

Optimized compression tools for data archiving and transfer



# Abbrevia<sup>TM</sup> 3

Squeeze the most from your application with TurboPower's all-in-one compression and archiving toolkit

- *Build industry-standard PKZip®, zlib, gzip, tar and CAB compression into your applications*
- *Archive and maintain multiple files in just a single file with our new Compound File Engine.*
- *Works across development environments, supporting Windows with Delphi and C++Builder, and Linux with Kylix*



# Abbrevia 3<sup>TM</sup>

TurboPower Software Company  
Colorado Springs, CO

[www.turbopower.com](http://www.turbopower.com)

© 1997-2001 TurboPower Software Company. All rights reserved.

First Edition August 1997  
Second Edition May 1999  
Third Edition May 2001

## License Agreement

This software and accompanying documentation are protected by United States copyright law and also by International Treaty provisions. Any use of this software in violation of copyright law or the terms of this agreement will be prosecuted to the best of our ability.

Copyright © 1997-2001 by TurboPower Software Company, all rights reserved.

TurboPower Software Company authorizes you to make archival copies of this software for the sole purpose of back-up and protecting your investment from loss. Under no circumstances may you copy this software or documentation for the purposes of distribution to others. Under no conditions may you remove the copyright notices made part of the software or documentation.

You may distribute, without runtime fees or further licenses, your own compiled programs based on any of the source code of Abbrevia. You may not distribute any of the Abbrevia source code, compiled units, or compiled example programs without written permission from TurboPower Software Company.

Note that the previous restrictions do not prohibit you from distributing your own source code, units, or components that depend upon Abbrevia. However, others who receive your source code, units, or components need to purchase their own copies of Abbrevia in order to compile the source code or to write programs that use your units or components.

The supplied software may be used by one person on as many computer systems as that person uses. Group programming projects making use of this software must purchase a copy of the software and documentation for each member of the group. Contact TurboPower Software Company for volume discounts and site licensing agreements.

This software and accompanying documentation is deemed to be "commercial software" and "commercial computer software documentation," respectively, pursuant to DFAR Section 227.7202 and FAR 12.212, as applicable. Any use, modification, reproduction, release, performance, display or disclosure of the Software by the US Government or any of its agencies shall be governed solely by the terms of this agreement and shall be prohibited except to the extent expressly permitted by the terms of this agreement.

With respect to the physical media and documentation provided with Abbrevia, TurboPower Software Company warrants the same to be free of defects in materials and workmanship for a period of 60 days from the date of receipt. If you notify us of such a defect within the warranty period, TurboPower Software Company will replace the defective media or documentation at no cost to you.

TurboPower Software Company warrants that the software will function as described in this documentation for a period of 60 days from receipt. If you encounter a bug or deficiency, we will require a problem report detailed enough to allow us to find and fix the problem. If you properly notify us of such a software problem within the warranty period, TurboPower Software Company will update the defective software at no cost to you.

TurboPower Software Company further warrants that the purchaser will remain fully satisfied with the product for a period of 60 days from receipt. If you are dissatisfied for any reason, and TurboPower Software Company cannot correct the problem, contact the party from whom the software was purchased for a return authorization. If you purchased the product directly from TurboPower Software Company, we will refund the full purchase price of the software (not including shipping costs) upon receipt of the original program media and documentation in undamaged condition. TurboPower Software Company honors returns from authorized dealers, but cannot offer refunds directly to anyone who did not purchase a product directly from us.

**TURBOPOWER SOFTWARE COMPANY DOES NOT ASSUME ANY LIABILITY FOR THE USE OF Abbrevia BEYOND THE ORIGINAL PURCHASE PRICE OF THE SOFTWARE. IN NO EVENT WILL TURBOPOWER SOFTWARE COMPANY BE LIABLE TO YOU FOR ADDITIONAL DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE THESE PROGRAMS, EVEN IF TURBOPOWER SOFTWARE COMPANY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.**

By using this software, you agree to the terms of this section and to any additional licensing terms contained in the DEPLOY.HLP file. If you do not agree, you should immediately return the entire Abbrevia package for a refund.

All TurboPower product names are trademarks or registered trademarks of TurboPower Software Company. Other brand and product names are trademarks or registered trademarks of their respective holders.

---

# Table Of Contents

<b>Chapter 1: Introduction .....</b>	<b>1</b>
System Requirements .....	3
Installation .....	4
Organization of this Manual .....	5
Technical Support .....	7
Unified Modeling Language (UML) .....	8
<b>Chapter 2: Overview of Compression .....</b>	<b>13</b>
PKZIP's Compression Techniques .....	15
Abbrevia's Zip Compression Capabilities .....	17
Abbrevia's ZLib Compression Capabilities .....	19
Abbrevia's GZip and TAR Compression Capabilities .....	20
Cabinet File Compression Techniques .....	21
Abbrevia's Cabinet File Compression Capabilities .....	22
Compression Used by Abbrevia's Compound Storage .....	23
Additional Reading .....	24
<b>Chapter 3: Tutorials and Tours .....</b>	<b>25</b>
Tutorial 1: TAbZipBrowser - Searching for files .....	26
Tutorial 2: TAbCabBrowser: Contents of a Cabinet .....	31
Tutorial 3: TAbZipKit: Stream Compression .....	34
Tutorial 4: TAbMakeSelfExe - Creating a Self-Extracting Zip Archive .....	37
Tour 1: TPZip .....	39
Tour 2: ZipView .....	47
<b>Chapter 4: Base Browser Component .....</b>	<b>53</b>
TAbBaseComponent Class .....	54
TAbBaseBrowser Class .....	56
<b>Chapter 5: Non-Visual Zip Components .....</b>	<b>69</b>
TAbCustomZipBrowser Class .....	70
TAbZipBrowser Component .....	71
TAbCustomUnZipper Class .....	74
TAbUnZipper Component .....	75
TAbCustomZipper Class .....	82
TAbZipper Component .....	83
TAbCustomZipKit Class .....	98
TAbZipKit Component .....	99

<b>Chapter 6: Cabinet Components (Windows Only) .....</b>	<b>107</b>
TAbCustomCabBrowser Class .....	108
TAbCabBrowser Component .....	109
TAbCustomCabExtractor Class .....	112
TAbCabExtractor Component .....	113
TAbCustomMakeCab Class .....	118
TAbMakeCab Component .....	119
TAbCustomCabKit Class .....	125
TAbCabKit Component .....	126
<b>Chapter 7: TAbZipOutline Visual Component .....</b>	<b>133</b>
TAbCustomZipOutline Class .....	134
TAbZipOutline Component .....	135
<b>Chapter 8: Archive Viewer Components .....</b>	<b>173</b>
TAbColors Class .....	174
TAbBaseViewer Class .....	176
TAbZipView Component .....	185
TAbCabView Component .....	187
<b>Chapter 9: Compound File Classes .....</b>	<b>189</b>
<b>Chapter 10: Low-Level Compression Classes .....</b>	<b>203</b>
TAbArchiveItem Class .....	204
TAbArchive Class .....	210
TAbArchiveStreamHelper Class .....	237
TAbZipItem Class .....	242
TAbZipArchive Class .....	247
TAbZipStreamHelper Class .....	259
TAbGZipItem Class .....	260
TAbGZipArchive Class .....	267
TAbGZipStreamHelper Class .....	268
TAbTarItem Class .....	269
TAbTarArchive Class .....	277
TAbTarStreamHelper Class .....	278
TAbZLibStream Class .....	279
TAbCabItem Class .....	280
TAbCabArchive Class .....	282
<b>Chapter 11: Miscellaneous Components, Classes, and Routines .....</b>	<b>289</b>
TAbMeter Component .....	290
TAbMakeSelfExe Component .....	293

TAbDirDlg Class .....	296
Zip File Association Routines (Windows Only) .....	299
Low-level Deflate Compression .....	301
TheTAbDeflateHelper Class .....	303
Simple Stream Compression Routines .....	310
<b>Chapter 12: Abbrevia 3 COM Automation Object (Windows Only) .....</b>	<b>313</b>
What's an Automation Object? .....	314
The Abbrevia Type Library .....	315
System Requirements .....	316
Installation and Object Registration .....	317
Registering the Abbrevia COM Object .....	318
Using the Abbrevia COM Object with Microsoft Internet Information Server .....	319
Using the Abbrevia Automation Object with Microsoft Visual Basic 6 .....	320
IZipItem Object .....	321
IIGZipItem Object .....	332
ITARZipItem Object .....	334
IZipKit Object .....	337
<b>Identifier Index .....</b>	<b>i</b>
<b>Subject Index .....</b>	<b>vii</b>



---

# Chapter 1: Introduction

Abbrevia is a set of components that allows you to add file archiving and data compression to your applications. Using Abbrevia, your application can organize data (by grouping files into an archive) and save disk space (by compressing the data).

