Table of Contents

TABLE OF CONTENTS 1

INTRODUCTION 3

Conventions Used In This Document: 3

ABOUT THE ALAGAD ZIP COMPONENT 3

Requirements to Use the Zip Component 4

LIMITATIONS OF THE ZIP COMPONENT 4

Password Protected Zip Files 4

Not That Fast… 4

USING THE ZIP COMPONENT ON SHARED HOSTING 5

INSTALLATION OF THE ZIP COMPONENT 5

INSTANTIATING THE ZIP COMPONENT 5

Instantiation from a Local Directory 6
Using <cfobject>: 6
Using CreateObject(): 6

Instantiation from a Mapped Directory 6
Using <cfobject>: 6
Using CreateObject(): 6

Instantiation from a Custom Tag Path 6
Using <cfobject>: 6
Using CreateObject(): 7

USING THE INSTANTIATED ZIP COMPONENT 7

Creating a New Zip and Adding a File 7
Source Code 7

Add a Directory to an existing Zip 7
Source Code 7

Extracting a File from a Zip 7
Source Code 8

Read and Write data directly to a Zip 8
Source Code 8

WORKING WITH ZIP ENTRY PATHS 8

WORKING WITH FILE SYSTEM PATHS 9

ZIP COMPONENT METHODS 9

Working with Zip Files 9
- Overview 9
- addDirectory() 10
- addFile() 11
- deleteDirectory() 11
- deleteFile() 12
- entryExists() 12
- extractAll() 13
- extractDirectory() 13
- extractFile() 14
- getCompression() 15
- getVersion() 15
- init() 16
- isDirectory() 16
- isFile() 17
- list() 17
- readAsBinaryFile() 18
- readAsTextFile() 19
- setCompression() 19
- writeAsBinaryFile() 20
- writeAsTextFile() 21

Introduction

Welcome to the Alagad Zip Component documentation. This documentation is intended to help you integrate the Alagad Zip Component into your ColdFusion application.

The Alagad Zip Component is a 100% native ColdFusion component (CFC) for manipulating zip files. This component requires no third party software to operate and no additional configuration.

This documentation is specific to the Alagad Zip Component. Throughout the rest of the document the Alagad Zip Component will simply be referred to as the Zip Component.

This documentation assumes that you have some experience with ColdFusion and CFCs.

Conventions Used In This Document:

Words surrounded in angle brackets indicate ColdFusion or HTML tags. For example, <cfset> indicates the ColdFusion cfset tag.

Words highlighted in blue followed by parenthesis indicate Alagad Zip Component methods. For example drawRectangle() indicates the drawRectangle method.

ColdFusion and HTML code is highlighted in brown. For example:

```
<cfset example="Hello World" />
```

Paths to files or directories are surrounded in quotation marks. For example "/MyDirectory/".

The "¬" character at the end of a line of example code indicates a line which should be continued on the same line, even though it's shown as wrapping.

About the Alagad Zip Component

The Alagad Zip Component is a ColdFusion Component (CFC) used to create and manipulate zip files. Written entirely in ColdFusion CFML, the Zip Component does not require installation of any additional software. The Zip Component is not a CFX tag and is not Platform dependant.

Because the Zip Component is written in pure ColdFusion and instantiates only native Java objects, it compiles with the rest of your CFML files to Java bytecode and is extremely portable.

Using the Zip Component you can perform a wide range of actions on zip files, including:

- List all files in zip files

- Add files and directories to zip files
- Extract files and directories from zip files
- Delete files and directories from zip files
- Read data directly from zip files into variables
- Write data directly into zip files from variables
- Control zip file compression

The Alagad Zip Component requires almost no effort to install and use. Simply place the component into your custom tags directory or any directory in your site and then instantiate it using the ColdFusion CreateObject method. Once you have the Zip Component instantiated you can use its methods to begin creating and manipulating zip files.

Requirements to Use the Zip Component

The Alagad Zip Component works with ColdFusion MX 6.1 and later on any supported platform.

