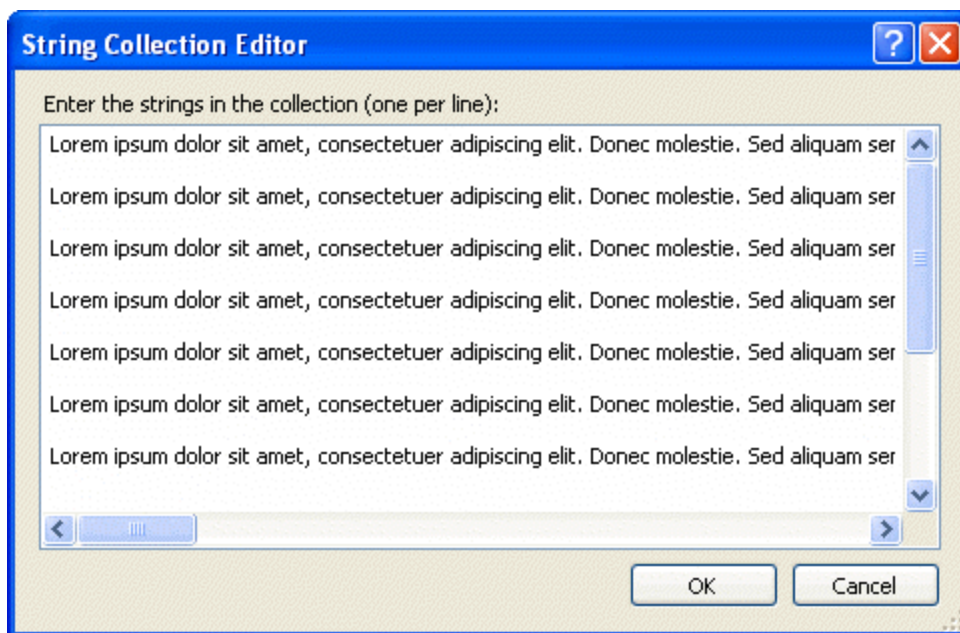
Start a new Visual Studio project, see [Creating a .NET Project](#) (page 12). From the Toolbox, add the following controls to the form, by performing a drag-and-drop operation or by double-clicking the component:

- Four **Button** controls along the left edge of the form, as shown in the picture above. In the Properties window make the following changes to each **Button** control:

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

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

- A **TextBox** on the upper-right of the form. Set the **MultiLine** property to **True**. Select the **ellipsis** button located next to **Lines** property. In the **String Collection Editor** dialog box type text to use as an initial value.



- A **Label** control below the text box.

Step 2: Add a reference to the C1Zip assembly.

Go to the Solution Explorer window and click the **Show All Files** button. Right-click on **References**, and select the **Add Reference** menu option. Select the C1.C1Zip assembly from the list, or browse to find the C1.C1Zip.2.dll file.

Select the **Form1.vb** tab (**Form1.cs** for C#) or go to **View | Code** to open the Code Editor. At the top of the file, add the following statements:

- Visual Basic


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

- C#

```
using System.IO;
using Cl.ClZip;
```

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

See also, [Adding a Reference to the ClZip Assembly](#) (page 14).

Step 3: Add code to compress strings.

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

- Visual Basic

```
Private m_CompressedString As Byte()
Private Sub btnCompressString_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnCompressString.Click

    ' Compress the string.
    Dim ticks As Long = DateTime.Now.Ticks
    m_CompressedString = CompressString(textBox1.Text)

    ' Tell the user how long it took.
    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)

    ' We can now expand it.
    btnExpandString.Enabled = True
End Sub
```

- C#

```
private byte[] m_CompressedString;
private void btnCompressString_Click(object sender, EventArgs e)
{

    // Compress the string.
    long ticks = DateTime.Now.Ticks;
    m_CompressedString = CompressString(textBox1.Text);

    // Tell the user how long it took.
    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);

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

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

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

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

- Visual Basic

```
Public Function CompressString(ByVal str As String) As Byte()  
  
    ' Open the memory stream.  
    Dim ms As MemoryStream = New MemoryStream()  
  
    ' Attach a compressor stream to the memory stream.  
    Dim sw As ClzStreamWriter = New ClzStreamWriter(ms)  
  
    ' Write the data into the compressor stream.  
    Dim writer As StreamWriter = New StreamWriter(sw)  
    writer.Write(str)  
  
    ' Flush any pending data.  
    writer.Flush()  
  
    ' Return the memory buffer.  
    CompressString = ms.ToArray()  
End Function
```

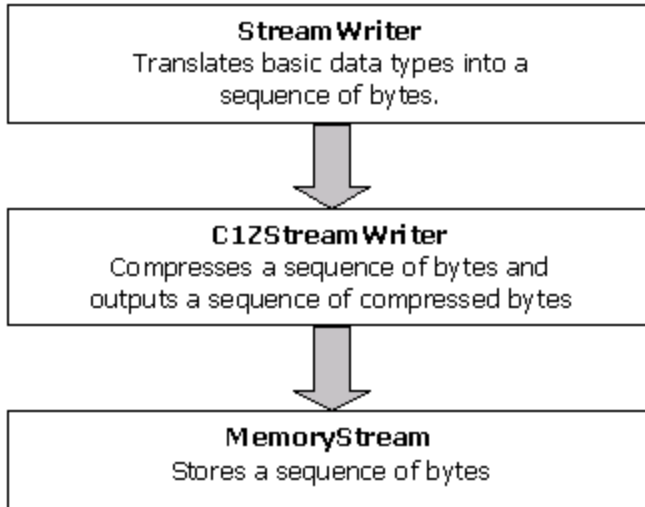
- C#

```
public byte[] CompressString(string str)  
{  
  
    // Open the memory stream.  
    MemoryStream ms = new MemoryStream();  
  
    // Attach a compressor stream to the memory stream.  
    ClzStreamWriter sw = new ClzStreamWriter(ms);  
  
    // Write the data into the compressor stream.  
    StreamWriter writer = new StreamWriter(sw);  
    writer.Write(str);  
  
    // Flush any pending data.  
    writer.Flush();  
  
    // Return the memory buffer.  
    return ms.ToArray();  
}
```

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

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

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



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

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

Step 4: Add code to expand strings.

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

- Visual Basic

```

Private Sub btnExpandString_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnExpandString.Click

    ' Expand the string.
    Dim ticks As Long = DateTime.Now.Ticks
    TextBox1.Text = ExpandString(m_CompressedString)

    ' Tell the user how long it took.
    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, EventArgs e)
{

    // Expand the string.
    long ticks = DateTime.Now.Ticks;
    textBox1.Text = ExpandString(m_CompressedString);

    // Tell the user how long it took.
    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);
}

```

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

- Visual Basic

