Website: www.chestysoft.com Email: info@chestysoft.com

csASPZipFile 3.0 - ASP Component for Creating Zip Files, Reading Zip Files and Controlling Downloads

This ASP component can create zip files from one or more files on the server. The resulting zip file can be saved on the server or sent to the browser as a binary stream. The zip file can be split to form a spanned archive. There is some support for extracting zip files which is limited to files using the inflate/deflate compression method and does not include password protected files. This component can also be used to control the download of binary files. Finally there are some file utility functions included.

A free, fully functional trial version of csASPZipFile is available. This trial version has a built in expiry date that causes the main functions to stop working after that time. This is the only difference in functionality between the trial and full versions. This means that you can fully test if this component is suitable for your application before considering whether to license the full version.

Version 3.0 is supplied as two different DLL files, one is 32 bit and the other 64 bit. Refer to the next section for more details of registration and component instantiation.

Using These Instructions

These instructions are divided into a number of sections with the relevant methods and properties described in each. There are quick links to some sections below. A full Table of Contents is available on the next page and an index listing all commands in alphabetical order is included at the back for easy reference. The PDF version also has bookmarks for direct navigation to each heading.

The component contains two classes, one for creating zip files (as well as the download and utility functions) and another for opening zip files. These classes have separate sections in the instructions and a separate alphabetical command list at the end.

Click on one of the links below to go directly to the section of interest:

Chestysoft, August 2018
www.chestysoft.com

TABLE OF CONTENTS

1. Registering the Component and Getting Started

1.1. Registration and Server Permissions

Before the component can be used the DLL file must be registered on the server. This can be done using the command line tool REGSVR32.EXE. Take care to use the correct version of this tool as there is a 64 bit version in the Windows\System32 folder and a 32 bit version in the Windows\SysWOW64 folder. The syntax is:

regsvr32 *dllname*

where *dllname* is the path and name of the DLL to register.

There are two DLL files supplied in the zip archive, one for 32 bit systems and one for 64 bit. The 32 bit file is called csASPZipFile.dll (csASPZipFileTrial.dll for the trial version). The 64 bit file is called csASPZipFile64.dll (csASPZipFile64Trial.dll for the trial version). The 64 bit file cannot be used on 32 bit systems.

Chestysoft has a free utility that performs the registration function through a Windows interface instead of using regsvr32. This tool can be downloaded from the Chestysoft web site:
www.chestysoft.com/dllregsvr/default.asp

Both classes will be registered at once. If you are upgrading from an earlier version of csASPZipFile which only contained the MakeZip class it is important to unregister the old component first, and IIS must be stopped before unregistration can take place.

We suggest creating a folder specifically for component DLLs rather than using the Windows System folder as this makes them easier to manage and avoids the naming confusion on the 64 bit systems.

The application that uses the component must have permission to read and execute the DLL. In a web application like ASP this means giving the Internet Guest User account Read and Execute permission on the file. This account must also have the appropriate permissions for file handling. Read permission is required to read/open a file from disk. Write permission is required to create a new file and Modify is required to edit or delete an existing file. These permissions can be set in Windows Explorer and applied to either a folder or individual files.

1.2. Object Creation

In any script or programme that uses the component an object instance must be created. There are two classes inside the component, MakeZip is used for creating zip files, downloading files and for the file utility functions. OpenZip is used for opening zip files and adding files to existing zip archives. The syntax in ASP to create both classes is as follows.

For the full 32 bit version:

```
Set ZipWriter = Server.CreateObject("csASPZipFile.MakeZip")

Set ZipReader = Server.CreateObject("csASPZipFile.OpenZip")
```

For the trial 32 bit version:

```
Set ZipWriter = Server.CreateObject("csASPZipFileTrial.MakeZip")

Set ZipReader = Server.CreateObject("csASPZipFileTrial.OpenZip")
```

For the full 64 bit version:

```
Set ZipWriter = Server.CreateObject("csASPZipFile64.MakeZip")
```

```
Set ZipReader = Server.CreateObject("csASPZipFile64.OpenZip")
```

For the trial 64 bit version:

```
Set ZipWriter = Server.CreateObject("csASPZipFile64Trial.MakeZip")

Set ZipReader = Server.CreateObject("csASPZipFile64Trial.OpenZip")
```

The object names are "ZipWriter" and "ZipReader", but any variable names could be used. It is not necessary to create a class instance unless that class is used in the script.

1.3. The Trial Version

The trial version of the component is supplied as a separate DLL called csASPZipFileTrial.dll (or csASPZipFile64Trial.dll). This trial version is fully functional but it has an expiry date, after which time it will stop working. The object can still be created after the expiry date but it cannot create or read zip files.

The expiry date can be found by reading the *Version* property of the MakeZip class.

Version - String, read only. This returns the version information and for the trial, the expiry date.

Example:

```
Set Zip = Server.CreateObject("csASPZipFileTrial.MakeZip")
Response.Write Zip.Version
```

Visit the Chestysoft web site for details of how to buy the full version - http://www.chestysoft.com

1.4. Using csASPZipFile with Component Services

A COM component can be added to a COM+ Application in Component Services, One reason to do this is to be able to run a 32 bit DLL on a 64 bit system. Another is to specify a Windows account to use the component to allow that component to access network files that would be unavailable if the component was called by the default internet guest user.

An online description of configuring Component Services is available here:

http://www.chestysoft.com/component-services.asp

On Windows 2008 and later it is necessary to "Allow intrinsic IIS properties" in the COM+ component properties. csASPZipFile will run without this but the StreamZip and StreamFile methods and some of the utility functions require intrinsic IIS properties.

1.5. System Requirements

csASPZipFile version 3.0 does not support earlier Windows operating systems. It requires Windows 2003 or later for a server or Windows XP or later for a desktop. It will not register or run on Windows 2000. We can still provide version 2 for users of an older operating system.

1.6. The Use of Brackets in These Instructions

In these instructions we show brackets around method parameters when the method returns a value, but not when there is no return value. We show brackets around property parameters when a property has an input value.

The requirement for brackets depends on the scripting language used. Most languages require the parameters of methods and properties to be enclosed by brackets but ASP using VBScript is an exception. In VBScript brackets are required around property parameters. They are required around method parameters only if the method returns a value and if this value is used. Brackets can be used around method parameters without an error if there is a single parameter, but an error will be generated if brackets are used to enclose multiple parameters.

For example, the *ZipAdd* method (described later) has a single parameter and a return value. The following lines are valid in ASP using VBScript:

```
Zip.ZipAdd "c:\images\1.jpg"
Zip.ZipAdd("c:\images\2.jpg")
Count = Zip.ZipAdd("c:\images\3.jpg")
```

The last line would generate an error if the brackets were missing.