ColdFusion MX 6.1 users will not need to do any additional configuration for the Alagad Zip Component to work.

Users who are running ColdFusion MX without the 6.1 update will need to install the 6.1 update which is freely available on macromedia.com.

The Zip Component also works BlueDragon 6.1 and later servers, except for the free versions which do not allow usage of precompiled code.

If you are using the Zip Component on a platform which is not supported by ColdFusion MX 6.1, such as OS X, you need to use a 1.4.1 or later JRE.

Limitations of the Zip Component

Because the Zip Component is built on top of functionality provided by the Java Platform, the Zip Component is limited by the capabilities provided by Java. In particular, you should be aware of two notable limitations when designing your application.

Password Protected Zip Files

Java does not provide any mechanism for reading password protected zip files. As a result, the Alagad Zip Component does not support reading or writing password protected Zip files. You will receive error messages if you try to read password protected zip files.

Not That Fast…

Another important fact you should keep in mind is that working zip files can be expensive in terms of CPU usage, memory usage and the amount of time operations take to complete. This fact becomes more apparent when working with large files.

In many cases it may be a good idea to try to work with zip asynchronously or though a scheduled processes. You should consider this when designing your application.

Using the Zip Component on Shared Hosting

The Alagad Zip Component will work in a shared hosting environment. However, many hosting providers use “Sandbox” security to disable certain functionality on the server.

For the Alagad Zip Component to function, your hosting provider may need to enable the CreateObject() function and the <cffile> tag. Contact your hosting provider for more information.

Installation of the Zip Component

Installation of the Alagad Zip Component is quite simple. First, download the Alagad Zip Component in zip format from http://www.alagad.com.

If you are using ColdFusion server you will want to extract the zip.cfc file from “ColdFusion MX 6.1 and later Version” folder. If you are using BlueDragon you will want to extract the zip.cfc file from the “BlueDragon JX 6.2 Version” folder.

If you want to use the Zip Component from any website on your server you can place the zip.cfc file in any custom tags folder.

If you want to use the Zip Component in only one website, copy the zip.cfc file into the website’s root or any directory under your web site’s root.

If you want to place the Zip Component outside of your web site directory hierarchy and out side of any of custom tags folder, then simple copy the zip.cfc file anywhere you want on your file system and create a mapping to that directory using the ColdFusion administration interface. For more information see the ColdFusion documentation.

Instantiating the Zip Component

You must instantiate the Zip Component before you can call its methods. When instantiating the Zip Component you must also call it’s init() method and pass in the path to the zip file you will be working with. This file will be created if it does not already exist.

The following examples demonstrate instantiating the Zip Component using the ColdFusion <cfobject> tag and CreateObject() function. In each example the Zip Component is configured to use a file named “myZipFile.zip” which is in the same directory as the script instantiating the Zip Component.

The expandPath() function is provided by ColdFusion and translates a relative path to an absolute path. For more information see the ColdFusion documentation.

For more information on using ColdFusion components see the Using ColdFusion components section of the ColdFusion documentation.

Instantiation from a Local Directory

If you installed the Zip Component in a directory under your web site's root you would instantiate it as follows below. In both cases we are assuming the zip.cfc file was placed in a directory "/path/to/" under your web site's root. We are creating an instance of the zip.cfc named myZip.

Using <cfobject>:

```
<cfobject component="path.to.Zip" name="myZip" />
<cfset myZip.init(expandPath("myZipFile.zip")) />
```

Using CreateObject():

```
<cfset myZip = CreateObject("Component", ¬
"path.to.Zip").init(expandPath("myZipFile.zip"))) />
```

Instantiation from a Mapped Directory

If you installed the Zip Component in a ColdFusion mapped directory you would instantiate it as follows below. In both cases we are assuming the zip.cfc file was placed in a directory which was mapped to with the logical path "/mapped/path/to/". We are creating an instance of the zip.cfc named myZip.