A file archive is a collection of related files that are maintained as a single unit. Common archive procedures include browsing, adding, extracting, and deleting files from the archive.

Data compression is a process that manipulates a collection of data and produces a representation of that data in less space. There are two types of compression: lossless compression and lossy compression. Data compressed using a lossless compression technique can be completely recovered with 100% accuracy. Important examples of lossless compression include the compression used in backup and restore tools, disk compression tools such as Stacker, the Microsoft CAB format, and the PKZIP tools developed by PKWare, Inc. Lossy compression techniques allow for some loss of accuracy. Lossy compression is often used for compressing graphic, voice, or video data. JPEG is an example of a lossy compression technique.

The components in Abbrevia implement PKZIP-compatible, TAR, GZip, and (on Windows) Microsoft CAB file format data compression and archiving.

PKZIP (the originator of .ZIP files) is a widely used tool for data compression and file archival, developed by Phil Katz of PKWare, Inc. PKZIP is distributed via shareware and commercial channels. The PKZIP file formats and algorithms are published by PKWare.

TAR and GZip are well known tools on the Linux platform for creating compressed collections of files. TAR (an acronym for Tape ARchive) is a method for concatenating a number of files into a single large file. It was originally used for backing up files to tape storage, hence the name.

GZip is a format for compressing data using a method similar to that of PKZIP. Unlike Zip, however, the GZip format does not handle enclosing multiple files well, so the nearly universal practice is use TAR to make a collection of the set of files and then GZip the resulting “TARBall” to compress it.

Abbrevia provides routines and components that can handle TAR Archives, GZips containing a single file, and the common combination of a GZipped TAR.

Abbrevia includes classes for creating compound files that essentially encapsulate an entire file system within a single disk file. Abbrevia's compound file is roughly equivalent to Microsoft's Structured Storage model, but without COM interfaces. Files within a compound file are automatically compressed when they are added and decompressed when extracted.

The Abbrevia cabinet components implement CAB file format data compression and archiving via the Microsoft CABINET.DLL. The Microsoft CABINET.DLL is a system DLL that is installed with 32-bit versions of Microsoft Windows. Cab files can be created using MSZip or LZX compression, or simply stored.

Abbrevia provides automatic disk spanning compatible with that performed by PKZIP. When an archive fills a removable disk or reaches a configurable threshold size, Abbrevia automatically prompts for another disk. Abbrevia can also extract files from spanned disk archives. (Note: On Linux this functionality is subject to proper floppy drive mount and unmount permissions, and the floppy file system should be mounted as a DOS Compatible FAT format of some type.)

Abbrevia also provides automatic file spanning to files of a specified size. When an archive reaches a configurable threshold size, Abbrevia automatically prompts for a new file name (or can generate a default). Abbrevia can also extract files from Abbrevia created spanned file sets (this function is special to Abbrevia and is not part of the formal ZIP file specification, so other ZIP handling programs will likely not recognize such spanned file sets).

Abbrevia supports creation of self-extracting Zip archives. A self-extracting archive is an executable file that, when executed, can extract files contained within itself. Abbrevia can read and manipulate existing self-extracting archives or create new ones. The programmer can create custom self-extracting “stub” programs for this purpose.

Abbrevia provides both non-visual and visual components with full source code.

---

# System Requirements

To use Abbrevia in Windows, you must have the following hardware and software:

- A computer capable of running Windows 9x, Me, NT, or Windows 2000.
- Delphi version 3 or later, or C++Builder version 3 or later. Abbrevia 3 does not support Delphi 1, Delphi 2, or C++Builder 1.
- A hard disk with at least 10 MB of free space is strongly recommended. To install all Abbrevia files and compile the example programs requires about 3 MB of disk space.

To use Abbrevia in Linux, you must have the following hardware and software:

- A computer capable of running one of the supported distributions of Linux.
- Kylinx version 1 or later.
- A hard disk with at least 10 MB of free space is strongly recommended. To install all Abbrevia files and compile the example programs requires about 3 MB of disk space.

## Installation

---

Install Abbrevia directly from the TurboPower Product Suite CD. Simply insert the CD into your CD-ROM drive, select Abbrevia 3 from the list of products, click “Install”, and follow the instructions.

If the TurboPower introductory splash screen does not appear automatically upon insertion of the CD, do the following:

- In Windows, run *X:\CDROM.EXE* where *X* is the letter of your CD-ROM drive.
- In Linux, run `/mnt/cdrom/BrowseCD.sh` (this assumes your CD-ROM drive is mounted on `/mnt/cdrom`).

---

# Organization of this Manual

Each chapter starts with an overview of the classes and components discussed in that chapter. The overview also includes a hierarchy for those classes and components. Each class and component is then documented individually, in the following format:

## Overview

A description of the class or component.

## Hierarchy

Shows the ancestors of the class being described, generally stopping at a VCL class. The hierarchy also lists the unit in which each class is declared and the number of the first page of the documentation of each ancestor. Some classes in the hierarchy are identified with a number in a bullet: ①. This indicates that some of the properties, methods, or events listed for the class being described are inherited from this ancestor and documented in the ancestor class.

## Properties

Lists all the properties in the class. Some properties may be identified with a number in a bullet: ①. These properties are documented in the ancestor class from which they are inherited.

## Methods

Lists all the methods in the class. Some methods may be identified with a number in a bullet: ①. These methods are documented in the ancestor class from which they are inherited.

## Events

Lists all the events in the unit. Some events may be identified with a number in a bullet: ①. These events are documented in the ancestor class from which they are inherited.

## Reference Section

Details the properties, methods, and events of the class or component. These descriptions are in alphabetical order. They have the following format:

- Declaration of the property, method, or event.
- Default value for properties, if appropriate.
- A short, one-sentence purpose. A ↗ symbol is used to mark the purpose to make it easy to skim through these descriptions.
- Description of the property, method, or event. Parameters are also described here.

- Examples are provided in many cases.
- The “See also” section lists other properties, methods, or events that are pertinent to this item.

Throughout the manual, the  symbol is used to mark a warning or caution. Please pay special attention to these items.

## Naming conventions

To avoid class name conflicts with components and classes included with the compiler or from other third party suppliers, all Abbrevia class names begin with “TAb.” The “Ab” stands for Abbrevia.

“Custom” in a component name means that the component is a basis for descendant components. Components with “Custom” as part of the class name do not publish any properties. Instead, descendants publish the properties that are applicable to the derived component. If you create descendant components, use these custom classes instead of descending from the component class itself.

## On-line help

Although this manual provides a complete discussion of Abbrevia, keep in mind that there is an alternative source of information available. Once properly installed, help is available from within the IDE. Pressing <F1> with the caret or focus on an Abbrevia property, routine or component displays the help for that item.

---

## Technical Support

The best way to get an answer to your technical support questions is to post it in the Abbrevia newsgroup on TurboPower's news server ([news.turbopower.com](http://news.turbopower.com)). Many of TurboPower's customers find the newsgroups a valuable resource where they can learn from other's experiences and share ideas in addition to getting answers to questions.

To get the most from the newsgroups, it is recommended that you use dedicated newsreader software.

These newsgroups are public, so please do not post your product serial number, unlocking code, or any other private information (such as credit card numbers) in your messages.