An example of a method with multiple parameters is the *SaveZipDisk* method (also described later). The following line is valid in ASP using VBScript:

```
Zip.SaveZipDisk "download.zip", 1
```

If there were brackets enclosing the parameters it would generate an error.

Alternatively, brackets can be used with the Call command:

```
Call Zip.SaveZipDisk("download.zip", 1)
```

One further point on syntax worth noting is that an equals sign must be used when setting a property. For example:

```
Zip.PathRoot = "inetpub\wwwroot\"
```

A common mistake is to miss the equals sign and this generates the error "Object doesn't support this property or method". This error message is misleading.

2. The MakeZip Class

All the methods and properties described in this section use the MakeZip class, which is instantiated as described in section 1.2 above, using the class name "csASPZipFile.MakeZip" (or "csASPZipFileTrial.MakeZip" for the trial version of the component).

2.1. Creating Zip Files

The MakeZip class contains a list of files which can be added to or deleted using a number of properties and methods. When a zip file is created these files will be made into a zip file. Other properties control disk spanning and splitting. The resulting zip file can be output by saving to disk on the server, exporting as a binary data stream or sent straight to the browser.

2.1.1. File List Methods and Properties

The following properties and methods control the file list. Where an index is used to specify an item in the list it is zero based. Each entry in the list is a full physical path and file name.

ZipFileCount - Read only property. Returns an integer value which is the number of files currently listed.

ZipAdd (*FileName*) - Method to add file with physical path *FileName* to the file list. Returns an integer value which is the index of the new file.

ZipClear - Method to clear the file list. No parameters.
ZipDelete(*Index*) - Method to delete the entry specified by Index.
ZipFile(*Index*) - Property to read or write an individual file name in the list.

ZipAdd can also accept a *FileName* parameter that is a URL and it will retrieve the file from a remote server. It must begin with "http://" or "https://".

ZipAddDirectory (*Path*) - Path is a physical path to a directory and all files in the directory and in any sub directories will be added to the file list. It has an integer return value which is the number of files found. The drive is needed but the trailing backslash is optional.

Example:

```
FileCount = Zip.ZipAddDirectory("c:\files\")
```

2.1.2. Properties of Zip Files

The following properties control disk spanning, splitting and preserving directory information.

SpanDisk - Boolean. Set to true for a spanned archive. (default = false)
SplitArchive - Boolean. Set to true for a split archive. Setting *SplitArchive* to true also sets *SpanDisk* to true.(default = false)

DiskSize - Integer. Size in KB into which the zip file will be split. This has a minimum size of 100. Default is 1024 (1 MB).

DiskCount - Integer, read only. The number of separate files created when spanning or splitting is used.

KeepPathInfo - Boolean. Set to true to preserve the path information for the files in the archive. (default = false)

PathRoot - String. When *KeepPathInfo* is true the value of *PathRoot* is

removed from each path. This must not include the drive details. (default = null string)

Example:

If the files to be stored are in the directory "c:\inetpub\wwwroot\application" and the path info is to be saved but excluding the "c:\inetpub\wwwroot\" part of the path, *KeepPathInfo* is set to true and *PathRoot* is set to "inetpub\wwwroot\".

Attachment - Boolean. Used with *StreamZip* and *StreamZipData* it indicates to the browser whether the file should be displayed inline or saved as an attachment. (default = false)

The browser will usually prompt with a Save As dialogue box for a zip file, regardless of the value of Attachment, so it is usually unnecessary to change this property when working with zip files.

2.1.3. Methods for Exporting Zip Files

The following methods produce the zip file and specify the destination. It can be saved to disk on the server, streamed to the browser or exported as a binary variable.

SaveZip *Path* - The zip file is saved to disk. Path is the full physical path and file name and should include the .zip extension.

When a spanned zip file is saved a 3 digit number, 001, 002 etc will be inserted after the file name and before the extension. When a split archive is saved the extension will be .z01, .z02 etc with the last file using the .zip extension.

SaveZipDisk *Path*, *DiskNo* - This saves a single part of a spanned or split archive. *Path* is the full phyical path of the filename that it will be saved as. *DiskNo* is the section of the archive to be saved where the first part is 1 and the last is the value of *DiskCount*.

StreamZip *Name* - The zip file is streamed to the browser. Name is the name of the file that is to be displayed in any Save As dialogue box and it is a file name and .zip extension, not a full path. Do not use this with spanned archives.

There is more about streaming files later in these instructions.

StreamZipDisk *Name*, *DiskNo* - This streams a single part of a spanned archive to the browser. *Name* is the name of the file complete with .zip extension. The disk number will be inserted automatically into the file name before the period character. *DiskNo* is the section of the archive to be streamed where the first part is 1 and the last is the value of *DiskCount*. This command does not work with split archives and it is not recommended to stream split archives separately as the files can easily be saved with the wrong extension.

ZipData - This returns a variant array containing the zip file. It could be used in ASP with BinaryWrite to export to the browser if specific header information needed to be written. It could also be used to store the data in a binary field in a database.

ZipDiskData(*DiskNo*) - This returns a variant array containing a single part of a spanned or split archive archive. *DiskNo* is the section of the archive to be streamed where the first part is 1 and the last is the value of *DiskCount*.

2.1.4. Notes on Memory Use

When files are streamed to a browser using *StreamZip* or *StreamZipDisk* the entire zip file is loaded into memory. This becomes inefficient with large files (several megabytes) and in this case it is better to use *SaveZip* to save the file to the server. It can then be streamed using *StreamFile* and deleted after use. When files are saved in this way it is important to give each file a unique name because an error

will be generated if multiple users attempt to write to the same file. For example, in ASP the SessionID variable can be used as a temporary file name, or as part of the file name.

There can be a heavy load on the memory if any individual source file is very large, but functionality has been added to csASPZipFile at version 2.0 allowing a temporary file to be used while compressing the source file. Set the *TempFileName* property to use a temporary file.

TempFileName - String. When this property is set, a temporary file will be used during the compression process to reduce memory use. The value of *TempFileName* must be a a valid physical path to a file and the Internet Guest User must have permission to create this file. It should be a name that is unique to the user and in ASP we would recommend using the SessionID variable for the file name, or as part of the file name.

When *TempFileName* is used and files are created to disk using *SaveZip* the memory use for the component is quite small even when large files and archives are involved.

2.1.5. Examples

Here are some examples of generating zip files using ASP.