For more information on ColdFusion mappings see the ColdFusion documentation.

Using <cfobject>:

```
<cfobject component="mapped.path.to.Zip" name="myZip" />
<cfset myZip.init(expandPath("myZipFile.zip")) />
```

Using CreateObject():

```
<cfset myZip = CreateObject("Component", ¬
"mapped.path.to.Zip").init(expandPath("myZipFile.zip")) />
```

Instantiation from a Custom Tag Path

If you installed the Zip Component in a ColdFusion custom tags path you would instantiate it as follows below. In both cases we are creating an instance of the zip.cfc named myZip.

Using <cfobject>:

```
<cfobject component="Zip" name="myZip" />
<cfset myZip.init(expandPath("myZipFile.zip")) />
```

Using CreateObject():

```
<cfset myZip = CreateObject("Component", ¬
"Zip")").init(expandPath("myZipFile.zip")) />
```

Using the Instantiated Zip Component

Once you have instantiated the Zip Component you can call methods on the component to perform Zip related actions such as adding files to and extracting files from zip files.

Below are some complete examples of the most common actions. These examples use the CreateObject() to instantiate the object, but the same process works equally well with <cfobject>.

These examples are also provided in the zip file downloaded from Alagad.com. Look in the examples folder.

Creating a New Zip and Adding a File

The following example demonstrates how to create a new zip file. Compression settings are also shown.

Source Code

```
<!---
Create a new instance of the zip component.  By specifying a zip file
name which doesn't exist you will create a new zip file.
--->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- add a file into the zip file --->
<cfset myZip.addFile(expandPath("bagOfBugers.JPG")) />
```

Add a Directory to an existing Zip

The following source code demonstrates how to add an entire folder to an existing zip file.

Source Code

```
<!---
Create a new instance of the zip component.  The Zip Component will
work with the specified zip file.
--->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- add an entire directory into the zip file --->
<cfset myZip.addDirectory(expandPath("photos")) />
```

Extracting a File from a Zip

The following source code demonstrates how to extract a file from a zip document.

Source Code

```
<!---
Create a new instance of the zip component.  The Zip Component will
work with the specified zip file.
--->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- extract a file from the zip file --->
<cfset myZip.extractFile("bagOfBugers.jpg", expandPath(".")) />
```

Read and Write data directly to a Zip

The following source code demonstrates how you can write directly into a zip file and read that data directly out.

Source Code

```
<!---
Create a new instance of the zip component.  By specifying a zip file
name which doesn't exist you will create a new zip file.
--->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- you can write data directly int a zip file using the
writeAsTextFile method --->
<cfset myZip.writeAsTextFile("example file.txt", "This is some sample
text.") />

<!--- read the text back out of the zip file --->
<cfset text = myZip.readAsTextFile("example file.txt") />

<!--- output the text from the zip file --->
<cfoutput>
      #text#
</cfoutput>
```

Working With Zip Entry Paths

Most methods in the zip component work with paths to “entries” in the zip document. Entries are any file or location in the zip document. Entries are specified in a format similar to relative URLs or paths.

Folders can be specified using either a front slash or back slash. Any path ending in a front slash or back is assumed to be a folder. The lack of a trailing slash may or may not indicate that the path is a file, depending on context.

Though out the zip component, efforts have been made to make zip entry paths as intuitive as possible. If it seems like a particular use of a zip entry path *should* work then it probably will.

So, for example, if you wanted to specify a file named “temp.txt” in a directory named “example” in your zip file, all of the following formats would be legal:

- example/temp.txt
- /example/temp.txt
- example\temp.txt
- \example\temp.txt

Working With File System Paths

In the same way that zip entry paths are supposed to behave intuitively, so should file system paths.

As an example of how the behavior should be intuitive, let's consider the extractFile() method. According to the extractFile()'s documentation the method accepts two required arguments, pathToZipEntry and extractToPath. Depending on how you use the method, the exact meaning of the extractTo varies.