The TurboPower KnowledgeBase ([www.turbopower.com/search](http://www.turbopower.com/search)) is another excellent support option. It has hundreds of articles about TurboPower products accessible through an easy-to-use search engine. The KnowledgeBase is open 24 hours a day, 7 days a week, so you will have another way to find answers to your questions even when TurboPower tech support personnel are not available.

Other support options are described in the support brochure included with Abbrevia. You can also read about support options at [www.turbopower.com/support](http://www.turbopower.com/support).

# Unified Modeling Language (UML)

The Unified Modeling Language (UML) is a visual modeling language intended to depict the design of object-oriented systems. UML consists of assorted graphical notations and a specification for combining these notations to describe the various aspects of an object-oriented system. It can be used to capture and portray both the static structure of a system and its dynamic behavior. Classes, objects, states, use cases, and components, (among other things) can all be graphically modeled with the UML.

At TurboPower Software, the UML is used to describe the design and requirements for a given system. Since UML is such a standardized way of showing relationships between aspects of a system, it makes sense for our manuals to use UML to show the associations and interactions between the various classes and components in the product.

This section provides a small primer on basic UML, enough to read the figures in this manual. If you want more information, please see *UML Distilled* by Martin Fowler (ISBN 0-201-32562-2).

Figure 1.1 shows the UML representation of a typical class. A standard class is a rectangle, divided into three compartments: the class name, the property listing, and the method listing. Both the property and method listings can be suppressed for clarity. (Sometimes the property compartment is known as the attribute compartment, but since the VCL name for these entities is “property”, we shall not use the word “attribute”.)

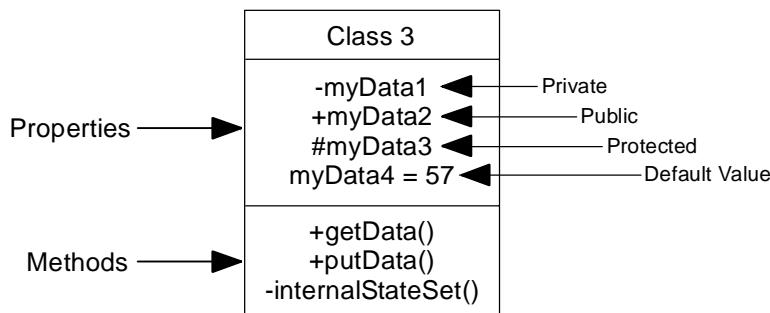


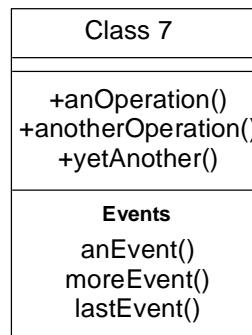
Figure 1.1: An example of a class in UML

Figure 1.1 also shows whether properties and methods are private, protected or public. A special character prior to the name shows the visibility of the identifier. If there is no such special character, no assumption about visibility can be made. Table 1.1 shows the visibility characters and their meaning.

**Table 1.1:** *Table of visibility characters*

Character	Meaning
-	Private
#	Protected
+	Public
	Not shown

The UML specification allows for additional compartments within a class notation. Events are shown as a fourth compartment as shown in Figure 1.2. The standard compartments of class name, properties, and methods are not required by UML to have a name, but additional compartments are. Figure 1.2 shows this by giving the bottom compartment a name of Events. Since events are always public, the visibility character is omitted. Notice that in Figure 1.2 the property compartment is suppressed.



*Figure 1.2: A component with events in UML*

Classes are combined in class diagrams. These diagrams show the relationships between the classes. In TurboPower Software products, *inheritance* and *containment* are the main types of static relationships that can be shown.

Inheritance using UML is shown in Figure 1.3. In this diagram class 2 inherits from class 1; in other words, the arrow points to the ancestor.

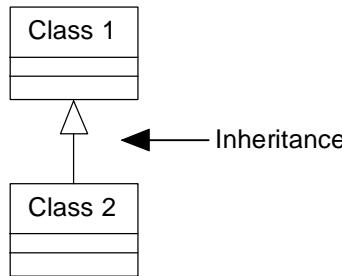


Figure 1.3: Class inheritance in UML

Containment can be shown in one of two ways: composition (containment by value) and aggregation (containment by reference). The two different kinds of containment cause some confusion. Typically, aggregation implies a set of one or more other objects used by the container object. For example, a polygon object would have a list of point objects describing the shape of the polygon. We can alter this list by adding more points or changing their values. The list of points is an *aggregation*.

The polygon might also have an object within it that described how the polygon is to be drawn: the boundary color, the fill color, whether a shadow has to be shown, etc. This object is totally contained within the polygon object, and although you can change the properties of the object, you cannot replace it with another. This state of affairs is known as *composition*. The contained object is viewed as an indelible part of the container.

Composition is shown with a black diamond and aggregation by a white diamond. In Figure 1.4, class 4 contains class 6 through composition and class 4 contains class 5 by aggregation.

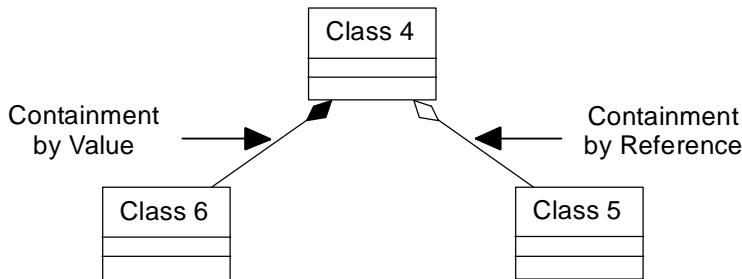


Figure 1.4: Class continuation in UML

A UML sequence diagram shows the dynamic interactions between classes and *actors* along a time line. (An actor is an entity that lies outside the system being depicted. The actor may be a person but equally well may be a server, a different system, or anything else.) A sequence diagram shows class instances as boxes at the top of vertical lines called lifelines. Time is assumed to flow from top to bottom. The sequence diagram helps with the visualization of the order of events happening in the system.

A close-ended arrow between two lifelines represents a method call or a message being sent; an a dashed line with a closed arrow heas shows a return value. Method calls are nearly always labeled with the method names. Return values typically list the data being returned. *Self-delegation* is when an instance calls a method on itself and is shown by a close-ended arrow looping back to the same lifeline.

In a sequence diagram, instances are shown as the instance name followed by a colon and then the class name. Class names are always shown, but instance names are optional.

Figure 1.5 shows an example sequence diagram. Here the user calls the Start method of Class 1. This in turn causes Class 1 to call the getData method of Class 3. Class 3 then calls its own internalSetState method, followed by a call to the Connect method of the database. It returns data to Class 1.

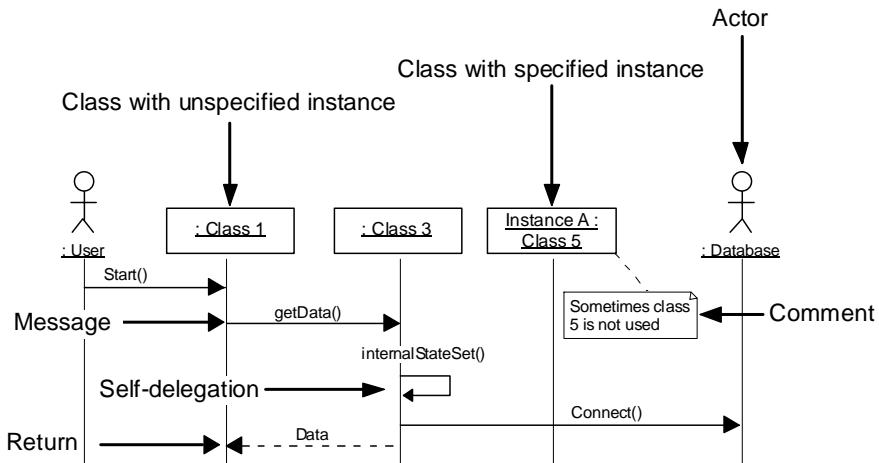


Figure 1.5: Sequence diagram in UML

Comments can be attached to all UML diagrams. They are shown as a box with a folded top left corner containing the comment text. Comments can be attached to items for clarity via a dashed line.