Example 1 - Streaming a zip file.

```
<%
  Response.Buffer = true
  Response.Expires = 0
  Set Zip = Server.CreateObject("csASPZipFile.MakeZip")
  Zip.ZipAdd("C:\images\1.jpg")
  Zip.ZipAdd("C:\images\2.jpg")
  Zip.StreamZip "example.zip"
%>
```

This example takes two local files, creates a zip file and streams it as "example.zip". Note that a script that streams a file like this must not contain any other output. Any HTML or a Response.Write statement would send additional data in the data stream and the resulting zip file could be corrupt.

Example 2 - Saving a file.

```
<%
  Response.Buffer = true
  Response.Expires = 0
  Set Zip = Server.CreateObject("csASPZipFile.MakeZip")
  Zip.ZipAdd("C:\images\1.jpg")
  Zip.ZipAdd("C:\images\2.jpg")
  Zip.KeepPathInfo = true
  Zip.SaveZip "C:\zips\example.zip"
%>
```

Similar to Example 1 but the file is saved. This time the path information is stored within the file.

Example 3 - Saving a spanned archive.

```
<%
  Response.Buffer = true
  Response.Expires = 0
  Set Zip = Server.CreateObject("csASPZipFile.MakeZip")
  Zip.DirName = "C:\images\"
  For Each Filename in Zip.FileList
    Zip.ZipAdd(Zip.DirName & FileName)
  Next
```

```
  Zip.SpanDisk = true
  Zip.DiskSize = 500
  Zip.SaveZip "C:\zips\example.zip"
%>
```

This zips all the files in the directory "C:\images\" and creates a spanned archive where the individual file size is 500 KB. The files will be named "example001.zip", "example002.zip" and so on.

It is possible to stream individual parts of a spanned archive using either *StreamZipDisk* or *ZipDiskData* but there are limitations. A separate script will be needed for each section. The number of parts or disks used cannot be specified in advance. The size is specified and the number of parts or disks can be read after compression. The file is compressed in full each time an individual part is produced leading to an increased server load.

2.2. Renaming Files Inside the Archive

Sometimes it is necessary to give the file a different name or path inside the zip file compared with the name on the server. The following methods and properties allow a second file list to be created containing the name that is to be used inside the zip file. If this second file list is empty the original names will be used. Where an index is used to specify an item in the list it is zero based. Each entry in the list is either a path or just a file name. If it is a path there should be no drive letter or leading backslash.

AltNameCount - Read only property. Returns an integer value which is the number of alternative names currently listed.

AltNameAdd(*AltName*) - Method to add file with name or path *AltName* to the list. Returns an integer value which is the index of the new name.

AltNameClear - Method to clear the alternative name list. No parameters.
AltNameDelete(*Index*) - Method to delete the entry specified by *Index*.
AltName(*Index*) - Property to read or write an individual name in the list.

If alternative names are to be used it is important to add them in the same order as the original names. The first name in the alternative list will be used to name the first name in the source file list, etc.

Example - Giving files alternative names.

```
<%
  Set Zip = Server.CreateObject("csASPZipFile.MakeZip")
  Zip.ZipAdd("C:\location\of\file\originalname.ext")
  Zip.AltNameAdd("download\name1.ext")
  Zip.ZipAdd("C:\location\of\file\secondname.ext")
  Zip.AltNameAdd("download\name2.ext")
  Zip.KeepPathInfo = true
  Zip.SaveZip "C:\zips\example.zip"
%>
```

This shows two files being added. For every *ZipAdd* command there is a corresponding *AltNameAdd* to specify the alternative name. The alternative name is a path and there is no leading backslash.

2.3. Controlling Downloads

The MakeZip class in csASPZipFile controls file downloads by loading the file into memory from wherever it resides on the server, then streaming it to the browser. This allows code to be run before or after file transfer enabling form variables to be read, databases to be updated and any other record keeping that is required. The file to be downloaded does not need to be on a part of the server that is web shared, but the Internet Guest User must have read permission on the file.

The asp script that controls the download does not return an html page. It should have the response buffer set to true, and the appropriate MIME type set using Response.ContentType. The example in the next section shows this.

The browser may display the file or it may ask the user if they want to open or save the file. The exact behaviour depends on the type of browser and its configuration.

2.3.1. The StreamFile method

The simplest way of sending a file to a browser involves calling the *StreamFile* method. This takes the full physical path of the file as a parameter. This is the syntax of the method:

StreamFile *FileName* - *FileName* is the full physical path of the file to be streamed.

Here is a sample script:

```
<%
  Response.Buffer = true
  Response.Expires = 0
  Response.ContentType = "application/x-zip-compressed"
  Set Download = Server.CreateObject("csASPZipFile.MakeZip")
  Download.StreamFile "C:\download\sample.zip"
%>
```

This case shows a zip file being downloaded, so the ContentType is set accordingly. Setting the response buffer allows the download to be sent in smaller blocks instead of one big block and setting Response.Expires to zero stops the page being cached.

The *StreamFile* method builds the HTTP header and includes the file name and file size. Most browsers will prompt for this file name when saving the file. In the example shown that is "sample.zip". However, some browsers will use the name of the asp script instead.

An additional property, *PromptName*, can be set to send a different file name in the header. The previous example could be modified to prompt for the name "othername.zip":

```
Download.PromptName = "othername.zip"
Download.StreamFile "C:\download\sample.zip"
```

PromptName - String. The file name sent in the HTTP header when downloading using *StreamFile*, if different from the original file name. (Default = empty string)

2.3.2. The Attachment Property

As well as including the file name in the "Content-Disposition" header, the *StreamFile* method will specify whether the file should be displayed inline or as an attachment. Use the Boolean *Attachment* property for this and set it to True to show the file as an attachment. The default value is False, causing the file to be displayed by the browser if possible. Not all browsers interpret this directive but Internet Explorer, Chrome and Firefox will.

Behaviour does vary, not just between different browsers, but between browsers of the same type that have been configured differently by the user. It is important to check the behaviour of a download script in different browsers.

2.3.3. The FileData method

Another option is available to read file data which is more versatile. The *FileData* method takes the file name as an argument and returns an OLE variant containing the file data.

FileData(*FileName*)	-	Variant array return value. *FileName* is the path to the file.