Consider the following examples. These assume you have instantiated the zip component into a variable named myZip.

Note: The expandPath() method is a ColdFusion function which translates relative paths into absolute paths. So, if your script was executing in "c:\inetpub", expandPath(".") would translate to "c:\inetpub\.

```
<cfset myZip.extractFile("/example/file.txt", extractPath(".")) />
```

This usage causes the file to be extracted to "c:\inetpub\example\file.txt".

```
<cfset myZip.extractFile("/example/file.txt", extractPath("example"))/>
```

This usage causes the file to be extracted to "c:\inetpub\example\example\file.txt".

```
<cfset myZip.extractFile("/example/file.txt", extractPath("file.txt"))/>
```

This usage causes the file to be extracted to "c:\inetpub\file.txt".

Zip Component Methods

The Alagad Zip Component has several methods that you can use for reading, creating, and manipulating Zip files.

The following is a list of all public Zip Component methods.

Working with Zip Files

Overview

The Zip Component is designed to work with and manipulate one zip file at a time. When you initialize the zip component you will read an existing zip file or create a new zip file. The zip file data and component's current state is stored in instance variables within the Zip Component. As you call various methods on the Component you are manipulating the state of the Zip and the Zip data.

All operations on the zip file occur at the time the operation is executed. For example, if you use the deleteFile() method to delete a file from the zip document it will immediately be deleted.

If you need to manipulate more than one Zip at the same time, then use multiple instances of the Zip Component. Each instance operates separately.

addDirectory()

Description

The addDirectory() method adds a directory to the zip document. The optional pathToZippedDirectory argument specifies the location where the entry should be added in the zip file. If this argument is not provided the directory is added under the zip file's root directory. By default the directory is added recursively, however this can be disabled by setting the recursive argument to false.

For information on controlling compression settings see the setCompression() method documentation.

If, in the init() method, you do not specify an existing zip file this method will create a new zip file.

Syntax

addDirectory(pathToDirectory, pathToZippedDirectory, recursive)

Parameter	Required	Type	Description
pathToDirectory	Yes	String	This is the path to the folder to add to the zip file.
pathToZippedDirectory	No	String	This indicates where the directory will be placed inside the zip document. If not provided, this defaults to the root directory, or "/".
recursive	No	Boolean	Indicates if the directory is added recursively. Options are: True – (default) All files and directories under the specified directory are added to the zip file. False – Only files directly under the specified directory are added.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- add a directory to the root of the zip document --->
<cfset myZip.addDirectory(expandPath("example ")/>
```

See Also

addFile(), deleteDirectory(), setCompression()

addFile()

Description

The addFile() method adds a file to the zip document. The pathToZipEntry argument specifies the location where the entry should be added in the zip file.

For information on controlling compression settings see the setCompression() method documentation.

If, in the init() method, you do not specify an existing zip file this method will create a new zip file.

Syntax

addFile(pathToFile, pathToZipEntry)

Parameter	Required	Type	Description
pathToFile	Yes	String	This is the path to the file to add to the zip file.
pathToZipEntry	Yes	String	Indicates the directory or file in the zip file where the file should be placed.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- add a file to the zip document --->
<cfset myZip.addFile(expandPath("example.txt"), "example.txt") />
```

See Also

addDirectory(), deleteFile(), setCompression()

deleteDirectory()

Description

The deleteDirectory() method is used to delete an entire directory and all of it's contents from a zip file. This method executes recursively.

Syntax

deleteDirectory(pathToZippedDirectory)

Parameter	Required	Type	Description
pathToZippedDirectory	Yes	String	This specifies the directory to delete from the zipFile

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- delete a directory from the zip document --->
<cfset myZip.deleteDirectory("example.txt") />
```

See Also

deleteFile()

deleteFile()

Description

The deleteFile() method is used to delete a file from a zip document. The pathToZipEntry argument specifies the particular file to delete.

Syntax

deleteFile(pathToZipEntry)

Parameter	Required	Type	Description
pathToZipEntry	Yes	String	This specifies the file to delete from the zipFile

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- delete a file from the zip document --->
<cfset myZip.deleteFile("example.txt") />
```

See Also

addFile(), deleteDirectory()

entryExists()

Description

This method is used to check if an entry with a specific name exists. If the entry exists this method returns true, otherwise it returns false.