```

Public Function ExpandString(ByVal buffer As Byte()) As String

    ' Turn buffer into a memory stream.
    Dim ms As MemoryStream = New MemoryStream(buffer)

    ' Attach a decompressor stream to the memory stream.
    Dim sr As ClzStreamReader = New ClzStreamReader(ms)

    ' Read uncompressed data.
    Dim reader As StreamReader = New StreamReader(sr)
    ExpandString = reader.ReadToEnd()
End Function

```

- C#

```

public string ExpandString(byte[] buffer)
{

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

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

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

```

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

Step 5: Add code to compress binary data.

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

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

- Visual Basic

```

Private m_CompressedData As Byte()
Private Sub btnCompressData_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnCompressData.Click

    ' Open the memory stream.
    Dim ms As MemoryStream = New MemoryStream()

```

```

' Attach a compressor stream to the memory stream.
Dim sw As ClzStreamWriter = New ClzStreamWriter(ms)

' Attach a BinaryWriter to the compressor stream.
Dim bw As BinaryWriter = New BinaryWriter(sw)

' Write a bunch of numbers into the stream.
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

' Flush any pending output.
bw.Flush()

' Save the compressed data.
m_CompressedData = ms.ToArray()

' Done.
Dim msg As String
msg =String.Format("Generated table with {0} points," & " saved into
{1} bytes", count, m_CompressedData.Length)
Labell.Text = msg

' We can now expand it.
btnExpandData.Enabled = True
End Sub

```

- **C#**

```

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

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

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

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

```

```

        bw.Write(Math.Sin(a));
        bw.Write(Math.Cos(a));
    }

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

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

    // Done.
    string msg;
    msg = string.Format("Generated table with {0} points," +
        " saved into {1} bytes", count, m_CompressedData.Length);
    label1.Text = msg;

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

```

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

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

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

Step 6: Add code to expand the binary data.

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

Add the following **Click** event handler code for the **Decompress Data** command button:

- Visual Basic

```

Private Sub btnExpandData_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnExpandData.Click

    ' Open the memory stream on saved data.
    Dim ms As MemoryStream = New MemoryStream(m_CompressedData)

    ' Attach a decompressor stream to the memory stream.
    Dim sr As ClzStreamReader = New ClzStreamReader(ms)

    ' Read the uncompressed data.
    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

    ' Done, tell the user about it.
    Dim msg As String

```

```

    msg = String.Format("Read table with {0} points " & "from stream with
{1} bytes.", count, m_CompressedData.Length)
    Labell.Text = msg
End Sub

```

- **C#**

```

private void btnExpandData_Click(object sender, EventArgs e)
{
    // Open the memory stream on saved data.
    MemoryStream ms = new MemoryStream(m_CompressedData);

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

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

    // Done, tell the user about it.
    string msg;
    msg = string.Format("Read table with {0} points " +
        "from stream with {1} bytes.", count, m_CompressedData.Length);
    labell.Text = msg;
}

```

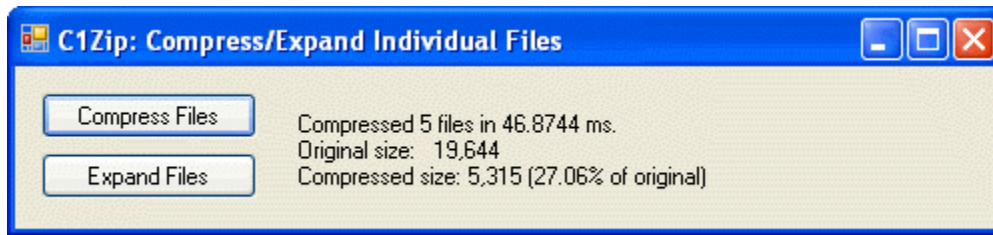
The code reads the data but does not display it. You can step through it in debug mode to make sure the data being read is the same that was written in.

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

This concludes the Compressing Data in Memory tutorial.

Compressing Files

This tutorial shows how you can compress and expand individual files. Note that these are not zip files; they are just compressed streams on disk. Zip files are covered in the [Handling Zip Files](#) (page 56) tutorial. Here is what the final application will look like:



Step 1: Create the main form.

Start a new Visual Studio project and from the Toolbox, add the following controls to the form:

- Two **Button** controls along the left edge of the form, as shown in the picture above. In the Properties window make the following changes:

Button	Button.Text Property	Button.Name Property	Button.Enabled Property
1	Compress Files	btnCompress	True (Default)
2	Expand Files	btnExpand	False

Note that the **Expand Files** button cannot be used until we have some compressed files to expand.

- A **Label** control on the right of the buttons. This control will display statistics about the compression/expanding process.

Step 2: Add a reference to the C1Zip assembly.

Go to the Solution Explorer window and click the **Show All Files** button. Right-click on **References**, and select the **Add Reference** menu option. Select the C1.C1Zip assembly from the list, or browse to find the C1.C1Zip.2.dll file.

Select the **Form1.vb** tab (**Form1.cs** in C#) or go to **View | Code** to open the Code Editor. At the top of the file, add the following statements:

- Visual Basic


```
Imports System.IO
Imports C1.C1Zip
```
- C#


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

This makes the objects defined in the **C1Zip** and **System.IO** assemblies visible to the project and saves a lot of typing.

See also, [Adding a Reference to the C1Zip Assembly](#) (page 14).

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

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

- Visual Basic


```
Private Const DIR_COMP = "\compressed"
Private Const DIR_EXP = "\expanded"
```
- C#


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

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

Step 4: Add code to compress files.

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

- Visual Basic

```
Private Sub btnCompress_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCompress.Click

    ' Get the application directory.
    Dim appPath As String = Application.ExecutablePath
    Dim i As Integer = appPath.IndexOf("\bin\")
    If i > 0 Then appPath = appPath.Substring(0, i)

    ' Create a directory for compressed files.
    If (Directory.Exists(appPath + DIR_COMP)) Then
        Directory.Delete(appPath + DIR_COMP, True)
    End If
    Directory.CreateDirectory(appPath + DIR_COMP)

    ' Prepare to collect compression statistics.
    Dim count As Long
    Dim size As Long
    Dim sizeCompressed As Long
    Dim ticks As Long = DateTime.Now.Ticks

    ' Compress all files in the application dir into the compressed dir.
    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"

        ' Compress file.
        CompressFile(dstFile, srcFile)

        ' Update stats.
        count = count + 1
        size = size + New FileInfo(srcFile).Length
        sizeCompressed = sizeCompressed + New FileInfo(dstFile).Length
    Next srcFile

    ' Show stats.
    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)
    Labell1.Text = msg