Using F*ileData* the previous example would become:

```
<%
  Response.Buffer = true
  Response.Expires = 0
  Response.ContentType = "application/x-zip-compressed"
  Set Download = Server.CreateObject("csASPZipFile.MakeZip")
  Response.AddHeader "Content-Disposition",
    "inline; filename=sample.zip"
  Response.AddHeader "Content-Length", Download.FileSize(
    "C:\download\sample.zip")
  Response.BinaryWrite Download.FileData("C:\download\sample.zip")
%>
```

These two examples have the same effect, but if the file data was to be stored in a database, for example, the *FileData* method would be used. The *StreamFile* method always adds the file name to the header, and always sends the data to the browser, and there may be occasions when this is undesireable.

The *FileData* method is more demanding on server memory and processing than *StreamFile* because it generates a variable that is as big as the file itself and so it should not be used with large files. IIS has a property called ASPBufferingLimit, or the Response Buffering Limit, which restricts the size of data that can be sent through BinaryWrite and it defaults to 4 MB, so *FileData* cannot be used with file sizes greater than this unless this property is changed. *StreamFile* sends the file in smaller chunks and is not restricted by ASPBufferingLimit.

2.4. Retrieving a File From a Remote Web Server

The csASPZipFile component can get a file from a remote server using the URL and then save it or stream it to the browser. The following commands are used:

StreamFromURL *URL*, *FileName*	-	Streams the file at *URL* directly to the browser. *FileName* is the name sent to the browser.
SaveFromURL *URL*, *FileName*	-	Saves the file at *URL* to disk where *FileName* is the physical path of the destination.
URLData(*URL*)	-	Returns the file data as a variant array.

Example of saving a file from a remote URL:

```
Download.SaveFromURL "http://domain/directory/file.ext",
"C:\download\file.ext"
```

Example of streaming the file to the browser:

```
Response.ContentType = "application/x-zip-compressed"
Download.StreamFromURL "http://domain/directory/file.zip", "file.zip"
```

This example shows a zip file so the content type is set accordingly.

The above commands can specify a user name and password with the request. Passwords are sent as plain text if the server uses Basic Authentication. If the server uses Integrated Windows Authentication csASPZipFile must be added to a COM+ application in Component Services, as described in Section 1.4, and a named account or the interactive user must be specified. Authentication will fail if the Service account is used.

Set the following properties before calling *StreamFromURL*, *SaveFromURL* or *URLData*.

URLUsername - String. Username to be passed with *StreamFromURL*, *SaveFromURL* or *URLData*.

URLPassword - String. Password to be passed with *StreamFromURL*, *SaveFromURL* or *URLData*.

The *HTTPUserAgent* property can be set to specify a user agent in the request header.

HTTPUserAgent - String. Value for the User Agent request header when *StreamFromURL*, *SaveFromURL* or *URLData* is called. This is null by default.

HTTPTimeout - Integer. Number of seconds before *StreamFromURL*, *SaveFromURL* or *URLData* will time out due to inactivity. A zero value is an indefinite time, and this is the default.

2.5. Verifying Completed Downloads

When files are downloaded using the csASPZipFile component, it is possible to record whether the download was successful. This is done by using the Response.IsClientConnected command immediately after streaming the file to the browser. The earlier example is modified as follows:

```
<%
  Response.Buffer = true
  Response.Clear
  Response.ContentType = "application/x-zip-compressed"

  Set Download = Server.CreateObject("csASPZipFile.MakeZip")
  Download.FileName = "C:\download\sample.zip"
  Download.StreamFile
  Response.Flush
  If Response.IsClientConnected = true Then
    'The download was completed
    'Do something
  Else
    'The download was not completed
    'Do something else
  End If
%>
```

It is important to include Response.Flush immediately after the file was streamed. This effectively pauses the script while the data is in transit to the browser. If the connection is not maintained during this time, Response.IsClientConnected will return false. It is also important to note that html output cannot be included at this stage and it is not possible to redirect the script. Only "hidden" server side processing can be done, but this can include database or text file manipulation to update records.

This method is not completely accurate. If the download is cancelled early enough neither of the options in the If statement will be reached. If the files is small (e.g. less than 200 KB) it will always appear to be a complete download. There is also no way to determine if the end user successfully saved the file after downloading.

2.6. Permissions and Accessing Remote Files

There are some important points to consider when working with files from a server side script. The script accesses files on the same server using the Internet Guest User account (IUSR_machine_name). This account frequently has limited default permissions so it may be necessary to adjust the permissions on files or directories with which the script needs to work.

Remote network files can be accessed by using the shared path or the UNC path but the csASPZipFile component must first be registered as a COM+ application in Component Services, as described in Section 1.4.

It is possible that a firewall on the server computer can interfere with the access of remote files. This could affect network files or files read using the URL functions.

2.7. MIME Types

When files are streamed to a browser, it is important to specify the MIME type using Response.ContentType. The example above shows the MIME type for a zip file. There is a function available which obtains the MIME type, given the file extension. It extracts the information from the server system registry, which means the Internet Guest Account must have permission to access the registry. It should have this level of permission already because it needs it to find the registration details of the component.

GetMimeType(*Extension*) - Read only property, string return value. Extension is the file extension, with or without the period character. If there is no MIME type recorded for that file type the return value will be "application/unknown".

The code:

```
Response.Write Download.GetMimeType("zip")
```

would result in the output: "application/x-zip-compressed", assuming there is an entry for ".zip" in the registry and the script has permission to read the registry. Otherwise it would return "application/unknown".

It is usually better to find the correct MIME type first and hard code it into the script. It is unlikely that the *GetMimeType* method will work without making adjustments to the security settings or using Component Services to higher level access rights to the component.

2.8. The Access Code Function

The MakeZip class of the csASPZipFile component has a built in function for generating access codes. This can be used in conjunction with file downloading, and is particularly useful if there is no database available on the server for maintaining a list of eligible users.

AccessCode (*String1*, *String2*, *IDNo*) - takes 3 strings as input and returns a 15 digit hexadecimal number. *String1* and *String2* can be anything. They could be name and email address, username and password, one could even be empty if required. The third value, *IDNo* is a number between 0 and 4294967295 (2^32). IDNo can be entered as an 8 digit hexadecimal number if it is enclosed in quotation marks and prefixed with a hash (#) or dollar ($) character, i.e. "#123456AB".

The returned value of *AccessCode* will always be the same for a given set of inputs, so a code can be given out on one web page and verified on another. The *IDNo* is hidden from the end user, so even if they have a copy of the component, they cannot predict the access code. *IDNo* should never be passed in a URL string or a form variable or displayed in any other way.