# Chapter 2: Overview of Compression

Data compression is generally achieved by studying the data and looking for redundant patterns and probabilities. This is a basic concern of information theory, a branch of mathematics that was founded by Claude Shannon at Bell Labs. Information theory can be used to determine the entropy, or information content, of a message. The entropy of a message describes the minimum amount of space that must be used to fully describe the message. In general, if the information content of a message is smaller than the existing representation of the message, the message is compressible.

Data compression methods generally operate by determining a model for the data, and then transforming the data into codes based on the model. The model of the data represents the probability of occurrence of certain symbols or strings of symbols in the data. More probable sequences of symbols are represented by shorter codes. Less probable sequences are represented by longer codes. This generally results in encoded data that takes less space than the decoded data.

A common technique for encoding data based on probability of occurrence is Shannon-Fano coding. This method was developed independently by Claude Shannon and R.M. Fano at the Massachusetts Institute of Technology. Each character is given a unique code with a different byte length. Shannon-Fano codes are easily arranged into a binary tree structure. Traversing a Shannon-Fano tree to search for the symbol given its code is very fast.

A similar technique is Huffman coding which is slightly more efficient than Shannon-Fano coding because it creates shorter codes for the most common elements. Since it requires roughly the same amount of computation, Huffman coding is more commonly used.

Simple compression techniques use static models of sample data to represent all similar data. If the actual data to be compressed is not similar to the model, this technique performs poorly. More modern techniques use adaptive modeling, which allows the model of the data to be updated as the document is compressed. As the document is decompressed, the model controlling the decompression is also updated. The first commonly known adaptive algorithm was developed in 1977 by Jacob Ziv and Abraham Lempel, and is known as LZ77. A significant feature of LZ77 is that it uses an adaptive dictionary technique.

Dictionary techniques are often used in concert with statistical modeling techniques. Dictionary techniques allow for brief replacements of commonly used phrases. This is similar to techniques used for references in a technical document. For example, once an author has referred to “See Knuth, D.E., *The Art of Computer Programming*, Volume 1., Fundamental Algorithms, Stanford, 1973,” it is possible to use simply “See Knuth” at later points in the document.

Dictionary techniques generally fall into two categories: *static* and *adaptive*. A static dictionary technique creates the dictionary and then uses it for the complete compression process. An example of a static dictionary might be a list of common phrases. As a phrase is matched in the uncompressed text, a code is stored to represent the phrase.

An adaptive dictionary is modified as compression occurs. This allows the dictionary to be built as processing occurs, and also improves the compression technique because the dictionary is built using the data under consideration. A simple method of creating an adaptive dictionary is to use a window of previously occurring text as the dictionary. As processing continues, the window slides forward to look at recent data. This is known as a sliding dictionary technique.

In 1978 Lempel and Ziv published “Compression of Individual Sequences via Variable-Rate Coding” in *IEEE Transactions on Information Theory*. The technique, known as LZ78, does not use a sliding window technique, but rather maintains a dictionary of previously seen phrases. The uncompressed data is converted to a series of tokens. The tokens have two components: a phrase location and the character that follows the phrase. The phrase dictionary is initialized with only a null string. When a phrase in the uncompressed stream is not found in the dictionary, it is added, one character at a time, to the dictionary. Theoretically, the dictionary can grow in an unlimited fashion. As the dictionary size increases, the code size must increase.

In 1984, Terry Welch published a variant of LZ78 that is known as LZW (Lempel-Ziv-Welch). It improves on LZ78 by initializing the dictionary with all single symbol phrases possible in the alphabet. Another common improvement to LZ78 is to monitor the performance of the compression, and thus determine if the dictionary is poorly matched to the input. If so, it may be advantageous to clear the dictionary and rebuild it with the new input.

Although this is only a brief introduction to the concepts of compression, it is all you need to know to understand and use Abbrevia. If you want to learn more about data compression, see “Additional Reading” on page 24.

---

## PKZIP's Compression Techniques

PKZIP includes support for six lossless compression techniques: storing, shrinking, reducing, imploding, and deflating. Within a PKZIP archive, any combination of these techniques can be used. Complete descriptions of the techniques are included in the file APPNOTE.TXT, included with registered versions of PKZIP.

No matter what compression technique is used, a file in an archive can optionally be encrypted. The encryption process uses a user-supplied password to encode the file. The file can be decrypted (converted back to its original form) only if the same password is used. This makes unauthorized access to the file more difficult. See "Encryption" on page 17 for more information.

### **Storing**

Storing is not really a compression technique. The data is stored "as-is" with no compression at all. Storing is supported in PKZIP to allow archiving. Files are often stored in an archive because they cannot be compressed efficiently. For example, very small files and previously compressed files cannot usually be compressed efficiently and are therefore just stored.

### **Shrinking**

PKZIP's shrinking algorithm uses a modified dynamic LZW compression algorithm. The differences are in the conditions that cause the code size to be increased and the conditions that cause the table to be cleared.

### **Reducing**

Reducing combines a dictionary technique to first compress repeated byte sequences and then a probabilistic method to compress the result. PKZIP includes four different levels of reduction.

## Imploding

Imploding combines two techniques. First, a sliding dictionary technique compresses repeated byte sequences. The output is then encoded using multiple Shannon–Fano trees. Either two or three Shannon–Fano trees are used (the archive item's header includes details about the implementation).

## Deflating

Deflating is similar to Imploding, but the size of the sliding dictionary is increased from 4KB or 8KB to 32KB. The secondary compression uses a combination of Huffman and Shannon–Fano Codes.

PKZIP 4.0 introduced Deflate64; a variant of the Deflate algorithm described on page 15, but using a 64KB dictionary. This allows for even greater compression levels; but is also somewhat slower than traditional deflate.

---

## Abbrevia's Zip Compression Capabilities

Abbrevia can extract any file from a PKZIP-compatible archive. All five of the compression methods used by PKZIP are supported for extracting files. For adding files to a PKZip compatible archive, Abbrevia does not support the older compression methods (i.e., shrinking, reducing, and imploding). Because deflating is the most efficient and most frequently used form of compression, Abbrevia uses it for compressing files. Abbrevia also supports storing for files that cannot be compressed efficiently, such as graphics files (e.g. JPEG) whose data is already compressed.

### Encryption

Abbrevia supports PKZIP file encryption. It can decrypt files that were encrypted by PKZIP and can also encrypt files as they are added to an archive.

A file is encrypted after it is compressed, and is decrypted before it is expanded. The encrypted file is stored. Only the file itself is encrypted, the archive directory information is not encrypted. Each file is encrypted using a user-supplied password, which is not stored in the archive. Each file in an archive can have a different password.

When you extract an encrypted file, you must supply the same password that was used to encrypt it. An archive can contain any combination of encrypted files (each possibly having a distinct password) and files that are not encrypted.

Abbrevia allows passwords up to 255 characters long. Passwords should generally be long (more than 8 characters) with a mixture of upper and lower case characters and punctuation. This makes unauthorized access to the encrypted file more difficult.

### Self-extracting archives

A self-extracting archive is an executable file that, when executed, can extract files contained within itself. Abbrevia can operate on a self-extracting archive created by PKZIP. In addition, Abbrevia allows you to create self-extracting archives. See "TAbMakeSelfExe Component" on page 37 for more information.

### Spanning

Abbrevia supports PKZIP-compatible spanned (removable disk) archives. Additionally, Abbrevia can create and extract files from spanned archive images residing on fixed drives. A spanned archive is a good way to save a large file (or set of files) to multiple floppy disk images.

When Abbrevia creates a spanned archive set or extracts files from a spanned set, it prompts the user to insert the appropriate disks (or specify the appropriate archive image file name). You can customize the prompts using the OnRequestBlankDisk, OnRequestImage, OnRequestLastDisk and OnRequestNthDisk events.

