# Compression the Fast (and Cheap!) Way

## Doug Hennig

Need a way to compress and decompress files in your applications? VFP guru Craig Boyd has created a free library to zip and unzip files quickly and easily.

Compressing and decompressing files using the ubiquitous ZIP format is useful for many purposes in VFP applications. For example, if your application creates and emails invoices as PDF files to customers, you may wish to compress the PDF files to minimize email size and time. Your application may exchange information with other applications and that information may be sent or received inside ZIP files. A simplified backup system can use compressed files.

There are several VFP compatible utilities available to compress and decompress files. I've used DynaZip for years, but sadly the vendor for that software has gone out of business. Some of these are commercial products (read, you have to pay for them), but Craig Boyd has created one that's free, easy to use, and small (only 80 – 148K, depending on which version you use).

There are two versions of Craig's library. VFPCompression.FLL (80K) requires the Visual C++ 8.0 runtime (MSVCR80.DLL) while VFPCompression71.FLL (148K) requires version 7.1 of that runtime (MSVCR71.DLL). MSVCR71.DLL is one of the required files for VFP applications, so you may wish to use VFPCompression71.FLL so you don't have to install MSVCR80.DLL on the user's system if they don't already have it.

Here's a list of blog entries Craig has posted about his VFPCompression library. You can read them all if you wish or jump to the last one for the most current download and documentation.

- http://tinyurl.com/3yh24tz
- http://tinyurl.com/3yhl9pu
- http://tinyurl.com/34fswf9
- http://tinyurl.com/3yljmlj
- http://tinyurl.com/2wcehtx
- http://tinyurl.com/39ujep6
- http://tinyurl.com/2vj4g6s
- http://tinyurl.com/2vzdpsl
- http://tinyurl.com/3akcnx6

VFPCompression functions have several common parameters:

- cFileName: the name of a file to zip.
- cFolderName: the name of a folder to zip.
- cZipFileName: the name of a zip file to work with.
- cPassword: an optional password to use for zipping or unzipping.
- lIgnorePaths: .T. to not store or respect relative paths in the zip file or .F. to store or respect them.
- cOutputFolderName: the name of the folder to extract files to.

All of the sample code shown in this article assumes you've already used SET LIBRARY TO VFPCompression.FLL to open the library.

To test out how VFPCompression works, run TestVFPCompression.PRG. Select a test, click the Show Code button code to see the code being tested, and click Run to run the test. Most of the tests display a message box asking you to open a ZIP file to see what the test did. Do so before closing the message box because the tests clean up after themselves by deleting files they create.

### Zipping a single file

To zip a single file, use ZipFileQuick(cFileName [, cPassword]). ZipFileQuick returns .T. if it succeeded or .F. if it failed. The name of the zip file it creates is the same as cFileName but with a ZIP extension. ZipFileQuick overwrites any existing file so use ZipOpen, ZipFile, and ZipClose (discussed below) to add files to an existing zip file.

cFileName must include a fully qualified path or ZipFileQuick does weird things: it returns .F. and doesn't create the zip file if the file is in the current directory but does create it if the file's in a subdirectory (for example, "MySubDirectory\SomeFile.TXT").

Note that using FULLPATH() to fully qualify the path name causes the filename to be upper-cased, so you might want to use GetProperFileCase.PRG, a utility included with the sample files for this article that returns the proper case for a filename, if case is important.

Example:

```
lcFile1 = fullpath('TestZip\File1.txt')
llResult = ZipFileQuick(lcFile1)
    && zip the file with no password
lcFile2 = fullpath('TestZip\File2.txt')
llResult = ZipFileQuick(lcFile2, 'MyPassword')
    && zip the file using a password
```

## Zipping a single folder

To zip a single folder and its contents, use ZipFolderQuick(cFolderName [, lIgnorePaths [, cPassword]]). ZipFolderQuick returns .T. if it succeeded or .F. if it failed. It creates a zip file in the current directory with a name the same as cFolderName and a ZIP extension. ZipFolderQuick overwrites any existing file so use ZipOpen, ZipFolder, and ZipClose (discussed below) to add files to an existing zip file.

Specifying .T. for the second parameter does something unusual: it doesn't store relative paths as you'd expect but it does include the folders in the zip file as empty folders. Also, if you don't specify the folder name as a fully qualified path, ZipFolderQuick unexpectedly returns .T. but the zip file it creates consists of an empty folder.