Here is an example:

A web page asks the user for their name and email address and passes these as form variables to a script which calculates an access code. The following lines will display the code:

```
<%
  Set Download = Server.CreateObject("csASPZipFile.MakeZip")
  Response.Write Download.AccessCode(Request.Form("Name"), _
```

```
    Request.Form("Email"), "#ABCDEF01")
%>
```

This creates an instance of the MakeZip class called "Download". It then takes the two form variables and passes them to the *AccessCode* method along with a value for *IDNo*. If this was completed by a user named "Fred", with an email address of "fred@somewhere.com", the access code returned would be "66E78194DA6131F".

Another page asks the user for their name, email address and access code and passes these as form variables to a script which verifies the code. The following lines perform the verification:

```
<%
  Set Download = Server.CreateObject("csASPZipFile.MakeZip")
  If Download.AccessCode(Request.Form("Name"), _
    Request.Form("Email"), "#ABCDEF01") =  _
    Request.Form("AccessCode") Then
    'Code is correct
    'Do something
  Else
    'Code is not correct
    'Do something else
  End If
%>
```

This takes the name and email address that were supplied as form variables and passes them to the *AccessCode* method, along with the same value of *IDNo* used earlier. The return value is then compared with the code supplied by the user. Note that all the string values are case sensitive.
As with any password system, there will be ways of breaking it or bypassing it. This system has no guarantees, but there are many applications where this level of protection is appropriate.

2.9. File Utilities

There are a number of file utility functions included for convenience. They are not intended to be a comprehensive set, because standard ASP has the File System Object to cover most file utilities. These are the functions that are most likely to be useful while controlling file downloads or creating zip files.

CurrentDir - This property returns the actual path of the directory containing the script. It is complete with the trailing backslash character. It only works with ASP.

ParentDir(*Directory*) - *Directory* is a string value and must be a full directory path. The return value is the parent directory.

Example:

```
Response.Write Download.ParentDir(Download.CurrentDir)
```

This would display the parent directory to the one containing the current script.

DirName - String. This is the directory that will be listed in the *FileList* collection, described next. It is a full physical path and can include a filter in the file name.

Example:

```
Download.DirName = "C:\zipfiles\"
```

This will assign all the files in the "zipfiles" directory to the *FileList* collection.

```
Download.DirName = "C:\zipfiles\*.zip"
```

This will assign all the files with the extension .zip in the "zipfiles" directory to the *FileList* collection.

FileList - Collection of strings. When a directory is assigned to the *DirName* property this collection will be populated by a list of the files in that directory. As a Collection it can be accessed by index or in a For .. Each loop and it has a count property.

Example:

```
Download.DirName = "C:\zipfiles\*.zip"
For Each ZipFile in Download.FileList
  Response.Write ZipFile & "<br>"
Next
```

This would display all the zip files in the specified directory.

DirSortType - Integer enumeration. This determines the order of the files in the *FileList* collection. It must be set before the *DirName* property. Available values are 0 - alphabetical ascending, 1 - alphabetical descending, 2 - date order ascending, 3 - date order descending. The default is 0. For date sorting it is the last modified date that is used.

FileExists(*FileName*) - Returns a Boolean value. *FileName* is the physical path and file name of the file in question.

GetFileName(*Path*) - This returns the file name, complete with extension but without the directory structure where *Path* is a full physical path to a file or part of a path.

GetExtension(*Path*) - This returns the extension, complete with the period character where *Path* is a full physical path to a file or part of a path.

ScriptName - A read only property returning the current script name complete with extension. This only works with ASP.

FileSize(*FileName*) - *FileName* is the full path and filename of a file. The return value is the file size in bytes.

Delete(*FileName*) - This deletes the file *FileName*. Note that it is permanently deleted, NOT placed in the Recycle Bin.

Copy *OldName*, *NewName* - This copies the file *OldName* to the location and name given by *NewName*. Full paths are required.

Rename *OldName*, *NewName* - This renames the file *OldName* to *NewName*. Full paths are required, and so renaming to a different directory is the equivalent of moving the file.

AppendToFile *FileName*, *NewLine* - This appends the string *NewLine* to the text file *FileName*. If the text file does not exist, it will be created if possible. The full physical path is required.

Example:

```
Download.AppendToFile Download.CurrentDir & "test.txt", "Hello"
```

This will append the line "Hello" at the end of a text file called test.txt which is in the same directory as the current script. If the file does not exist it will create it.

AppendToFile is the only command in this component for manipulating text files. It is useful for maintaining a simple log file containing download information. There is a full set of commands for dealing with text files in the built in File System Object.

All the file handling routines require that the Internet Guest Account has the appropriate permissions on the server, otherwise errors will result.

3. The OpenZip Class

All the methods and properties described in this section use the OpenZip class, which is instantiated as described in section 1.2 above, using the class name "csASPZipFile.OpenZip" (or "csASPZipFileTrial.OpenZip" for the trial version of the component).

This class has functionality to find information about files that are inside zip archives and to extract those files. Zip files can be taken from disk, from a variant array variable or from a remote URL. Files can be extracted to disk or to a variant array variable. Files can be extracted either to a named location or to a sub directory specified inside the zip archive. There is functionality to add files to or delete files from an existing archive. Spanned and split files can also be opened.

Not all variations of the zip format are supported. Only files using the deflate compression method can be read by csASPZipFile, and password protected (encrypted) files cannot be read. Even so, a large proportion of zip files can be opened. The deflate compression method is by far the most widely used compression method and a lot of popular archive software does not offer any other compression options.

3.1. Reading the Zip File

The first stage in extracting files from a zip archive is to read the zip file. This will set some properties giving information about the files and the number of files in the archive.

3.1.1. Methods for Reading the Zip File

The following methods are used to read an archive stored as a single file.

ReadZipFromFile *FileName* - *FileName* is a string and it is the full physical path to the zip file. After calling this method any further commands to extract from or edit the archive will apply to this file.

ReadZipFromURL *URL*- *URL* is a string and is the full URL to the zip file, starting with "http://" or "https://". After calling this method the zip file will be held in memory until the script ends, so it can be demanding on memory if used with large files.

ReadZipFromVariant *FileData* - *FileData* is a variant array containing the zip file. Calling this method will also hold the zip file in memory until the script ends.

Spanned or split archives can also be read, but only when they stored on disk. There is no functionality for reading them from a variant or URL. The file paths must be added to a list using the *AddSpannedFile* method, and they must be added in order. The archive can then be read using *ReadSpannedZip*.

Spanned archives and split archives are identical except for the file extensions. Spanned archives were traditionally used with floppy disks and all have the extension ".zip". Split archives have the extensions ".z01", ".z02", ".z03" etc, with the last file having the extension ".zip". The last file in a spanned or split set is the file that contains a summary of the files stored inside the archive. Some software may ask for this file first when reading the archive but it is the last file in the set.