Note that on Linux spanning to floppy disks is subject to proper floppy drive mount and unmount permissions. The floppy file system should be mounted as a DOS Compatible FAT format of some type

## Stream compression

Abbrevia supports two types of stream compression and decompression. First, with Abbrevia you can create a Zip item in an archive directly from a stream and extract an item from a zip archive directly to a stream. Adding an item directly from a stream involves storing the same type of PKZIP compatible header information into the archive.

The second type of stream compression and decompression simply takes an uncompressed stream and compresses its contents to a compressed stream. No PKZIP compatible header information is involved. Likewise, a compressed stream can be similarly decompressed. See the Chapter 11, “Miscellaneous Components, Classes, and Routines”, for more information about this type of simple stream compression.

---

## Abbrevia's ZLib Compression Capabilities

The Deflate engine in Abbrevia is general purpose in that it makes no assumptions regarding the source of data to be compressed or final destination of uncompressed data. As such, it may be used with other formats that make use of the Deflate algorithm for compression purposes.

In particular ZLib is a common method for adding compressed data to other file formats. The .PNG graphic format is a well-known example.

Abbrevia supports ZLib encoded streams via its TAbZLibStream class. This class may be instantiated by attaching to a stream containing ZLib formatted data from some other source (e.g. a .PNG graphics file) and can expand such data as required.

Conversely, TAbZLibStream can be attached to a stream being written to and provide ZLib compression of data into that stream.

---

## Abbrevia's GZip and TAR Compression Capabilities

Another use of the Abbrevia Deflate engine is in support of the GZip format. GZip is a common means for archiving files on Unix and Linux systems that also uses the Deflate algorithm for compressing data.

In general Abbrevia's GZip handling is transparently handled by the main archive components (AbZipper, AbZipKit, AbUnzipper) based on the file extension. You also have access to lower level classes for working with GZip data (files and stream) directly if desired.

Note that you can use the TAbMakeSelfExe component described under "Self-extracting archives" on page 17 on GZipped files as well.

TAR (an acronym for "Tape ARchive") is another archive format common on the Unix and Linux platforms. TAR is not really a compression format in that it simply concatenates the files, as-is, along with some informational data.

In general Abbrevia's TAR handling is transparently handled by the main archive components (AbZipper, AbZipKit, AbUnzipper) based on the file extension. You also have access to lower level classes for working with TAR data (files and stream) directly if desired.

A common practice due to TAR's lack of compression and GZip's relative awkwardness with handling multiple files is to create a "tarball" of the files and the GZip the TAR. Abbrevia's GZip and TAR handling functions also work with such files transparently.

---

## Cabinet File Compression Techniques

A Windows cabinet is a collection of compressed files that can be optimized for maximum compression. A set of cabinets can be created to hold the collection in fixed size files called a spanned cabinet set, or simply cabinet set. A cabinet can also span disks.

A folder is a contiguous block of compressed data within a cabinet or cabinet set. Files in a cabinet archive can be compressed across their file boundaries as a single folder.

Compressing files in this way can significantly improve compression ratios as opposed to compressing the files individually. However, there is a trade-off between random access time to an individual file and compression ratio since an entire folder must be decompressed to extract an item from it. Folders can span cabinets.

A file is an individual compressed item in the cabinet. Files can span folders.

The Microsoft CAB library supports three lossless compression techniques: storing, MSZIP, and LZX. Any combination of these techniques can be used within a cabinet archive. MSZIP is essentially the same as the deflation compression method used by PKZIP. LZX is an LZ77 based compression technique that uses static Huffman encoding. LZX uses more memory but can produce better compression ratio than MSZIP. Stored, of course, means the file is stored into the cabinet uncompressed.

---

## Abbrevia's Cabinet File Compression Capabilities

Abbrevia supports the three compression methods and can extract any file from a Microsoft CAB file compatible format archive. Abbrevia provides several components that make it easy to add CAB file support to your application. The Abbrevia cabinet components provide a friendly interface between your application and the Microsoft CABINET.DLL.

---

## Compression Used by Abbrevia's Compound Storage

Abbrevia uses the standard Deflate algorithm (see page 16) when adding files into a Compound Storage. Abbrevia does not directly support encryption for files contained within a compound file.

---

## Additional Reading

The following documents might be helpful in understanding data compression:

- Phil Katz, *APPNOTE.TXT*, included with PKZIP for Windows 2.5, PKWare Inc., 1996.
- Mark Nelson, *The Data Compression Book*, M&T Books, 1992.
- James A. Storer, *Data Compression: Methods and Theory*, Computer Science Press, Inc., 1988.
- J. Ziv and A. Lempel, “A universal algorithm for sequential data compression,” *IEEE Transactions on Information Theory*, Volume 23, Number 3, May 1977, pages 337-343.
- J. Ziv and A. Lempel, “Compression of individual sequences via variable-rate coding,” *IEEE Transactions on Information Theory*, Volume 24, Number 5, September 1978, pages 530-536.
- Max Maischein, “TAR Format”, 1996.
- “Microsoft Cabinet File Format,” Microsoft Cabinet SDK, 1997.
- “Microsoft MSZIP Data Compression Format,” Microsoft Cabinet SDK, 1997.
- “Microsoft LZX Data Compression Format,” Microsoft Cabinet SDK, 1997.
- P. Deutsch, “Deflate Compressed Data Format Specification,” RFC 1951.
- P. Deutsch, “ZLib Compressed Data Format Specification,” RFC 1950.
- P. Deutsch, “GZip file format specification version 4.3,” RFC 1952.

# Chapter 3: Tutorials and Tours

In this chapter you will find tours and tutorials that will help you explore Abbrevia's capabilities. The following topics are covered:

- A tutorial that demonstrates the use of the TAbZipBrowser component. The application allows you to search for a file in a specified directory—even files in zip archives or self-extracting archives.
- A tutorial that shows how to create an application that allows you display the contents of a cabinet archive using the TAbCabBrowser component.
- A tutorial that demonstrates how to use the AddFromStream and ExtractToStream methods of the TAbZipKit to add the lines of a TMemo directly to a zip archive, and extract an item from a zip archive directly into the TMemo.
- A tutorial that demonstrates the use of the TAbMakeSelfExe component to convert a zip archive to a self-extracting zip archive.
- A tour of a full-featured application that uses the TAbZipOutline component. This section describes the use of the application and also highlights Abbrevia features that you might want to use in your applications.
- A tour of a full-featured application that uses the TAbZipView component. This section describes the use of the application and also highlights Abbrevia features that you might want to use in your applications.

To run any of the tutorials, you must have successfully installed Abbrevia and performed the component installation, as described in “Installation” on page 4.

---

## Tutorial 1: TAbZipBrowser - Searching for files

This tutorial shows how to create an application that searches for a specified file, even if the file is in a PKZIP-compatible archive. It first searches the current directory for the file. It then uses the TAbZipBrowser to scan the directory information of any files in the directory with a ZIP or EXE extension, assuming that they are PKZIP-compatible archives. If the file is found, its path is displayed.

1. Create a new application.
2. Modify the form's properties. Set the form's BorderStyle to bsDialog, Caption to "File Finder", Height to 480, and Width to 444.
3. Drop a TEdit on the form. Set Left to 268, Top to 8, Text to "(blank)", and Width to 165.
4. Drop a TLabel on the form. Set Caption to "Search for &File:", FocusControl to Edit1, Left to 192, and Top to 16.
5. Drop a TButton on the form. Set Enabled to False, Kind to bkOK, Caption to "&Search", Left to 312, and Top to 40.
6. Drop another TButton on the form. Set Enabled to False, Kind to bkAbort, Caption to "&Cancel", Left to 312, and Top to 72.
7. Drop a TMemo on the form. Clear the TMemo's contents by double-clicking on the Lines property in the Object Inspector to launch the Property Editor. Press Ctrl-A to select all of the text, and then Delete. Click OK to close the property editor. Set Height to 336, Left to 192, Top to 112, and Width to 240.
8. Drop a TDriveComboBox on the form. Set Left to 8, Top to 8, and Width to 169 (Note: this component doesn't exist on Linux).
9. Drop a TDirectoryListBox (use a TAbDirectoryListBox on Linux) on the form. Set Height to 416, Left to 8, Top to 32, and Width to 169.