    ' Now we can expand.
    btnExpand.Enabled = True
End Sub
```

- C#

```
private void btnCompress_Click(object sender, EventArgs e)
{
```

```

// Get the application directory.
string appPath = Application.ExecutablePath;
int i = appPath.IndexOf(@"\bin\");
if (i > 0) appPath = appPath.Substring(0, i);

// Create a directory for compressed files.
if ((Directory.Exists(appPath + DIR_COMP)))
    Directory.Delete(appPath + DIR_COMP, true);
Directory.CreateDirectory(appPath + DIR_COMP);

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

// Compress all files in the application dir into the compressed dir.
foreach (string srcFile in Directory.GetFiles(appPath))
{
    string dstFile = appPath + DIR_COMP + Path.GetFileName(srcFile) +
".cmp";

    // Compress file.
    CompressFile(dstFile, srcFile);

    // Update stats.
    count++;
    size += new FileInfo(srcFile).Length;
    sizeCompressed += new FileInfo(dstFile).Length;
}

// Show stats.
string msg = string.Format("Compressed {0} files in {1} ms.\n\r" +
"Original size:  {2:#,###}\n\r" + "Compressed size: {3:#,###} ({4:0.00}%
of original)", count, (DateTime.Now.Ticks - ticks) /
(TimeSpan.TicksPerMillisecond, size, sizeCompressed, (sizeCompressed /
size) * 100.0);
labell1.Text = msg;

// Now we can expand.
btnExpand.Enabled = true;
}

```

The main line calls the utility function **CompressFile** utility method to compress each selected file. The compressed files are stored in the \compressed directory under the application folder. They have the same name as the original file, plus a CMP extension.

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

- Visual Basic

```

Private Function CompressFile( dstFile As String, srcFile As String) As
Boolean

    ' Prepare to compress file.
    Dim retVal As Boolean = True
    Dim srcStream As FileStream = Nothing
    Dim dstStream As FileStream = Nothing
    Try

```

```

        ' Open the files.
        srcStream = New FileStream(srcFile, FileMode.Open,
FileAccess.Read)
        dstStream = New FileStream(dstFile, FileMode.Create,
FileAccess.Write)

        ' Open a compressor stream on the destination file.
        Dim sw As ClzStreamWriter = New ClzStreamWriter(dstStream)

        ' Copy the source into the compressor stream.
        StreamCopy(sw, srcStream)
    Catch

        ' Exception? Tell the caller we failed.
        retval = False

    Finally

        ' Always close our streams.
        If Not (srcStream Is Nothing) Then srcStream.Close()
        If Not (dstStream Is Nothing) Then dstStream.Close()
    End Try

    ' Done.
    CompressFile = False
End Function

```

- **C#**

```

private bool CompressFile(string dstFile, string srcFile)
{

    // Prepare to compress file.
    bool retval = true;
    FileStream srcStream = null;
    FileStream dstStream = null;
    try
    {

        // Open the files.
        srcStream = new FileStream(srcFile, FileMode.Open,
FileAccess.Read);
        dstStream = new FileStream(dstFile, FileMode.Create,
FileAccess.Write);

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

        // Copy the source into the compressor stream.
        StreamCopy(sw, srcStream);
    }
    catch
    {

        // Exception? Tell the caller we failed.
        retval = false;
    }
}

```

```

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

// Done.
return false;
}

```

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

Finally, the function closes both streams. Note the use of the **Finally** statement to ensure that both streams are properly closed even if there are exceptions while the function is executing.

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

- 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#

```

private void StreamCopy(Stream dstStream, Stream srcStream)
{
    byte[] buffer= new byte[32768];
    for (;;)
    {
        int read = srcStream.Read(buffer, 0, buffer.Length);
        if (read == 0) break;
        dstStream.Write(buffer, 0, read);
    }
    dstStream.Flush();
}

```

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

Step 5: Add code to expand files.

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

- Visual Basic

```

Private Sub btnExpand_Click(sender As Object, e As EventArgs) Handles
    btnExpand.Click

    ' Get the application directory.
    Dim appPath As String = Application.ExecutablePath
    Dim i As Integer      = appPath.IndexOf("\bin\")
    If i > 0 Then appPath = appPath.Substring(0, i)

```

```

' Create a directory for expanded files.
If Directory.Exists(appPath + DIR_EXP) Then
    Directory.Delete(appPath + DIR_EXP, True)
End If
Directory.CreateDirectory(appPath + DIR_EXP)

' Prepare to collect compression statistics.
Dim count As Long
Dim size As Long
Dim sizeExpanded As Long
Dim ticks As Long = DateTime.Now.Ticks

' Expand all files in the "compressed" dir to the "expanded" dir.
Dim srcFile As String
Dim files As String()
files = Directory.GetFiles(appPath + DIR_COMP)
For Each srcFile In files

    ' Expand file.
    Dim dstFile As String = appPath + DIR_EXP + "\" +
Path.GetFileName(srcFile)
    dstFile = dstFile.Replace(".cmp", "")
    ExpandFile(dstFile, srcFile)

    ' Update stats.
    count = count + 1
    size = size + New FileInfo(srcFile).Length
    sizeExpanded = sizeExpanded + New FileInfo(dstFile).Length
Next srcFile

' Show stats.
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)
Labell.Text = msg
End Sub

```

- **C#**

```

private void btnExpand_Click(object sender, EventArgs e)
{
    // Get the application directory.
    string appPath = Application.ExecutablePath;
    int i = appPath.IndexOf(@"\bin\");
    if (i > 0) appPath = appPath.Substring(0, i);

    // Create a directory for expanded files.
    if (Directory.Exists(appPath + DIR_EXP))
        Directory.Delete(appPath + DIR_EXP, true);
    Directory.CreateDirectory(appPath + DIR_EXP);

    // Prepare to collect compression statistics.
    long count = 0;
    long size = 0;

```

```

long sizeExpanded = 0;
long ticks = DateTime.Now.Ticks;

// Expand all files in the "compressed" dir to the "expanded" dir.
foreach (string srcFile in Directory.GetFiles(appPath + DIR_COMP))
{
    // Expand file.
    string dstFile = appPath + DIR_EXP + @"\" +
Path.GetFileName(srcFile);
    dstFile = dstFile.Replace(".cmp", "");
    ExpandFile(dstFile, srcFile);

    // Update stats.
    count++;
    size += new FileInfo(srcFile).Length;
    sizeExpanded += new FileInfo(dstFile).Length;
}

// Show stats.
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);
labell1.Text = msg;
}

```

The main line calls the utility function **ExpandFile** utility method to expand the files that were compressed earlier. The expanded files are stored in the \expanded directory under the application folder. They have the same name as the original file, minus the CMP extension.

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

- Visual Basic