AddSpannedFile *FileName* - *FileName* is the physical path to the spanned or split file which is to be added to the list that is held in memory. All the files in the set must be added in order, with the first file added first. This method returns an integer value which is the zero based index of the file added.

ClearSpannedFiles - This clears the list of spanned files added using *AddSpannedFile*.

ReadSpannedZip - This reads the spanned or split archive specified using the *AddSpannedFile* method. Any further commands for extracting files from the archive will use these files.

Example of reading a spanned archive:

```
Zip.AddSpannedFile "C:\files\disk1.zip"
Zip.AddSpannedFile "c:\files\disk2.zip"
Zip.AddSpannedFile "c:\files\disk3.zip"
Zip.ReadSpannedZip
```

This would read a set of three files if they were called "disk1.zip", "disk2.zip" and "disk3.zip".

3.1.2. Properties Set by Reading the Zip File

After reading a file as described above some read only properties will be set to provide information about the archive and its contents. These include a Boolean property to indicate that an archive has been successfully read, a property counting the number of files, the name and path of each file, the date and time each file was modified and the compressed and uncompressed sizes.

Count - Integer, read only. This is the number of files in the archive. It will be zero when no file has been read.

FileAvailable - Boolean, read only. This will be set to true when a single file has been read.

SpannedFileAvailable - Boolean, read only. This will be set to true when a spanned file has been read.

The properties describing each file in the archive have an index, which is a zero based integer, i.e. the first file has an index of zero. The order of the files in the archive is the order in which they are shown in the Central Directory Record at the end of the file.

FileName(*Index*) - String, read only. The file name, which may include a path. If a path is included it will begin with a sub directory name, not a drive letter or a backslash character.

CompressedSize(*Index*) - Integer, read only. The compressed size of the file inside the archive.

UncompressedSize(*Index*) - Integer, read only. The size of the file when it is extracted.

ModDateTime(*Index*) - DateTime, read only. This is the date and time when the file was last modified, as a DateTime value.

Example of displaying the file properties from a zip archive:

```
Set Zip = Server.CreateObject("csASPZipFile.OpenZip")
Zip.ReadZipFromFile Server.MapPath("example.zip")
For I = 0 to Zip.Count - 1
  Response.Write Zip.FileName(I) & ", " & Zip.CompressedSize(I) & _
  ", " & Zip.UncompressedSize(I) & ", " & Zip.ModDateTime(I) & "<br>"
Next
```

This will open the file "example.zip" from the same directory as the script and loop through all the files, displaying the file name, the compressed and uncompressed file sizes and the last modified date and time.

3.2. Extracting Files From an Archive

Once a zip archive has been read, as described above, the contents can be extracted and either saved to disk as a file or exported as a variant array, for streaming to a browser or saving into a binary database field. The file can be saved using the existing name, or a new name.

The directory where files are to be saved must be specified in the *PathRoot* property, which must be set before calling one of the extraction methods. If the zip archive contains path information for each file this can also be used and sub directories will be created if required. The appropriate permissions must be set if directories are to be created.

PathRoot - String property. This is the full physical path to the directory where the files will be stored, or it will be the root directory below which sub directories will be created if the zip archive contains files with path information. This path should contain the drive as well as a trailing backslash, although the backslash will be added automatically if it is missing. This property must be set before extracting a file to disk.

The following methods save one or all of the files in the archive to disk.

ExtractFileToDisk *FileName*, *Index* - This will extract a single file. *FileName* is a string and it specifies the file name that will be used when the file is saved. If it is an empty string the file name from inside the archive will be used. Index is the integer index value of the file inside the archive where the first file has a zero index value. The physical path to the saved file is given by *PathRoot* + *FileName*. This method has a return value which is the file name that was used to save the file.

ExtractAllToDisk - This will extract all the files in the archive and save them to disk. The path to each saved file will be specified by the *PathRoot* property, as well as the file name inside the archive.

ExtractSpannedFileToDisk *FileName*, *Index* - This will extract a single file from a spanned or split archive. *FileName* is a string and it specifies the file name that will be used when the file is saved. If it is an empty string the file name from inside the archive will be used. *Index* is the integer index value of the file inside the archive where the first file has a zero index value. The physical path to the saved file is given by *PathRoot* + *FileName*. This method has a return value which is the file name that was used to save the file.

ExtractAllSpannedToDisk - This will extract all the files in a spanned archive and save them to disk. The path to each saved file will be specified by the *PathRoot* property, as well as the file name inside the archive.

Example of extracting a single file from the archive and saving it to disk:

```
Set Zip = Server.CreateObject("csASPZipFile.OpenZip")
Zip.ReadZipFromFile Server.MapPath("example.zip")
Zip.PathRoot = Server.MapPath(".") & "\"
Zip.ExtractFileToDisk "", 0
```

This will read the file "example.zip" and extract the first file using the original file name. The zip file is in the same directory as the script and this directory is used as the destination for the file. In practice it would be better to work with files in a different directory from the script to prevent the possibility of overwriting the script with the extracted file. If the file inside the archive contains path information, appropriate sub directories will be created and the Internet Guest User must have permission to do this.

The file could be extracted to a different location by assigning the PathRoot property a different value.

The following methods extract a single file as a variant array and can be used for storing the file in a binary database field or for streaming the file to the browser.

ExtractFileAsVariant *Index* - Variant return value. This method extracts a single file from the archive specified by *Index*, which is the integer index of the file inside the archive.

ExtractSpannedFileAsVariant *Index* - Variant return value. This method extracts a single file from a spanned or split archive. The file is specified by *Index*, which is the integer index of the file inside the archive.

Example of streaming a file from the archive to the browser:

```
Set Zip = Server.CreateObject("csASPZipFile.OpenZip")
Set Extra = Server.CreateObject("csASPZipFile.MakeZip")
Zip.ReadZipFromFile Server.MapPath("example.zip")
Response.ContentType = _
Extra.GetMIMEType(Extra.GetExtension(Zip.FileName(0)))
Response.BinaryWrite Zip.ExtractFileAsVariant(0)
```

This reads the zip file "example.zip" and sends the first file to the browser using BinaryWrite. It uses the *GetMIMEType* method from the MakeZip class to set the content type.

3.3. Editing an Existing Archive

It is possible to add and delete files from an existing zip archive. The existing archive must be saved on disk but new files can be added from disk, from a variant array variable, from a remote URL or as a complete directory. Only single file archives can be edited in this way, not spanned or split archives. The file to be edited must first be read as described in section 3.1.1 above and it must be read from disk.