10. Drop a TFileListBox (use a TAbFileListBox on Linux) on the form. Set Visible to False. Your form should look something like that shown in Figure 3.1.

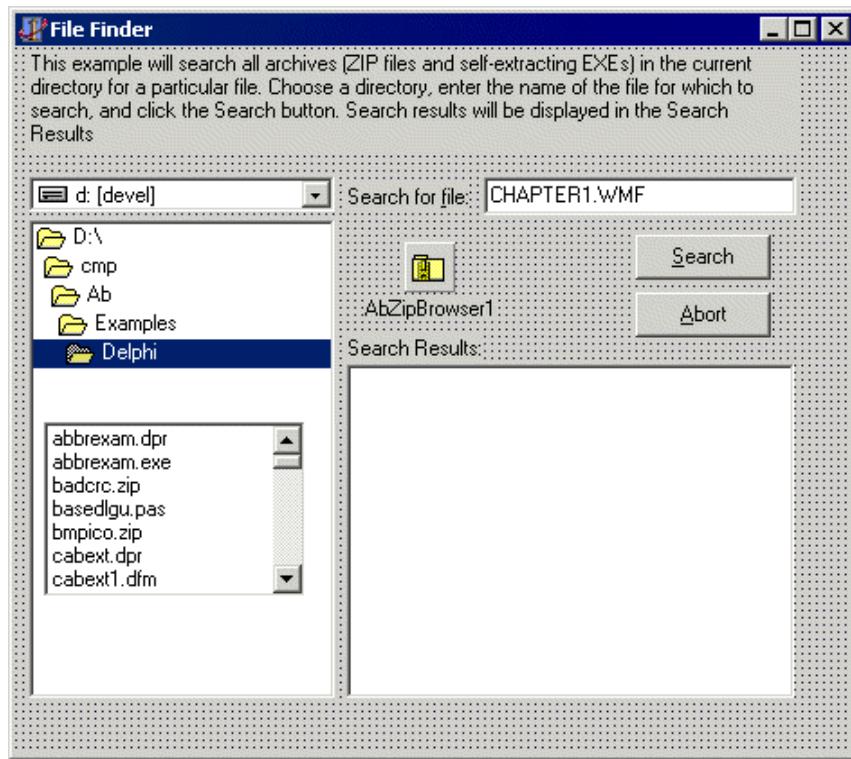


Figure 3.1: The File Finder Search main form

11. Connect the drive, directory and file list components by changing the TDIRECTORYLISTBOX FileList property to FileListBox1 and the TDRIVECOMBOBOX DirList property to DirectoryListBox1.
12. Add an Aborted variable to the private section of the form's declaration:

```
 TForm1 = class(TForm)
  ...
private
  Aborted : Boolean;
...
end;
```

13. Set the Aborted variable when the user clicks the Abort button. Double-click the Abort button to add the following event handler:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Aborted := True;
end;
```

14. Selectively enable the Search button by creating an OnChange event handler for the TEdit. Double-click the TEdit to add the following event handler:

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  Button1.Enabled := Length(Edit1.Text) > 0;
end;
```

15. Drop a TAbZipBrowser on the form.

16. Create an event handler for the Search button. Double-click the Search button to add the following event handler:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I, j : Integer;
  CurFile : string;
begin
  Button1.Enabled := False;
  Memo1.Clear;
  try
    Button2.Enabled := True;
    Aborted := False;
    {look in the file list box for the file}
    for i := 0 to pred(FileListBox1.Items.Count) do begin
      Application.ProcessMessages;
      if Aborted then
        break;
      if CompareText(Edit1.Text, FileListBox1.Items[i]) = 0
        then begin
          Memo1.Lines.Add('Found in ' + FileListBox1.Directory);
          break;
        end;
    end;
  except
  end;
```

```
{now add search of zip and self extracting files}
CurFile :=UpperCase(FileListBox1.Items[i]);
if (Pos('.ZIP', CurFile) > 0) or
   (Pos('.EXE', CurFile) > 0) then begin
try
  AbZipBrowser1.FileName := FileListBox1.Items[i];
except
end;
if AbZipBrowser1.Count > 0 then begin
  for j := 0 to pred(AbZipBrowser1.Count) do begin
    if Aborted then
      break;
    if AbZipBrowser1[j].MatchesStoredName(Edit1.Text)
      then begin
        Memo1.Lines.Add('Found in ' +
                      FileListBox1.Items[i]);
        break;
      end;
    end;
  end;
finally
  Memo1.Lines.Add('Done!');
  Edit1.Enabled := True;
  Button1.Enabled := True;
  Button2.Enabled := False;
end;
end;
```

This code implements the actions necessary when the Search button is clicked. First, the Abort button is enabled and the Aborted variable is reset. The current directory is searched, using the contents of the file list box. Next, each “.ZIP” or “.EXE” file is opened by setting the TAbZipBrowser FileName property to the name of the file (note the try/except block around setting the file name) (Note: Searching executable files are not supported on Linux). When the TAbZipBrowser FileName property is changed, it immediately attempts to load the file as if it were a PKZIP compatible file. If the file format is not correct, an exception is raised. If the archive is successfully loaded and it contains files, the Count property will be greater than 0. Each file name found in the archive is then compared with the search target using the MatchesStoredName method of TAbArchiveItem (see “TAbArchiveItem Class” on page 204). If the file is found, the TMemo is updated with the location of the file.

17. Compile and run the program. Use the directory list box to navigate to the desired directory, then type the file name in the “Search for file” edit control. Click on the Search button. If the file is found, its path is displayed in the file list box. Figure 3.2 shows the File Finder Search Results window.

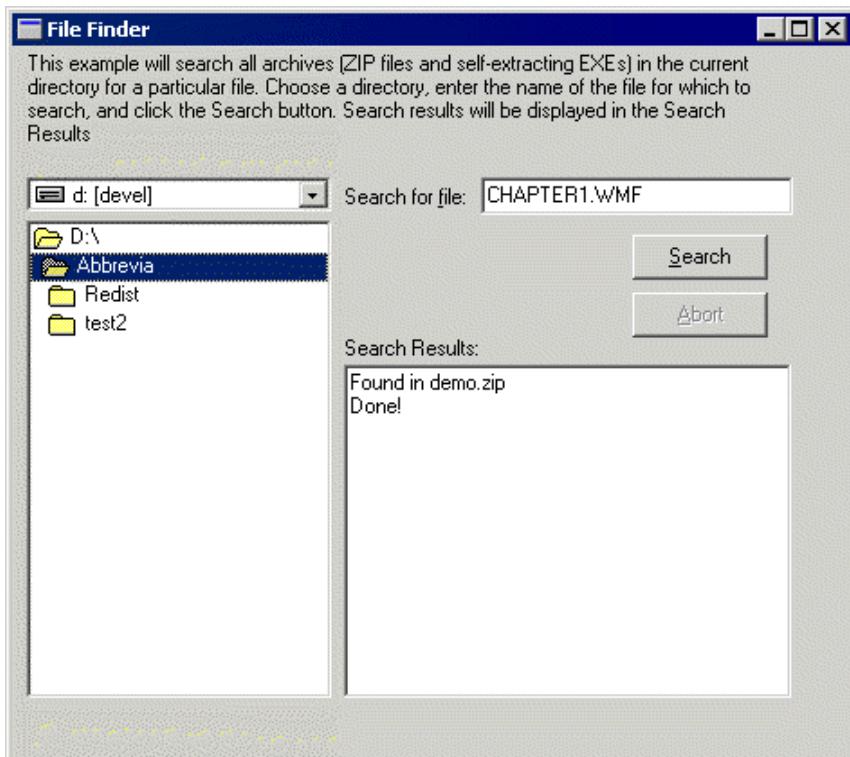


Figure 3.2: The File Finder program in action

In this case, the file was found in the D:\Abbrevia directory inside the demo.zip file. This tutorial is in the Finder project in the Abbrevia examples directory.

## Tutorial 2: TAbCabBrowser: Contents of a Cabinet

This tutorial shows how to use the TAbCabBrowser to view the contents of a cabinet file. Figure 3.3 shows the end result of this tutorial.

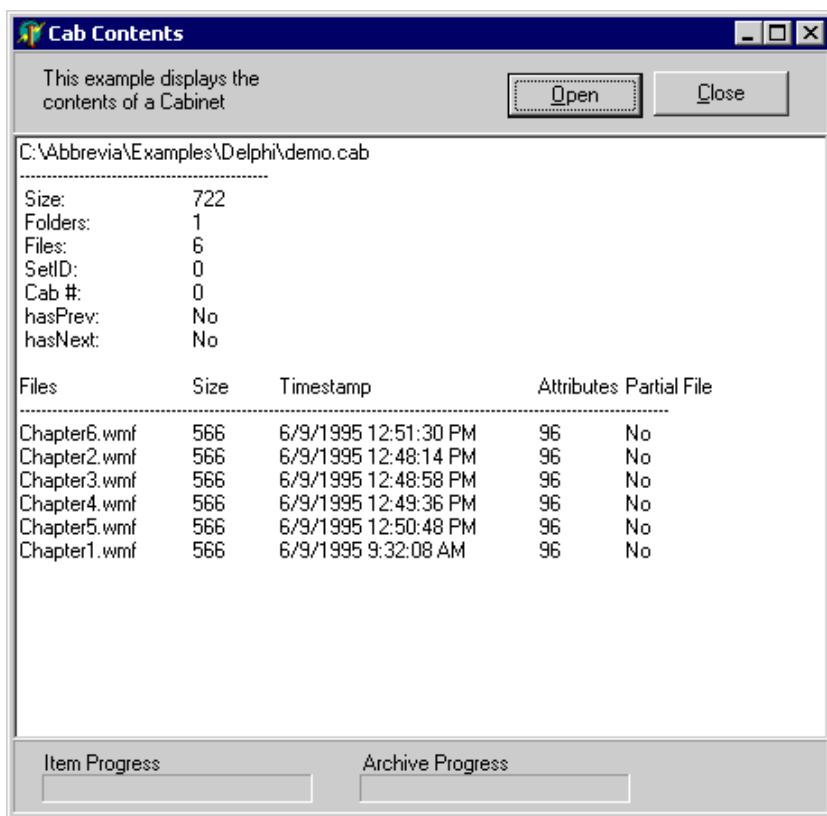


Figure 3.3: The Cab Contents program running

1. Create a new application in.
2. Change the form's Caption. Set Caption to "Cab Contents." Set the form's Height to 450 and the Width to 460.
3. Drop a TPanel on the form. Set Align to alTop and Height to 46.
4. Drop a TPanel on the form. Set Align to alBottom and Height to 41.

5. Drop a TButton on the top panel. Set the Caption to “&Open”. Drop another TButton on the top panel and set the Caption to “&Close.”
6. Drop 2 TABMeter components on the bottom panel.
7. Drop a TABCabBrowser on the form. Click on the ArchiveProgressMeter property and assign one of the meters to it. Click on the ItemProgressMeter property and assign the other meter.
8. Drop a TOpenDialog on the form.
9. Drop a TMemo on the form. Set Align to alClient.
10. Create the following event handler for the Open button's OnClick event:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    AbCabBrowser1.OpenArchive(OpenDialog1.FileName);
end;
```