Example:

```
lcFile = fullpath('TestZip')
llResult = ZipFolderQuick(lcFile)
    && zip the folder, preserve paths, no
    && password
llResult = ZipFolderQuick(lcFile, .T.)
    && zip the folder, don't preserve paths
llResult = ZipFolderQuick(lcFile, .F., ;
    'MyPassword')
    && zip the folder, preserve paths, use
    && password
```

## Zipping with more flexibility

When you want more flexibility, such as performing several zipping operations on a zip file or adding files to an existing zip file, use ZipOpen, as many ZipFile, ZipFileRelative, or ZipFolder calls as necessary, then ZipClose. Here's information about those functions.

ZipOpen(cZipFileName [, cFolderName [, lAppend]]): opens the zip file for processing. cFolderName is the name of the folder in which to create the zip file and lAppend should be .T. to add files to the zip file or .F. to overwrite the contents. ZipOpen returns .T. if it succeeded or .F. if it failed. I'm not quite sure why the second parameter is needed, since you can easily specify the folder to store the zip file into as part of the first parameter. If you do pass the second parameter, be sure to pass just the filename for the first parameter and a trailing backslash in the second parameter.

ZipFile(cFileName [, lIgnorePath [, cPassword]]): adds the specified file to the zip file. ZipFile returns .T. if it succeeded or .F. if it failed.

ZipFileRelative(cFileName [, cRelativePath [, cPassword]]): adds the specified file to the zip file but stores a relative path. cRelativePath is the relative path to store for the file (which can be different than the actual path for the file). ZipFileRelative returns .T. if it succeeded or .F. if it failed. The main use for this method is to create a zip file that unzips into a different folder structure than the original files were located in. This is useful when the directory structure you want to create on a user's system is different than it was on your development system and you don't want to go through the effort of creating a deployment folder structure just to create the zip file.

ZipFolder(cFolderName [ ,lIgnorePaths [, cPassword]]): zips the contents of a folder. ZipFolder returns .T. if it succeeded or .F. if it failed.

ZipClose(): closes the zip file, returning .T. if it succeeded or .F. if it failed.

Example: create a zip file and add some files to it:

```
llResult = ZipOpen(fullpath('TestZip.zip'))
llResult = ZipFile('TestZip\File1.txt')
llResult = ZipFile('TestZip\File2.txt')
llResult = ZipClose()
```

Example: add a folder to the existing zip file:

```
llResult = ZipOpen(fullpath('TestZip.zip'), ;
    '', .T.)
llResult = ZipFolder('TestZip\AnotherFolder')
llResult = ZipClose()
```

## Adding a comment to a zip file

ZipComment[cZipFileName, cComment] adds the specified comment to the zip file and returns .T. if it succeeded or .F. if it failed. If there's already a comment, it's overwritten. Pass a blank string to delete the comment.

## Zipping and unzipping a string

To zip a string, use ZipString(cString [, nLevel]), where cString is the string to zip and nLevel is a

compression level of 1 (fastest) to 9 (best compression); the default is 6. ZipString returns the compressed string.

As you may suspect, to unzip a string, use UnzipString(cString), which returns the decompressed string.

Example:

```
lcString       = 'This is a string to zip'
lcCompressed   = ZipString(lcString)
lcDecompressed = UnzipString(lcCompressed)
```

## Unzipping a zip file

To unzip a zip file, use UnzipQuick(cZipFileName [, cOutputFolderName [, lIgnorePaths [, cPassword]]]). UnzipQuick returns .T. if it succeeded or .F. if it failed. If the specified output folder doesn't exist, UnzipQuick creates it.

Note that relative paths stored in the zip file are relative to the current folder, not relative to the location of the zip file. For example, if a zip file in a subdirectory has no path information and you use UnzipQuick('MySubdirectory\MyFile.zip'), the files are extracted into the current folder, not into MySubdirectory.

Example:

```
lcFile = 'TestZip\File1.zip'
llResult = UnzipQuick(fullpath(lcFile))
    && unzip the specified file
llResult = UnzipQuick(fullpath(lcFile), ;
    curdir() + 'NewUnzipFolder')
    && unzip the specified file into a new
    && folder
```

## Unzipping with more flexibility

Like zipping, you have more flexibility if you use UnzipOpen, calls to Unzip or other functions, then UnzipClose.

In one respect, you can think of a zip file like a table: each file has an index number associated with it (like the record number of a record in a table) and one file is the "current" one. Some functions, such as UnzipFile and UnzipAFileInfo, operate on the current file. Other functions, such as UnzipByIndex and UnzipAFileInfoByIndex, allow you to specify the index number of the file to process. Like a table, there are functions allowing you to move the current file "pointer" to the first and next files, to a file by index, and to a file by name. Indexes are zero-based, so use 0 for the first file, 1 for the second, etc.

UnzipOpen(cZipFileName): opens the zip file for unzipping. UnzipOpen returns .T. if it succeeded or .F. if it failed.