AddToFileFromDisk *FileName*, *KeepDirInfo*, *PathRoot*, *AltFileName* - This adds a file from disk to the current archive. *FileName* is the physical path of the file to be added. *KeepDirInfo* is a Boolean parameter indicating whether the path information is to be saved inside the archive. *PathRoot* is the part of the physical path that will be removed from the beginning, if *KeepDirInfo* is true. *AltFileName* is the file name that will be written into the archive. After calling this method the new file will be added to the end of the archive and the *Count* property will be updated.

AddToFileFromURL *URL*, *KeepDirInfo*, *AltFileName* - This adds a file to the current archive, from a remote URL. *URL* is the full URL of the file to be added, starting with "http://". *KeepDirInfo* is a Boolean parameter indicating whether path information is to be saved inside the archive. *AltFileName* is the file name that will be written into the archive, including path information if required. After calling this method the new file will be added to the end of the archive and the *Count* property will be updated.

AddToFileFromVariant *FileData*, *FileName*, *KeepDirInfo* - This adds a file to the current archive where the file is stored as a variant array. *FileData* is this variant array variable. *FileName* is the file name that will be written into the archive, including path information if required. *KeepDirInfo* is a Boolean parameter indicating whether the path information is to be saved inside the archive. After calling this method the new file will be added to the end of the archive and the *Count* property will be updated.

AddToFileFromDir *DirName*, *KeepDirInfo*, *PathRoot* - This adds the files contained in the directory *DirName* to the current archive and includes files in sub directories. *KeepDirInfo* is a Boolean parameter indicating whether the path information is to be saved inside the archive. *PathRoot* is the part of the physical path that will be removed from the beginning, if *KeepDirInfo* is true.

Example of adding a file from disk to an existing archive:

```
Set Zip = Server.CreateObject("csASPZipFile.OpenZip")
Zip.ReadZipFromFile Server.MapPath("example.zip")
Zip.AddToFileFromDisk Server.MapPath("oldname.jpg"), false, "", _
"new.jpg"
```

The object is created and the zip archive is read using *ReadZipFromFile*. The file to be added is called "oldname.jpg". The path information is not required so the *KeepDirInfo* parameter is false and the *PathRoot* parameter is an empty string. To demonstrate how the file name inside the archive can be specified the *AltFileName* has been set to "new.jpg". The zip archive and the file to be added are both in the same directory as the script and full physical paths are needed to describe these files.

Example of adding a file from a URL to an existing archive:

```
Set Zip = Server.CreateObject("csASPZipFile.OpenZip")
Zip.ReadZipFromFile Server.MapPath("example.zip")
Zip.AddToFileFromURL "http://www.chestysoft.com/images/ch1o.gif", _
true, "images/logo.gif"
```

This opens the zip archive called "example.zip" and adds a file from a remote URL. The *KeepDirInfo* parameter is true so a path and file name will be added to the archive. This path is specified by the *AltFileName* parameter.

A file can be deleted from a zip archive by using the *DeleteFromFile* method. As with adding files, the archive must be read first using *ReadZipFromFile*.

DeleteFromFile(*Index*) - This deletes the file specified by Index from the zip archive that has been read. Index is an integer where 0 is the first file in the archive. After deletion, the *Count* property and the internal file properties are updated. *DeleteFromFile* will generate an error if it is used to delete the last file from the archive.

Example:

```
Set Zip = Server.CreateObject("csASPZipFile.OpenZip")
Zip.ReadZipFromFile Server.MapPath("example.zip")
Zip.DeleteFromFile(5)
```

This reads the zip archive called "example.zip" and deletes the file with index 5 (the 6th file) from it. If there are less than 6 files in the archive this will generate an error.

3.4. Properties Used With Remote URLs

When files are retrieved from remote URLs it is possible to send a user name and password with the request. It sends the passwords as plain text for Basic Authentication and it is not suitable for Integrated Windows Authentication. Set the following properties before calling one of the methods that uses a remote URL, e.g. *ReadZipFromURL* or *AddToFileFromURL*.

URLUsername - String. Username to be passed with *ReadZipFromURL* or *AddToFileFromURL*.

URLPassword - String. Password to be passed with *ReadZipFromURL* or *AddToFileFromURL*.

The *HTTPUserAgent* property can be set to specify a user agent in the request header.

HTTPUserAgent - String. Value for the User Agent request header when *ReadZipFromURL* or *AddToFileFromURL* is called. This is null by default.

HTTPTimeout - Integer. Number of seconds before *ReadZipFromURL*, or *AddToFileFromURL* will time out due to inactivity. A zero value is an indefinite time, and this is the default.

3.5. Notes On Memory Use

When files are extracted from a zip archive they will be stored in memory during the extraction process, which can be demanding on server memory if the files are large. This can be prevented by using a temporary file. Set the *TempFileName* property to use a temporary file during extraction.

TempFileName - String. When this property is set, a temporary file will be used during the decompression process to reduce memory use. The value of *TempFileName* must be a a valid physical path to a file and the Internet Guest User must have permission to create this file. It should be a name that is unique to the user and in ASP we would recommend using the SessionID variable for the file name, or as part of the file name.

When *TempFileName* is used and files are read from and saved to disk the memory use for the component is quite small even when large files and archives are involved.

When files are read form variant array variables or from remote URLs, the entire file will be loaded into memory and this can be demanding on server memory if these files are large.

4. Using csASPZipFile with Cold Fusion

csASPZipFile is a COM object and can be used with Cold Fusion if it is running on a Windows platform, although it should be noted that only the 32 bit version of Cold Fusion has COM support. The object is created with the <cfobject> tag:

```
<cfobject action="create" name="ZipWriter" class="csASPZipFile.MakeZip">
<cfobject action="create" name="ZipReader" class="csASPZipFile.OpenZip">
```

for the full version or class="csASPZipFileTrial.MakeZip" and class="csASPZipFileTrial.OpenZip" for the trial version.