Syntax

Boolean = entryExists(pathToZipEntry)

Parameter	Required	Type	Description
pathToZipEntry	Yes	String	This specifies the entry in the zip file to check if it exists.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- check if an entry exists --->
<cfif myZip.entryExists("example.txt")>
      The entry exists!
<cfelse>
      The entry does not exist!
</cfif>
```

See Also

isFile(), isDirectory()

extractAll()

Description

The extractAll() method extracts all files from the zip file and places them in the specified directory. All files are extracted recursively.

Syntax

extractAll(extractToDirectory)

Parameter	Required	Type	Description
extractToDirectory	Yes	String	This specified the directory to extract files into.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- extract all files recursively into the temp directory --->
<cfset myZip.extractAll("c:\temp") />
```

See Also

extractDirectory(), extractFile()

extractDirectory()

Description

The extractDirectory() method is used to extract a directory from a zip file to a specified location. By default this method will extract files and directories under the specified directory recursively.

In other words, if you have a directory in your zip file such as, “directory” and you extract it to “c:\temp”, the zip component will extract all of the files and directories under “directory” to the “c:\temp” folder. This includes “directory/example” and

“directory/example/file.txt”. However, if you set the recursive argument to false only the files directly under “directory” will be extracted into “c:\temp”.

Syntax

extractDirectory(pathToZippedDirectory, extractToDirectory, recursive)

Parameter	Required	Type	Description
pathToZippedDirectory	Yes	String	This specifies the directory in the zip file to extract.
extractToDirectory	Yes	String	This is the location where the files in the specified directory will be extracted.
recursive	No	Boolean	Indicates if files are extracted recursively. True – (default) The all files and directories are extracted from the specified folder. False – Only files directly under the specified folder are extracted.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- extract a directory recursively --->
<cfset myZip.extractDirectory("temp", "c:\temp") />
```

See Also

addDirectory(), extractFile(), extractAll()

extractFile()

Description

The extractFile() method is used to extract a file from a zip file to a specified location. By default this method will maintain the directory hierarchy defined in the zip document.

In other words, if you have a zip entry such as, “directory/file.txt” and you extract it to “c:\temp”, the zip component will extract this file to “c:\temp\directory\file.txt.” However, if you set the preserveDirectories argument to false, or if you specify a particular file in the extractToPath argument, the directory structure will be disregarded and your file will be extracted to “c:\temp\file.txt.”

Syntax

extractFile(pathToZipFile, key)

Parameter	Required	Type	Description
pathToZipEntry	Yes	String	This specifies the entry in the zip file

			to extract.
extractToPath	Yes	String	This is the location where the file will be extracted.
preserveDirectories	No	Boolean	Indicates if the directory structure in the zip file is maintained when the zipped file is extracted. True – (default) The directory structure is maintained. False – The file is written directly into the directory specified.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- extract a file from the zip document --->
<cfset myZip.extractFile("example.txt", expandPath(".")) />
```

See Also

addFile(), extractDirectory(), extractAll()

getCompression()

Description

The getCompression() method returns the current compression level setting for the zip component. This is not the compression setting for any particular file, but the setting which will be used when the next file is added to the zip component.

For more information on the compression level settings see the setCompression() documentation.

Syntax

Compression = getCompression()

See Also

setCompression()

getVersion()

Description

This method returns the version of the zip component as a string.

Syntax

Version = getVersion()

init()

Description

The init() method is used to configure the zip component. As explained in the section entitled "Working with Zip Files", the zip component works with only one zip file at a time. The init method is used configure the file that the zip component instance is working with. If the file specified doesn't exist, it is created. If the file does exist it is used.