```

Private Function ExpandFile(dstFile As String, srcFile As String) As
Boolean

    ' Prepare to expand file.
    Dim retval As Boolean = True
    Dim srcStream As FileStream = Nothing
    Dim dstStream As FileStream = Nothing
    Try

        ' Open the files.
        srcStream = New FileStream(srcFile, FileMode.Open,
FileAccess.Read)
        dstStream = New FileStream(dstFile, FileMode.Create,
FileAccess.Write)

        ' Open an expander stream on the compressed source.
        Dim sr As ClzStreamReader = New ClzStreamReader(srcStream)

        ' Copy the expander stream into the destination file.
        StreamCopy(dstStream, sr)
    Catch

        ' Exception? Tell the caller we failed.
        retval = False
    End Try
End Function

```

```

Finally
    ' Always close our streams.
    If Not (srcStream Is Nothing) Then srcStream.Close()
    If Not (dstStream Is Nothing) Then dstStream.Close()
End Try

' Done.
ExpandFile = retval
End Sub

```

- **C#**

```

private bool ExpandFile(string dstFile, string srcFile)
{
    // Prepare to expand file.
    bool retval = true;
    FileStream srcStream = null;
    FileStream dstStream = null;
    try
    {
        // Open the files.
        srcStream = new FileStream(srcFile, FileMode.Open,
FileAccess.Read);
        dstStream = new FileStream(dstFile, FileMode.Create,
FileAccess.Write);

        // Open an expander stream on the compressed source.
        C1ZStreamReader sr = new C1ZStreamReader(srcStream);

        // Copy the expander stream into the destination file.
        StreamCopy(dstStream, sr);
    }
    catch
    {
        // Exception? Tell the caller we failed.
        retval = false;
    }
    finally
    {
        // Always close our streams.
        if (srcStream != null) srcStream.Close();
        if (dstStream != null) dstStream.Close();
    }

    // Done.
    return retval;
}

```

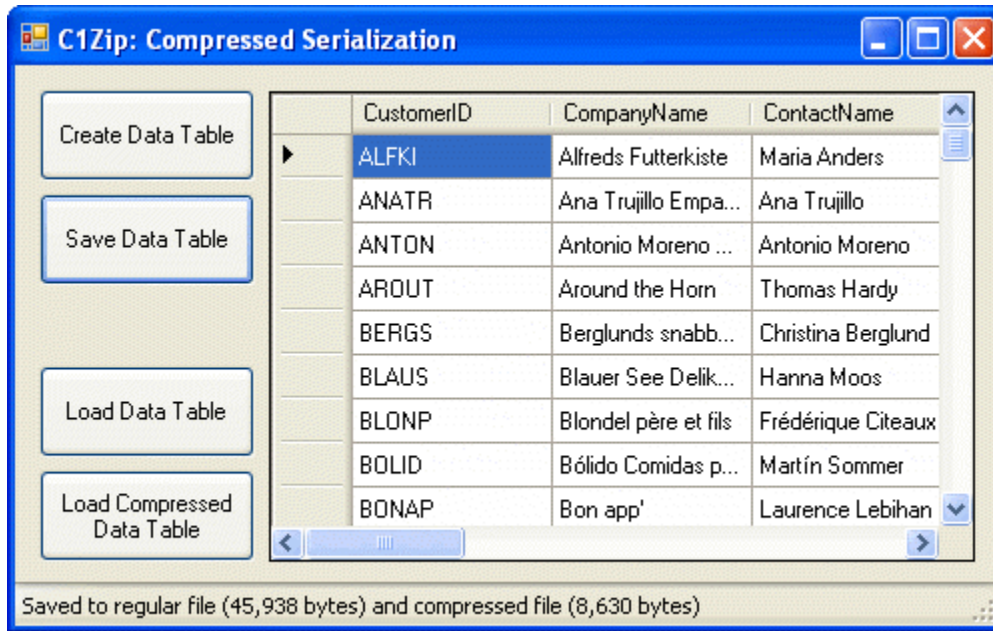
The function is similar to **CompressFile**, except it attaches a **C1ZStreamReader** to the source stream instead of attaching a **C1ZStreamWriter** to the destination stream.

This concludes the Compressing Files tutorial.

Compressed Serialization

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

The sample creates a data table using the NorthWind database. The table is saved (serialized) into regular and compressed streams. Finally, the data is loaded back from either stream. Here is what the final application will look like:



Step 1: Create the main form.

Start a new Visual Studio project and from the Toolbox, add the following controls to the form:

- Four **Button** controls along the left edge of the form, as shown in the picture above. In the Properties window make the following changes to each **Button** control:

Button	Button.Text Property	Button.Name Property	Button.Enabled Property
1	Create Data Table	btnCreate	True (Default)
2	Save Data Table	btnSave	False
3	Load Data Table	btnLoad	False
4	Load Compressed Data Table	btnLoadCompressed	False

Note that the save and load buttons cannot be used until the data table has been created or saved.

- A **DataGridView** control on the right of the form.
- A **ToolStripStatusLabel** control docked at the bottom of the form. To add this control, first add a **StatusStrip** control to the form. Then click the **Add ToolStripStatusLabel** drop-down arrow and select **StatusLabel**. A **ToolStripStatusLabel** control appears and is docked at the bottom of the form.

Step 2: Add references and Imports statements.

Go to the Solution Explorer window and click the **Show All Files** button. Right-click on References, and select the **Add Reference** menu option. Select the C1.C1Zip assembly from the list, or browse to find the C1.C1Zip.2.dll file.

Select the **Form1.vb** tab (**Form1.cs** in C#) or go to **View | Code** to open the Code Editor. At the top of the file, add the following statements:

- Visual Basic

```
Imports System.IO
Imports System.Data.OleDb
Imports System.Runtime.Serialization.Formatters.Binary
Imports Cl.ClZip
```

- C#

```
using System.IO;
using System.Data.OleDb;
using System.Runtime.Serialization.Formatters.Binary;
using Cl.ClZip;
```

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

See also, [Adding a Reference to the ClZip Assembly](#) (page 14).

Step 3: Declare constants.

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

- 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#

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

These constants define the name of the database used to populate the data table and the file names used to serialize the data.

Step 4: Add code to create the data table.

Add the following code to handle the **Click** event for the **Create Data Table** button:

- Visual Basic

```
Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles
btnCreate.Click

    ' Open the table.
    Dim conn As String
    conn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" & MDBFILE & ";";
    Dim rs As String = "select * from customers"

    ' Show status.
    Cursor = Cursors.WaitCursor
    ToolStripStatusLabel1.Text = "Loading data from mdb file..."

    ' Load data.
    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

' Show status.
Cursor = Cursors.Default
ToolStripStatusLabel1.Text = "Loaded " & ds.Tables(0).Rows.Count & "
records from mdb file."