Each command must be placed inside a <cfset> tag and all method parameters must be enclosed in brackets. The earlier example of saving a zip file would become:

```
<cfobject action="create" name="Zip" class="csASPZipFile.MakeZip">
<cfset Zip.ZipAdd("C:\images\1.jpg")>
<cfset Zip.ZipAdd("C:\images\2.jpg")>
<cfset Zip.KeepPathInfo = true>
<cfset Zip.SaveZip("C:\zips\example.zip")>
```

Alternatively, the commands can be put inside a <cfscript> block:

```
<cfscript>
Zip.ZipAdd("C:\images\1.jpg");
Zip.ZipAdd("C:\images\2.jpg");
Zip.KeepPathInfo = true;
Zip.SaveZip("C:\zips\example.zip");
</cfscript>
```

Cold Fusion version 5 does not have an equivalent command to BinaryWrite in ASP, so the streaming commands do not work. If a zip file is to be streamed after creation it must be saved to disk first, possibly using a temporary file name. The <cfcontent> tag is used to stream the file to the browser and this can also delete the file. One way of generating a unique temporary file name is to use the CreateUUID function. The SessionID could also be used if your server is configured to use them. Here is an example of streaming a file:

```
<cfobject action="create" name="Zip" class="csASPZipFile.MakeZip">
<cfset Zip.ZipAdd("C:\images\1.jpg")>
<cfset Zip.ZipAdd("C:\images\2.jpg")>
<cfset Zip.KeepPathInfo = true>
<cfset tempname=ExpandPath(".") & CreateUUID() & ".zip">
<cfcontent type="application/x-zip-compressed" deletefile="yes"
file=#tempname#>
```

The Internet Guest User must have Full Control permission on the directory containing the temporary file for the deletion to work.

Cold Fusion does have an undocumented method of streaming a file and if this is used it is not necessary to save the zip as a temporary file. The following commands will stream a zip file assuming that a csASPZipFile object called "Zip" has been created and the files have been added to the archive.

```
<cfscript>
Context = GetPageContext();
Context.SetFlushOutput(false);
Response = Context.GetResponse().GetResponse();
Out = Response.GetOutputStream();
Response.SetContentType("application/x-zip-compressed");
Out.Write(Zip.ZipData);
Out.Flush();
```

```
Response.Reset();
Out.Close();
</cfscript>
```

This example does not include the Content Length and so the browser will be unable to display a progress bar as the file downloads. A modification can be made which will resolve this, although it will use more memory. The zip file is temporarily stored in memory while the size is found.

```
Temp = Zip.ZipData;
Response.SetContentLength(ArrayLen(Temp));
Out.Write(Temp);
```

There are some examples of using this and other components on our demonstration web site:

http://www.chestysoft.co.uk/cfdemos.cfm

5. Using csASPZipFile with ASP.NET

csASPZipFile can be used with ASP.NET. The component must be registered on the server, as described earlier, and it can be called using Server.CreateObject. For example:

```
Dim Zip = Server.CreateObject("csASPZipFile.MakeZip")
```

Note that the object is created using Dim instead of Set. For the trial version the class name is "csASPZipFileTrial.MakeZip").

The *StreamZip* and *StreamFile* commands cannot be used in ASP.NET. It is possible to stream a file to the browser using *ZipData* and BinaryWrite, but not directly, because of incompatibilities between the data types of ActiveX and .NET. There is a workaround described in the VB.NET example below.

```
<%@ Page language="vb" debug="true" %>
<%
  Response.Expires = 0
  Response.Buffer = true
  Response.Clear
  Dim Zip = Server.CreateObject("csASPZipFile.MakeZip")
  Zip.ZipAdd("c:\files\file.ext")
  Dim ZipArray As Array = Zip.ZipData
  Dim OutArray(ZipArray.Length - 1) As Byte
  Array.Copy(ZipArray, OutArray, ZipArray.Length)
  Response.ContentType = "application/x-zip-compressed"
  Response.AddHeader("Content-Disposition", "inline;
    filename=sample.zip")
  Response.BinaryWrite(OutArray)
%>
```

This code will zip the file "c:\files\file.ext" and stream it to the browser. The output from the *ZipData* command is cast to an array and then to an array of bytes to make it compatible with the BinaryWrite method. This technique should be avoided for large files, and on Windows 2003 Server the ASPBufferingLimit metabase property will need to be increased if files larger than 4 MB are streamed in this way.

The methods that work directly with files can be used in ASP.NET although brackets must enclose all method parmeters.

It is possible to use the streaming methods by using the aspcompat directive at the start of the script:

<%@ Page language="vb" debug="true" aspcompat="true" %>

5.1. Early Binding

The previous example used late binding, which is the easier way of calling an ASP component in ASP.NET. It is more efficient to use early binding, but this requires the creation of a .NET Framework Interop Assembly using the TLBIMP tool, supplied with the Framework. This assembly is a DLL which acts as a wrapper for the ASP component.

After registering the component, run TLBIMP.exe from the command prompt or from the Run box in the Start Menu. The syntax is:

```
TLBIMP ComponentName.dll /out:NewName.dll
```

Full paths are required for both DLLs. The new DLL needs to be put in the website's BIN directory. The script that calls the component must import the Interop Assembly as a NameSpace. The component instance is created using the following VB.NET syntax:

```
Dim ObjName As New ClassNameClass()
```

ObjName is the name of the object instance and *ClassName* is the name of the class in the ASP component, which is MakeZip or OpenZip in csASPZipFile.

The script that uses the component must import the Interop Assembly as a Namespace. If the Interop Assembly is called "csaspzipfilenet.dll" the following line imports it:

```
<%@ Import Namespace = "csaspzipfilenet" %>
```

In the previous example the only other change required is to replace the Server.CreateObject line with:

```
Dim Zip As New MakeZipClass()
```

6. Revision History

The current version of csASPZipFile is 3.0.

New in Version 1.1

The SaveZip and SaveZipDisk commands have been improved to run faster and allow the creation of larger zip files. The StreamFile command has also been modified to allow downloading of large files.
PromptName property added.

New in Version 2.0

TempFileName added to optionally reduce memory use when saving zip files.
ZipAddDirectory changed from a property to a method.
OpenZip class added to provide functionality to read some zip archives.

New in Version 3.0

64 bit version released.
Fix to the bug which caused some extended ANSI characters in filenames to map incorrectly.

7. Other Products From Chestysoft

Visit the Chestysoft web site for details of other COM objects.

ActiveX Controls

csXImage	-	ActiveX control to display, edit and scan images.
csXGraph	-	ActiveX control to draw pie charts, bar charts and line graphs.

ASP Components

csImageFile	-	Resize, create and edit images.
csDrawGraph	-	Draw pie charts, bar charts and line graphs in ASP.
csASPGif	-	Create and edit animated GIFs.
csASPUpload	-	Process file uploads through a browser.
csFileDownload		Control file downloads with an ASP script.
csFTPQuick	-	ASP component to transfer files using FTP.

ASP.NET

csNetUpload	-	ASP.NET component for saving HTTP uploads.

Web Hosting

We can offer ASP enabled web hosting with our components installed. Click for more details.

8. Alphabetical List of Commands - MakeZip Class

9. Alphabetical List of Commands - OpenZip Class