This method must be called when the component is instantiated. In the case when you are using the <cfobject> syntax you will need to call this method immediately after instantiating the component. If you are using the createObject() method you can either call this method immediately after instantiating the component or you can chain it onto the createObject() call.

Syntax

Zip = init(pathToZipFile)

Parameter	Required	Type	Description
pathToZipFile	Yes	String	This is the path to the zip file that the component will use. If you specify a file which does not exist it will be created.

Example

```
<!--- init component and open example.zip --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />
```

isDirectory()

Description

This method indicates if a particular entry in the zip file is a directory or not. If the entry does not exist an error is thrown.

Syntax

Boolean = isDirectory(pathToZipEntry)

Parameter	Required	Type	Description
pathToZipEntry	Yes	String	This is the path to the entry in the zip file to check if it is a directory.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- check if an entry is a file or not --->
<cfif myZip.isDirectory("example ")>
      The entry is a directory!
<cfelse>
```

```
      The entry is not a directory!
</cfif>
```

See Also

isFile(), entryExists()

isFile()

Description

This method indicates if a particular entry in the zip file is a file or not. If the entry does not exist an error is thrown.

Syntax

Boolean = isFile(pathToZipEntry)

Parameter	Required	Type	Description
pathToZipEntry	Yes	String	This is the path to the entry in the zip file to check if it is a file.

Example

```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- check if an entry is a file or not --->
<cfif myZip.isFile("example.txt")>
      The entry is a file!
<cfelse>
      The entry is not a file!
</cfif>
```

See Also

isDirectory(), entryExists()

list()

Description

The list method returns a query which describes the contents of a zip file. The query is made up of the following columns:

Directory
This is the directory an entry is under in the zip file.

File
This is the file name for the entry.

FullPath
This is the full path to a file in the zip file. This is the same as if you were to concatenate the File column onto the Directory column.

CompressedSize
This is the size of the file after compression.

UncompressedSize
This is the size of the file when uncompressed.

LastModified
This is the date the file in the zip document was last modified.

Syntax
Query = list()

Example
```
<!--- Read a zip file --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- list the contents of the zip file --->
<cfset list = myZip.list() />

<!--- dump all the resulting query --->
<cfdump var="#list#" />
```

readAsBinaryFile()

Description
The readAsBinaryFile() method can be used to read binary data out of any file in a zip file. The data is returned directly by the method. This is useful in the case when you do not necessarily want to extract a file to disk, but you do need to read the file.

Syntax
Binary = readAsBinaryFile(pathToZipEntry)

Parameter	Required	Type	Description
pathToZipEntry	Yes	String	This is the path to the entry in the zip file that you want to read.

Example
```
<!---
Create a new instance of the zip component.  This will read the
example.zip file.
--->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- read some text from an entry in the zip file --->
<cfset data = myZip.readAsTextFile("example.txt") />

<!--- dump the data --->
<cfdump var="#data#" />
```

See Also

writeAsBinaryFile()

readAsTextFile()

Description

The readAsTextFile() method can be used to read textual data out of any file in a zip file. The data is returned directly by the method. This is useful in the case when you do not necessarily want to extract a file to disk, but you do need to read the file.

Syntax

Text = readAsTextFile(pathToZipEntry)

Parameter	Required	Type	Description
pathToZipEntry	Yes	String	This is the path to the entry in the zip file that you want to read.

Example

```
<!---
Create a new instance of the zip component.  This will read the
example.zip file.
--->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- read some text from an entry in the zip file --->
<cfset text = myZip.readAsTextFile("example.txt") />