' Bind to the grid.
DataGridView1.DataSource = ds.Tables(0)

' Enable the save button.
btnSave.Enabled = True
End Sub

```

- **C#**

```

private void btnCreate_Click(object sender, EventArgs e)
{
    // Open the table.
    string conn = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" +
MDBFILE + ";";
    string rs = "select * from customers";

    // Show status.
    Cursor = Cursors.WaitCursor;
    toolStripStatusLabel1.Text = "Loading data from mdb file...";

    // Load data.
    OleDbDataAdapter da = new OleDbDataAdapter(rs, conn);
    DataSet ds = new DataSet();
    try
    {
        da.Fill(ds);
    }
    catch
    {
        MessageBox.Show("Could not load data from " + MDBFILE);
    }

    // Show status.
    Cursor = Cursors.Default;
    toolStripStatusLabel1.Text = "Loaded " + ds.Tables[0].Rows.Count + "
records from mdb file.";

    // Bind to the grid.
    dataGridView1.DataSource = ds.Tables[0];

    // Enable the save button.
    btnSave.Enabled = true;
}

```

The function uses standard ADO.NET objects and methods to create and populate a **DataTable** object, which is then bound to the **DataGrid** control.

Step 5: Add code to save the data table.

Add the following code to handle the **Click** event for the **Save Data Table** button:

- Visual Basic

```

Private Sub btnSave_Click(sender As Object, e As EventArgs) Handles
btnSave.Click

    ' Get the data table from the grid.
    Dim dt As DataTable = DataGridView1.DataSource

    ' Show status.
    Cursor = Cursors.WaitCursor
    ToolStripStatusLabel1.Text = "Serializing data to regular file..."

    ' Serialize the data set to a regular file.
    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()

    ' Show status.
    Cursor = Cursors.WaitCursor
    ToolStripStatusLabel1.Text = "Serializing data to compressed file..."

    ' Serialize the data set to a compressed file.
    fn = Application.StartupPath & FN_COMPRESSED
    fs = New FileStream(fn, FileMode.Create)
    Dim compressor As GZipStreamWriter = New GZipStreamWriter(fs)
    bf = New BinaryFormatter()
    bf.Serialize(compressor, dt)
    Dim lenCompressed As Long = fs.Length
    fs.Close()

    ' Show status.
    Cursor = Cursors.Default
    ToolStripStatusLabel1.Text = string.Format( "Saved to regular file
({0:#,###} bytes) and " & "compressed file ({1:#,###} bytes)", lenRegular,
lenCompressed)

    ' Enable the load buttons.
    btnLoad.Enabled = True
    btnLoadCompressed.Enabled = True
End Sub

```

- **C#**

```

private void btnSave_Click(object sender, EventArgs e)
{

    // Get the data table from the grid.
    DataTable dt = (DataTable)dataGridView1.DataSource;

    // Show status.
    Cursor = Cursors.WaitCursor;
    toolStripStatusLabel1.Text = "Serializing data to regular file...";

    // Serialize the data set to a regular file.
    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();

// Show status.
Cursor = Cursors.WaitCursor;
toolStripStatusLabel1.Text = "Serializing data to compressed file...";

// Serialize the data set to a compressed file.
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();

// Show status.
Cursor = Cursors.Default;
toolStripStatusLabel1.Text = string.Format("Saved to regular file
({0:#,###} bytes) and " + "compressed file ({1:#,###} bytes)", lenRegular,
lenCompressed);

// Enable the load buttons.
btnLoad.Enabled = true;
btnLoadCompressed.Enabled = true;
}

```

The first set of code serializes the **DataTable** into a regular file, and the second serializes the **DataTable** into a compressed file. Note that only one additional line is required to compress the data.

In both cases, the serialization is executed by the **BinaryFormatter** object. The only difference is that in the first case, the **Serialize** method is called with a regular file stream as a parameter; in the second, a **C1ZStreamWriter** is used instead.

Step 6: Add code to load the data table from the regular file.

Add the following code to handle the **Click** event for the **Load Data Table** button:

- Visual Basic

```

Private Sub btnLoad Click(sender As Object, e As EventArgs) Handles
btnLoad.Click

    ' Clear grid, show status.
    Cursor = Cursors.WaitCursor
    DataGridView1.DataSource = Nothing
    ToolStripStatusLabel1.Text = "Loading from regular file..."

    ' Deserialize from regular file.
    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()

    ' Show result.

```

```

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)
{
    // Clear grid, show status.
    Cursor = Cursors.WaitCursor;
    dataGridView1.DataSource = null;
    toolStripStatusLabel1.Text = "Loading from regular file...";

    // Deserialize from regular file.
    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();

    // Show result.
    Cursor = Cursors.Default;
    dataGridView1.DataSource = dt;
    toolStripStatusLabel1.Text = "Loaded from regular file in " +
ms.ToString() + " ms.";
}

```

The first main line of code creates a new **BinaryFormatter** object and the second one calls its **Deserialize** method. The **Deserialize** method takes a single parameter: the stream in which the object is defined. In this case, the stream is a regular file stream.

Step 7: Add code to load the data table from the compressed file.

Add the following code to handle the **Click** event for the **Load Compressed Data Table** button:

- **Visual Basic**

```

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

    ' Clear grid, show status.
    Cursor = Cursors.WaitCursor
    DataGridView1.DataSource = Nothing
    ToolStripStatusLabel1.Text = "Loading from compressed file..."

    ' Deserialize from compressed file.
    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()

```

```

    ' Show result.
    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)
{
    // Clear grid, show status.
    Cursor = Cursors.WaitCursor;
    dataGridView1.DataSource = null;
    toolStripStatusLabel1.Text = "Loading from compressed file...";

    // Deserialize from compressed file.
    string fn = Application.StartupPath + FN_COMPRESSED;
    FileStream fs = new FileStream(fn, FileMode.Open);
    long ticks = DateTime.Now.Ticks;
    ClzStreamReader decompressor;
    decompressor = new ClzStreamReader(fs);
    BinaryFormatter bf = new BinaryFormatter();
    DataTable dt = (DataTable)bf.Deserialize(decompressor);
    long ms = (DateTime.Now.Ticks - ticks) / TimeSpan.TicksPerMillisecond;
    fs.Close();