UnzipClose(): closes the zip file, returning .T. if it succeeded or .F. if it failed.

Unzip([lIgnorePaths [, cPassword]]): unzips the entire zip file into a folder with the same name as the zip file unless you previously called UnzipSetFolder. For example, for a zip file named TestZip.ZIP, the zip file is extracted into a folder named TestZip, which is created if it doesn't exist. Unzip returns .T. if it succeeded or .F. if it failed.

Example:

```
llResult = UnzipOpen(fullpath('TestZip.zip'))
llResult = Unzip()
    && Unzips the files into a folder named
    && TestZip
llResult = UnzipClose()
```

UnzipTo(cOutputFolderName [, lIgnorePaths [, cPassword]]): unzips the entire zip file into the specified folder. UnzipTo creates the folder if it isn't specified, and returns .T. if it succeeded or .F. if it failed.

Example:

```
llResult = UnzipOpen(fullpath('TestZip.zip'))
llResult = UnzipTo(fullpath(''))
    && unzip the files into the current folder
llResult = UnzipClose()
```

UnzipFile(cOutputFolderName [, lIgnorePaths [, cPassword]]): unzips the current file into the specified folder and returns .T. if it succeeded or .F. if it failed. Use one of the functions below to move the "record pointer" in the zip file to make a specific file the current one.

Example:

```
llResult = UnzipOpen(fullpath('TestZip.zip'))
llResult = UnzipFile(fullpath('TestZip'))
    && unzip the first file, since that's the
    && current one
llResult = UnzipClose()
```

UnzipByIndex(nIndex [, cOutputFolderName [, lIgnorePaths [, cPassword]]]): unzips the file with the specified index and returns .T. if it succeeded or .F. if it failed.

Example:

```
llResult = UnzipOpen(fullpath('TestZip.zip'))
llResult = UnzipByIndex(4, ;
    fullpath('TestZip'))
    && unzip the fifth file (indexes are
    && 0-based, so "4" means the fifth one)
llResult = UnzipClose()
```

UnzipFileCount(): returns the number of files in the zip file.

UnzipGotoTopFile([cExtension]): moves the zip file pointer to the first file, returning .T. if it succeeded and .F. if not. If cExtension is specified, it acts like a filter on filenames, so UnzipGotoTopFile goes to the first file with that extension.

UnzipGotoNextFile([cExtension]): moves the zip file pointer to the next file, returning .T. if it succeeded and .F. if not. If cExtension is specified,

UnzipGotoNextFile goes to the next file with that extension.

UnzipGotoFileByName(cFileName [, lIgnorePath]): moves the zip file pointer to the file with the specified name, returning .T. if it succeeded and .F. if not. You can specify a relative path if desired. Pass .T. for lIgnorePath to ignore relative file paths.

UnzipGotoFileByIndex(nIndex): moves the zip file pointer to the file with the specified index, returning .T. if it succeeded and .F. if not.

UnzipAFileInfo(cArrayName): fills the specified array with information about the current file; see **Table 1** for the contents of the array. If the array already exists, it's destroyed and recreated.

**Table 1**. The contents of the array filled by UnzipAFileInfo.

| Row | Content | Data Type |
|---|---|---|
| 1 | File Name | Character |
| 2 | Comment | Character |
| 3 | Version | Numeric |
| 4 | Version Needed | Numeric |
| 5 | Flags | Numeric |
| 6 | Compression Method | Numeric |
| 7 | DOS Date | Datetime |
| 8 | CRC | Numeric |
| 9 | Compressed Size | Numeric |
| 10 | Uncompressed Size | Numeric |
| 11 | Internal Attribute | Numeric |
| 12 | External Attribute | Numeric |
| 13 | Folder | Logical |

Example:

```
llResult = UnzipOpen(fullpath('TestZip.zip'))
lnFiles  = UnzipFileCount()
messagebox('The zip file contains ' + ;
    transform(lnFiles) + ' files.')
UnzipGotoTopFile()
for lnI = 1 to lnFiles
    UnzipAFileInfo('laInfo')
    messagebox(laInfo[1] + ;
        iif(laInfo[13], ' (folder)', '') + ;
        chr(13) + 'Compressed size: ' + ;
        transform(laInfo[9]) + chr(13) + ;
        'Uncompressed size: ' + ;
        transform(laInfo[10]))
    UnzipGotoNextFile()
next lnI
llResult = UnzipClose()
```

UnzipAFileInfoByIndex(cArrayName, nIndex): like UnzipAFileInfo, but instead of the current file, references the file whose index is specified in nIndex.