<!--- output the text --->
<cfoutput>
      #text#
</cfoutput>
```

See Also

writeAsTextFile()

setCompression()

Description

The setCompression() method sets the compression levels for files added to the zip file. Low values indicate less compression. High values indicate more compression. The default compression level is 6.

The results of using this method can be counter intuitive. For example imagine that you do the following steps:

1) Add a file to a zip document using the default compression level of 6
2) Change the compression level using setCompression() to 9.
3) Add another file to the zip document.

You might expect that the first file would still have a compression level of 6. This is *not* true. Both files will now have a compression level of 9. The opposite is true too. If the initial compression level was 9, and the second file was added at 6 then both files will have a compression level of 6

This is due to how the Zip Component compensates for the limitations inherent in Java. Java does not have a robust set of classes for working with Zip files. As a result it's not technically possible to simply add a new file to a zip document. The Zip Component works around this by creating a new zip file each time you add a file to your zip file. The new zip file is created in a temp directory and all of the files in the first document are copied into the new document and then the new file is added. Once created, the new zip file is moved to replace your old zip file.

The implication of this is that any time you change the compression settings and then add a new file, the old entries in your zip file will be recompressed using the new level as they are moved into the new zip document.

Syntax

setCompression(compression)

Parameter	Required	Type	Description
Compression	Yes	Numeric	This argument sets the compression level used when adding data to the zip file. Valid options are 0 to 9. 0 is the lowest level of compression. 9 is the maximum amount of compression.

Example

```
<!--- Create a new instance of the zip component.  By specifying a zip
file name which doesn't exist you will create a new zip file. --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- set the compression level to maximum amount --->
<cfset myZip.setCompression(9) />

<!--- add a file to the zip file --->
<cfset myZip.addFile(expandPath("Documentation.doc")) />
```

See Also

getCompression()

writeAsBinaryFile()

Description

The writeAsBinaryFile() method is used to write binary data directly into a zip file. This is useful in cases where you have data which may exist in a variable or which may be returned from a function call, but which does not exist as a file on

disk. This data can easily be added to the zip file by using the writeAsBinaryFile() method.

By using this method you can avoid writing data to disk before adding your binary data to the zip file.

Syntax

writeAsBinaryFile(pathToZippedFile, contents)

Parameter	Required	Type	Description
pathToZippedFile	Yes	String	The path to the entry in the zip file that the contents will be written to. If the entry does not exist it will be created. If it does exist it will be overwritten.
contents	Yes	Binary	The binary contents to write to the file. This must be a binary value.

Example

```
<!--- Create a new instance of the zip component.  By specifying a zip
file name which doesn't exist you will create a new zip file. --->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- create some binary data --->
<cfset binData = ToBinary(ToBase64("This is some sample data")) />

<!--- write the binary data to the zip file --->
<cfset myZip.writeAsBinaryFile("data.bin", binData) />

<!--- read the binary data and dump it out --->
<cfdump var="#myZip.readAsBinaryFile("data.bin")#">
```

See Also

readAsBinaryFile()

writeAsTextFile()

Description

The writeAsTextFile() method is used to write directly into a zip file. This is useful in cases where you have data which may exist in a variable or which may be returned from a function call, but which does not exist as a file on disk. This data can easily be added to the zip file by using the writeAsTextFile() method.

By using this method you can avoid writing data to disk before adding your text data to the zip file.

Syntax

writeAsTextFile(pathToZippedFile, contents)

Parameter	Required	Type	Description

pathToZippedFile	Yes	String	The path to the entry in the zip file that the contents will be written to. If the entry does not exist it will be created. If it does exist it will be overwritten.
contents	Yes	String	The contents to write to the file. This must be a string value.

Example

```
<!---
Create a new instance of the zip component.  By specifying a zip file
name which doesn't exist you will create a new zip file.
--->
<cfset myZip = CreateObject("Component", ¬
"zip.zip").init(expandPath("example.zip")) />

<!--- You can write data directly in to a zip file using the
writeAsTextFile method --->
<cfset myZip.writeAsTextFile("example file.txt", "This is some sample ¬
text.") />

<!--- read the text back out of the zip file --->
<cfset text = myZip.readAsTextFile("example file.txt") />

<!--- output the text from the zip file --->
<cfoutput>
      #text#
</cfoutput>
```

See Also

readAsTextFile()