    // Show result.
    Cursor = Cursors.Default;
    dataGridView1.DataSource = dt;
    toolStripStatusLabel1.Text = "Loaded from compressed file in " +
ms.ToString() + " ms.";
}

```

The main lines are similar to the code used to deserialize data from a regular file. The only difference is that instead of passing a regular file stream to the **Deserialize** method, we now use a **ClzStreamReader** object.

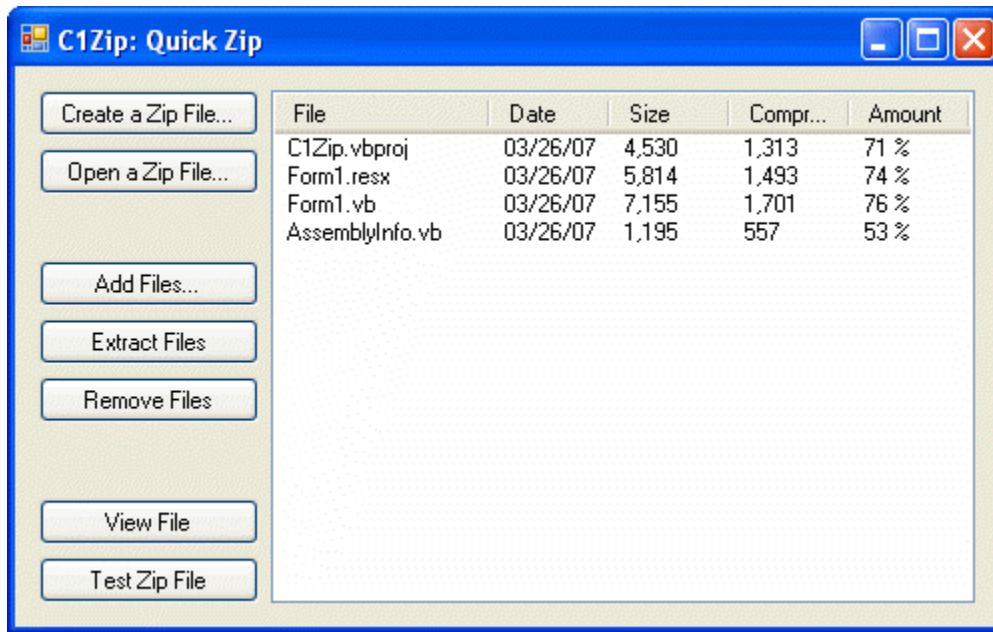
This concludes the Compressed Serialization tutorial.

Handling Zip Files

This tutorial shows how you can handle zip files, including the following operations:

- Creating a zip file.
- Opening an existing zip file.
- Adding entries to the zip file.
- Extracting entries from the zip file.
- Removing entries from the zip file.
- Expanding entries into memory streams.
- Testing the integrity of a zip file.

Here is what the final application will look like:



Note: This is a very simple application, designed to highlight the main functionality of the C1ZipFile object. The distribution disk also includes a more sophisticated version, which supports more advanced features such as drag and drop, zipping folders, setting the compression level, and so on.

Step 1: Create the main form.

Start a new Visual Studio project.

- From the Toolbox, add the following controls to the form:
 - Seven **Button** controls along the left edge of the form, as shown in the picture above. In the Properties window make the following changes to each **Button** control:

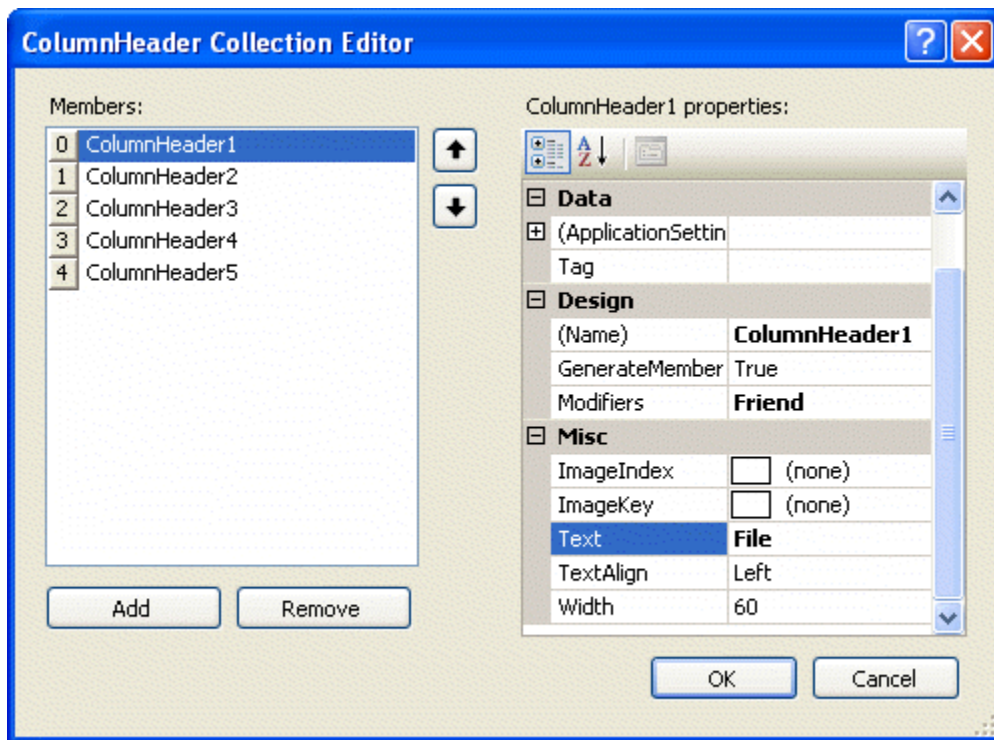
Button	Button.Text Property	Button.Name Property
1	Create Zip File...	btnNew
2	Open Zip File...	btnOpen
3	Add Files...	btnAdd
4	Extract Files	btnExtract
5	Remove Files	btnRemove
6	View File	btnView
7	Test Zip File	btnTest

- A **ListView** control covering the right part of the form. In the Properties window, click on the **ellipsis** button next to the **Columns** property. The **ColumnHeader Collection Editor** dialog box appears.

Note: The **Column Header Collection Editor** can also be accessed by clicking **Edit Columns** in the **ListView Tasks** menu. To access the **ListView Tasks** menu, click the smart tag (🔗) in the upper right corner of the **ListView** control.

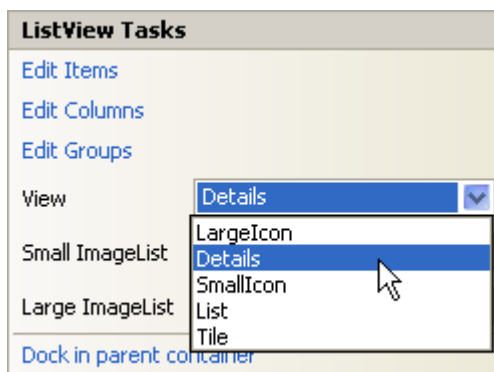
- Click the **Add** button to add five ColumnHeaders, and then set the **Text** property of each column to **File**, **Date**, **Size**, **Compressed**, and **Amount**, respectively.

Here is what the **ColumnHeader Collection Editor** should look like:



3. Select **OK** to close the editor box.

Note that to view the column headers, in the Properties window set the **View** property to **Details** or in the **List View Tasks** menu, select **Details** from the **View** drop-down box.



Step 2: Add a reference to the C1Zip assembly.

Go to the Solution Explorer window, right-click on References, and select the **Add Reference** menu option. Select the C1.C1Zip assembly from the list, or browse to find the C1.C1Zip.2.dll file.

Select the **Form1.vb** tab (**Form1.cs** in C#) or go to **View | Code** to open the Code Editor. At the top of the file, add the following directives:

- Visual Basic


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

- C#