## Callback functions

You can provide the user with feedback about the zipping or unzipping process using the ZipCallback function. Pass ZipCallback the name of a VFP function, procedure, or method you want called whenever a VFPCompression event occurs. Before calling the callback code, VFPCompression creates three private variables with information about the event:

- cZipObjectName: the name of the zip file (in the case of the zip file being opened or closed) or the file being zipped or unzipped.
- nZipEvent: the event number; see **Table 2** for a list of values.
- nZipBytes: the number of bytes read or written during a zip/unzip operation. This value is cumulative for a given file.

**Table 2**. Events raised by VFPCompression.

| Event | nZipEvent |
|---|---|
| Zip opened | 0 |
| Zip/unzip file start | 1 |
| Zip read or unzip write | 2 |
| Zip/unzip file end | 3 |
| Zip/unzip folder opened | 4 |
| Zip closed | 5 |

The Callbacks sample shows two callbacks. The first is a simple log file that records the progress of the compression and decompression. The second is fancier: it uses a form with a Ctl32_ProgressBar control (written by Carlos Alloatti and available at http://www.ctl32.com.ar) to display an attractive progress bar as a large file is zipped.

```
* Set up the callback.

ZipCallback('GiveFeedback()')

* Zip a folder.

llResult = ZipFolderQuick(fullpath('TestZip'))

* Unzip the zip file.

lcFile = fullpath('TestZip.zip')
llResult = UnzipQuick(lcFile, curdir() + ;
    'NewUnzipFolder')

* Display the progress log.

modify file log.txt

* Fancier callback.

loForm = newobject('ProgressForm', ;
    'Samples.vcx')
loForm.Show()
ZipCallback('loForm.Update()')
llResult = ZipOpen(fullpath('TestZip.zip'))
```

```
llResult = ZipFile(home() + 'dv_foxhelp.chm')
llResult = ZipClose()
messagebox('Done processing')

* Clean up.

erase NewUnzipFolder\*.*
erase NewUnzipFolder\AnotherFolder\*.*
rd NewUnzipFolder\AnotherFolder
rd NewUnzipFolder
erase Log.txt

function GiveFeedback
do case
    case nZipEvent = 0
        lcEvent = 'Zip file ' + ;
            cZipObjectName + ' opened'
    case nZipEvent = 1
        lcEvent = 'Started processing ' + ;
            'file cZipObjectName
    case nZipEvent = 2
        lcEvent = 'Reading/writing file ' + ;
            cZipObjectName + ': bytes ' + ;
            transform(nZipBytes)
    case nZipEvent = 3
        lcEvent = 'Done processing file ' + ;
            cZipObjectName
    case nZipEvent = 4
        lcEvent = 'Opened folder ' + ;
            cZipObjectName
    case nZipEvent = 5
        lcEvent = 'Zip file ' + ;
            cZipObjectName + ' closed'
endcase
strtofile(lcEvent + chr(13) + chr(10), ;
    'Log.txt', .T.)
```

## Summary

Craig Boyd has created a very easy-to-use and inexpensive (free!) library we can add to our VFP applications to compress and decompress files. I've been using it for a couple of years with great success in my applications and am sure you'll love it too.

*Doug Hennig is a partner with Stonefield Systems Group Inc. and Stonefield Software Inc. He is the author of the award-winning Stonefield Database Toolkit (SDT); the award-winning Stonefield Query; the MemberData Editor, Anchor Editor, and CursorAdapter and DataEnvironment builders that come with Microsoft Visual FoxPro; and the My namespace and updated Upsizing Wizard in Sedna.*

*Doug is co-author of "Making Sense of Sedna and SP2," the "What's New in Visual FoxPro" series (the latest being "What's New in Nine"), "Visual FoxPro Best Practices For The Next Ten Years," and "The Hacker's Guide to Visual FoxPro 7.0." He was the technical editor of "The Hacker's Guide to Visual FoxPro 6.0" and "The Fundamentals." All of these books are from Hentzenwerke Publishing (http://www.hentzenwerke.com). He wrote over 100 articles in 10 years for FoxTalk and has written numerous articles in FoxPro Advisor, Advisor Guide to Visual FoxPro, and CoDe. He currently writes for FoxRockX (http://www.foxrockx.com).*

*Doug spoke at every Microsoft FoxPro Developers Conference (DevCon) starting in 1997 and at user groups and developer conferences all over the world. He is one of the organizers of the annual Southwest Fox conference (http://www.swfox.net). He is one of the administrators for the VFPX VFP community extensions Web site (http://vfpx.codeplex.com). He has been a Microsoft Most Valuable Professional (MVP) since 1996. Doug was awarded the 2006 FoxPro Community Lifetime Achievement Award (http://tinyurl.com/ygnk73h).*