```
using System.IO;
using Cl.ClZip;
```

This makes the objects defined in the ClZip and System.IO assemblies visible to the project.

See also, [Adding a Reference to the ClZip Assembly](#) (page 14).

Step 3: Declare a ClZipFile object.

Switch to the Code Editor and type (or copy) the following data member declaration:

- Visual Basic

```
Private m_Zip As ClZipFile
```

- C#

```
private ClZipFile m_Zip;
```

This is the main object in this application. It implements the methods used to handle zip files.

Step 4: Add code to initialize the ClZipFile object.

Add the following code to the **Form_Load** event:

- Visual Basic

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

    ' Create the ClZipFile member.
    m_Zip = New ClZipFile()
End Sub
```

- C#

```
private void Form1_Load(object sender, EventArgs e)
{

    // Create the ClZipFile member.
    m_Zip = new ClZipFile();
}
```

This code creates a new ClZipFile object and assigns it to the **m_Zip** member. Note that the ClZipFile object cannot be used yet. It needs to be attached to an existing zip file first using the Open method (or to a new one using the Create method).

Step 5: Add code to create a zip file.

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

- Visual Basic

```
Private Sub btnNew_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnNew.Click

    ' Show open file dialog.
    Dim fo As SaveFileDialog = New SaveFileDialog()
    fo.FileName = "*.zip"
    If fo.ShowDialog() <> Windows.Forms.DialogResult.OK Then
        Exit Sub
    End If

    ' Open zip file.
    Try
        m_Zip.Create(fo.FileName)
    Catch
```

```

        MessageBox.Show("Can't create ZIP file, please try again.",
"C1Zip")
    End Try

    ' Update display.
    UpdateDisplay()
End Sub

```

- **C#**

```

private void btnNew_Click(object sender, EventArgs e)
{

    // Show open file dialog.
    SaveFileDialog fo = new SaveFileDialog();
    fo.FileName = "*.zip";
    if (fo.ShowDialog() != DialogResult.OK ) return;

    // Open zip file.
    try
    {
        m_Zip.Create(fo.FileName);
    }
    catch
    {
        MessageBox.Show("Can't create ZIP file, please try again.",
"C1Zip");
    }

    // Update display.
    UpdateDisplay();
}

```

Step 6: Add code to open a zip file.

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

- **Visual Basic**

```

Private Sub btnOpen_Click(sender As Object, e As EventArgs) Handles
btnOpen.Click

    ' Show open file dialog.
    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

    ' Update display.
    UpdateDisplay()
End Sub

```

- **C#**

```

private void btnOpen_Click(object sender, EventArgs e)
{

    // Show open file dialog.

```

```

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.");
}

// Update display.
UpdateDisplay();
}

```

The main line calls the `Open` method on the `C1ZipFile` object to attach the object to an existing zip file. Note the use of a **Try/Catch** statement, mainly to handle situations where the user selects a file that is not a zip file.

After opening the file, the code calls the **UpdateDisplay** utility function to display the contents of the zip file.

Step 7: Add code to update the display.

Next, add the **UpdateDisplay** utility function to display the contents of the zip file:

- Visual Basic

```

Private Sub UpdateDisplay()

    ' Update the ListView control to show the zip file contents.
    With ListView1

        ' Remove any existing items.
        .Items.Clear()

        ' Add each entry.
        Dim ze As C1ZipEntry
        For Each ze In m_Zip.Entries

            ' Calculate the compression amount.
            Dim pct As Double = 0
            If ze.SizeUncompressed > 0 Then
                pct = 1 - ze.SizeCompressed / ze.SizeUncompressed
            End If

            ' Build ListView item.
            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)

            ' Save ZipEntry into item tag.
            lvi.Tag = ze

            ' Add item to ListView.
            .Items.Add(lvi)
        Next ze
    End With
End Sub

```

```

        ' Update UI.
        Dim hasEntries As Boolean = (.Items.Count > 0)
        btnExtract.Enabled = hasEntries
        btnRemove.Enabled = hasEntries
        btnTest.Enabled = hasEntries
    End With
End Sub

```

- C#

```

private void UpdateDisplay()
{
    // Remove any existing items.
    listView1.Items.Clear();

    // Add each entry.
    foreach (C1ZipEntry ze in m_Zip.Entries)
    {
        // Calculate the compression amount.
        double pct = 0;
        if (ze.SizeUncompressed > 0)
        {
            pct = 1 - (((double)ze.SizeCompressed) /
                ((double)ze.SizeUncompressed));
        }

        // Build ListViewItem.
        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 %") });

        // Save ZipEntry into item tag.
        lvi.Tag = ze;

        // Add item to ListView.
        listView1.Items.Add(lvi);
    }

    // Update UI.
    bool hasEntries = (listView1.Items.Count > 0);
    btnExtract.Enabled = hasEntries;
    btnRemove.Enabled = hasEntries;
    btnTest.Enabled = hasEntries;
}

```

The main lines start by declaring a `C1ZipEntry` object and using it in a **ForEach** loop over the entries in the zip file (`m_Zip.Entries`).

For each entry, the function creates a **ListViewItem** containing the information extracted from the `C1ZipEntry` object and adds the new entry to the **ListView** control. Note that the `C1ZipEntry` object itself is also saved in the **Tag** property of the `ListView` items.

Finally, the code enables or disables the command buttons depending on whether or not there are entries available to be extracted, removed or tested.

Step 8: Add code to test the integrity the zip file.

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

- Visual Basic

```
Private Sub btnTest_Click(sender As Object, e As EventArgs) Handles
btnTest.Click

    ' Test each entry.
    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

    ' If we got here, everything is OK.
    MessageBox.Show("All entries passed CRC check", "C1Zip")
End Sub
```

- C#

```
private void btnTest_Click(object sender, EventArgs e)
{

    // Test each entry.
    foreach (C1ZipEntry ze in m_Zip.Entries)
    {
        if (!ze.CheckCRC32())
        {
            MessageBox.Show("** Entry " + ze.FileName + " has errors.",
"C1Zip", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
    }

    // If we got here, everything is OK.
    MessageBox.Show("All entries passed CRC check", "C1Zip");
}
```

The main lines declare a C1ZipEntry variable and use it to loop over the entries in the zip file. The routine then calls the CheckCRC32 method on each entry to check whether the actual checksum of the bytes stored in the file matches the checksum stored in the entry header. The function returns true if the values match and false if they don't (which indicates the entry is corrupted).

Step 9: Add code to view the contents of an entry.

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

- Visual Basic

```
Private Sub btnView_Click(sender As Object, e As EventArgs) Handles
btnView.Click

    ' Get the first selected item that is not a directory.
    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

' Make sure we got something.
If ze Is Nothing Then
    MessageBox.Show("Sorry, no files to show...", "C1Zip")
    Exit Sub
End If

' Read entry content into a string.
Dim entry As Stream = ze.OpenReader()
Dim sr As StreamReader = New StreamReader(entry)
Dim entryText As String = sr.ReadToEnd()
entry.Close()

' Make sure the entry is not too big for the MessageBox.
If entryText.Length > 16000 Then
    entryText = entryText.Substring(0, 16000)
End If

' Show the entry in the message box.
MessageBox.Show(entryText, ze.FileName)
End Sub

```

- **C#**

```

private void btnView_Click(object sender, EventArgs e)
{
    // Get the first selected item that is not a directory.
    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;
        }
    }

    // Make sure we got something.
    if (ze == null )
    {
        MessageBox.Show("Sorry, no files to show...", "C1Zip");
        return;
    }

    // Read entry content into a string.
    Stream entry = ze.OpenReader();
    StreamReader sr = new StreamReader(entry);
    string entryText = sr.ReadToEnd();
    entry.Close();

    // Make sure the entry is not too big for the MessageBox.
    if (entryText.Length > 16000 )

```

```

    {
        entryText = entryText.Substring(0, 16000);
    }

    // Show the entry in the message box.
    MessageBox.Show(entryText, ze.FileName);
}

```

The function starts by choosing the first selected zip entry that is not a subdirectory. If a valid entry cannot be found, the function exits.

The function then calls the `OpenReader` method to obtain a **Stream** that can be used to read the contents of the entry. There are no temporary files or additional delays involved in this step; the data is expanded as it is read from the stream.

Next, the function reads the entry content directly into a string using the **ReadToEnd** method on the **StreamReader** object, then closes the stream.

Finally, the string is truncated and displayed in a message box.

Note that this function was designed to show the contents of short text files. If you try to view the contents of binary files, it will show only a couple of characters.

Step 10: Add code to add entries to the zip file.

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

- Visual Basic

```

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

    ' Get list of files to add.
    Dim fo As OpenFileDialog = New OpenFileDialog()
    fo.Multiselect = True
    fo.FileName = "*.*)"
    If fo.ShowDialog <> Windows.Forms.DialogResult.OK Then Exit Sub

    ' Add files in the list.
    Dim file As String
    For Each file In fo.FileNames()
        m_Zip.Entries.Add(file)
    Next file

    ' Done.
    UpdateDisplay()
End Sub

```

- C#

```

private void btnAdd_Click(object sender, EventArgs e)
{

    // Get list of files to add.
    OpenFileDialog fo = new OpenFileDialog();
    fo.Multiselect = true;
    fo.FileName = "*.)*";
    if (fo.ShowDialog() == DialogResult.OK)
    {

        // Add files in the list.
        foreach (string file in fo.FileNames)
        {

```

```

        m_Zip.Entries.Add(file);
    }

    // Done.
    UpdateDisplay();
}
}

```

Adding new entries to a zip file is easy. The main line simply calls the Add method on the C1ZipFile object, passing a string that contains the full name of the file to be added. That's all there is to it.

Step 11: Add code to extract a zip entry.

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

- Visual Basic

```

Private Sub btnExtract_Click(sender As Object, e As EventArgs) Handles
btnExtract.Click

    ' Make sure we have some selected entries.
    Dim cnt As Integer = ListView1.SelectedIndices.Count
    If cnt = 0 Then
        MessageBox.Show("Sorry, no files to extract...", "C1Zip")
        Exit Sub
    End If

    ' Confirm with user.
    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

    ' Extract all selected entries.
    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

    ' Done.
    UpdateDisplay()
End Sub

```

- C#

```

private void btnExtract_Click(object sender, EventArgs e)
{

    // Make sure we have some selected entries.
    int cnt = listView1.SelectedIndices.Count;
    if (cnt == 0)
    {
        MessageBox.Show("Sorry, no files to extract...", "C1Zip");
        return;
    }
}

```

```

    }

    // Confirm with user.
    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;

    // Extract all selected entries.
    C1ZipEntry ze;
    foreach (ListViewItem lvi in ListView1.SelectedItems)
    {
        ze = (C1ZipEntry)lvi.Tag;
        if ( ze.SizeCompressed > 0 )
        {
            m_Zip.Entries.Extract(ze.FileName);
        }
    }

    // Done.
    UpdateDisplay();
}

```

The most important line calls the Extract method on the Entries property of the C1ZipFile object. The parameter is the string that contains the file name of the entry (which is stored in the entry's FileName property).

By default, the Extract method expands the entry and saves it in the same directory where the zip file is located. There are other versions of this method that allow you to specify the destination directory and file name for the expanded file. For more details, see the C1.C1Zip Namespace section.

Step 12: Add code to remove entries from the zip file.

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

- Visual Basic

```

Private Sub btnRemove_Click(sender As Object, e As EventArgs) Handles
btnRemove.Click

    ' Make sure we have some selected entries.
    Dim cnt As Integer = ListView1.SelectedIndices.Count
    If cnt = 0 Then
        MessageBox.Show("Oops, no files to remove...", "C1Zip")
        Exit Sub
    End If

    ' Confirm with user.
    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

    ' Delete all selected entries.
    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

' Done.
UpdateDisplay()
End Sub

```

- **C#**

```

private void btnRemove_Click(object sender, EventArgs e) {

    // Make sure we have some selected entries.
    int cnt = listView1.SelectedIndices.Count;
    if (cnt == 0 )
    {
        MessageBox.Show("Oops, no files to remove...", "C1Zip");
        return;
    }

    // Confirm with user.
    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;

    // Delete all selected entries.
    foreach (ListViewItem lvi in listView1.SelectedItems)
    {
        C1ZipEntry ze = (C1ZipEntry)lvi.Tag;
        if (ze.SizeCompressed > 0)
        {
            m_Zip.Entries.Remove(ze.FileName);
        }
    }

    // Done.
    UpdateDisplay();
}

```

The most important line calls the Remove method on the Entries property of the C1ZipFile object. The parameter is the string that contains the file name of the entry (which is stored in the entry's FileName property).

This concludes the Handling Zip Files tutorial.