11. Create the following event handler for the Close button's OnClick event:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  AbCabBrowser1.CloseArchive;
  Memo1.Clear;
end;
```

12. Create the following event handler for the AbCabBrowser1.OnLoad event:

```
procedure TForm1.AbCabBrowser1Load(Sender: TObject);
const
  BoolToStr : array[Boolean] of string = ('No', 'Yes');
var
  i : Integer;
  LI : Longint;
  DT : TDateTime;
  s : string;
begin
  Memo1.Clear;
  with AbCabBrowser1 do begin
    Memo1.Lines.Add(Filename);
    Memo1.Lines.Add('-----');
    IntToStr(Items[i].ExternalFileAttributes) + #9 +
      BoolToStr[Items[i].PartialFile];
    Memo1.Lines.Add(s);
  end;
end;
```

13. Compile and run the program. Click the Open button and select a CAB file. The contents of the CAB file are displayed in the memo.

This tutorial is in the Excbrows project in the Abbrevia examples directory.

## Tutorial 3: TAbZipKit: Stream Compression

This tutorial shows how to create an application that compresses the lines of a TMemo directly to a zip archive and extracts a zip archive item directly to a TMemo (see Figure 3.4).

3

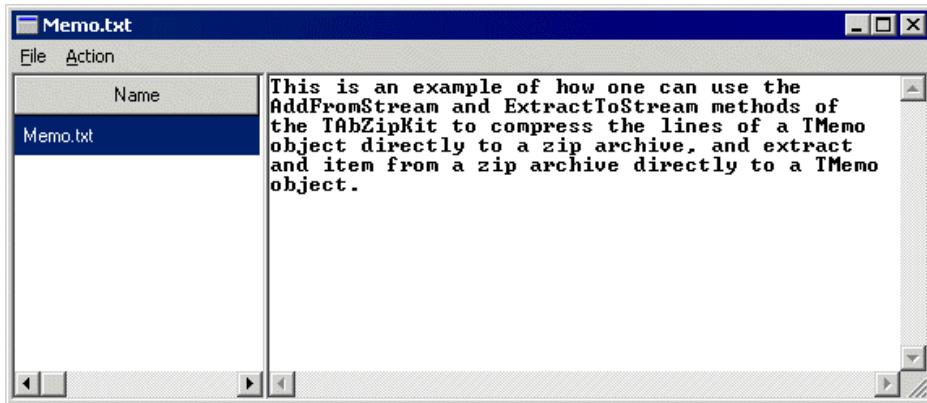


Figure 3.4: The Stream Compression example

1. Create a new application.
2. Change the form's Caption. Set Caption to "Compressed Memo." Set the form's Height to 190 and the Width to 456.
3. Drop a TAbZipKit on the form.
4. Drop a TAbZipView on the form. Set Align to alLeft and Width to 125. Double-click the ZipComponent property to assign AbZipKit1 to it.
5. Drop a TOpenDialog on the form.
6. Drop a TMemo on the form. Set Align to alClient. Clear the TMemo's contents.
7. Drop a TMainMenu on the form. Double-click it and add the following menu items:

```
File1: Caption = '&File'  
  Open1 : Caption = '&Open'  
  Close1: Caption = '&Close'  
  
Action1: Caption = '&Action'  
  Add1 : Caption = '&Add from memo'  
  Clear1 : Caption = '&Clear memo'  
  Extract1 : Caption = '&Extract to memo'
```

8. Create the following event handler for the menu's Open item:

```
procedure TForm1.Open1Click(Sender: TObject);
begin
  OpenDialog1.Filename := '*.zip';
  if OpenDialog1.Execute then
    AbZipKit1.OpenArchive(Dialog1.Filename);
end;
```

9. Create the following event handler for the menu's Close item

```
procedure TForm1.Close1Click(Sender: TObject);
begin
  AbZipKit1.CloseArchive;
end;
```

10. Create the following event handler for the menu's Add item:

```
procedure TForm1.Add1Click(Sender: TObject);
var
  FromStream : TMemoryStream;
  FN : string;
begin
  FromStream := TMemoryStream.Create;
  try
    Memo1.Lines.SaveToStream(FromStream);
    if InputQuery('Streams', 'Give it a filename', FN) then
      begin
        AbZipKit1.AddFromStream(FN, FromStream);
      end;
  finally
    FromStream.Free;
  end;
end;
```

11. Create the following event handler for the menu's Extract item:

```
procedure TForm1.Extract1Click(Sender: TObject);
var
  ToStream : TMemoryStream;
  Item : TAbArchiveItem;
begin
  Memo1.Clear;
  ToStream := TMemoryStream.Create;
  try
    Item := AbZipView1.Items[AbZipView1.ActiveRow];
    AbZipKit1.ExtractToStream(Item.FileName, ToStream);
    ToStream.Position := 0;
    Memo1.Lines.LoadFromStream(ToStream);
  finally
    ToStream.Free;
  end;
end;
```

12. Create the following event handler for the menu's Clear item:

```
procedure TForm1.Clear1Click(Sender: TObject);
begin
  Memo1.Clear;
end;
```

13. Compile and run the application. From the File menu, open a new or existing zip archive. Type a few lines of text in the memo. From the Action menu add the memo lines to the archive and provide a file name, e.g. "TEST.TXT". You should see the file appear as an item in the TAbZipView.

14. Now select the item, "TEST.TXT", in the viewer. From the Action menu, clear the memo and extract the item "TEST.TXT" back into the memo.

This tutorial is in the Streams project in the examples directory.

## Tutorial 4: TAbMakeSelfExe - Creating a Self-Extracting Zip Archive

This tutorial shows how to create a self-extracting zip archive from an existing zip file (see Figure 3.5).

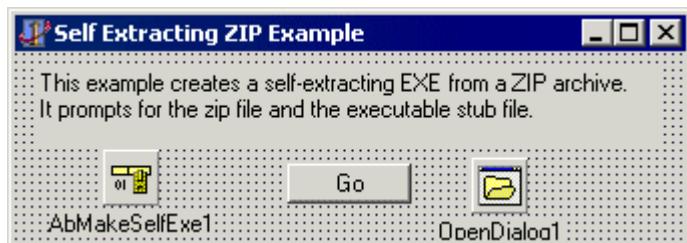


Figure 3.5: The Self Extracting ZIP example's main form

1. Create an executable stub file by simply opening either of the example projects, `Selfstub` or `Selfstbv` located in the examples directory and compiling them. The executable stub file is that part of the self-extracting archive that performs the decompression. Close the project.
2. Create a new application. Change the form's Caption "Self Extracting Zip Example". Set the form's Height to 120 and the Width to 343.
3. Drop a `TOpenDialog` on the form.
4. Drop a `TButton` on the form and create the following event handler for the `OnClickEvent`:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if AbMakeSelfExe1.Execute then
    ShowMessage(AbMakeSelfExe1.SelfExe + ' has been created');
end;
```

5. Drop a TAbMakeSelfExe on the form and create the following event handlers for the OnGetStubExe and OnGetZipFile events:

```
procedure TForm1.AbMakeSelfExe1GetStubExe(Sender: TObject;
  var afilename: String; var Abort: Boolean);
begin
  OpenDialog1.Title := 'Select executable stub';
  OpenDialog1.Filename := '';
  OpenDialog1.Filter := 'Exe files|*.exe';
  Abort := not OpenDialog1.Execute;
  if not Abort then
    aFileName := OpenDialog1.Filename;
end;

procedure TForm1.AbMakeSelfExe1GetZipFile(Sender: TObject;
  var afilename: String; var Abort: Boolean);
begin
  OpenDialog1.Title := 'Select Zip File';
  OpenDialog1.Filename := '';
  OpenDialog1.Filter := 'Zip files|*.zip';
  Abort := not OpenDialog1.Execute;
  if not Abort then
    aFileName := OpenDialog1.Filename;
end;
```

6. Compile and run the program. This example demonstrates how to create a self-extracting executable file from a zip archive.

This tutorial is in the Makesfx project in the examples directory.

## Tour 1: TPZip

The TurboPower Zip demonstration program (TPZip) is a comprehensive compression/decompression application that uses many of the capabilities of Abbrevia. You can use TPZip to view existing archives, extract files from archives, or create new archives.

When you run the TPZip program, the main window is displayed, as shown in Figure 3.6.

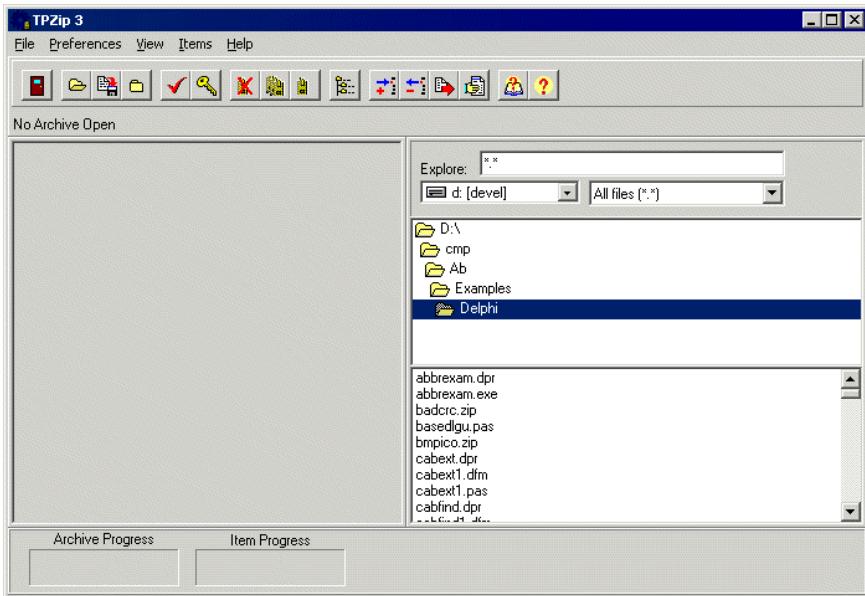


Figure 3.6: The TPZip program main window

The right half of the window is a file browser. You can use it to navigate to any directory and then click on a ZIP file in the lower list box and drag it to the large panel on the left. The contents of the ZIP file are displayed in an outline on the left side of the window.

You can also open a ZIP file by double-clicking on it or by using the File|Open menu option or the Open button on the toolbar. If you use the menu option or the toolbar button, the Open a Zip Archive dialog box is displayed so you can navigate to the desired file.

If you open the Demo.Zip file from the examples directory, the main window should then look something like that in Figure 3.7.

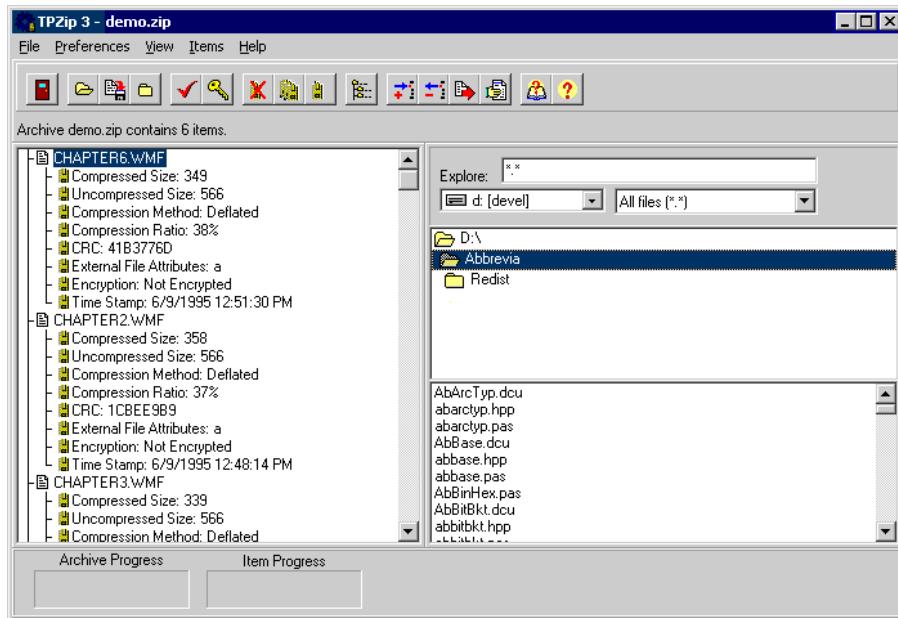


Figure 3.7: The Demo.Zip window

For each item in the archive, the outline displays its name, path (if path information was saved in the archive), and information about the file. You can choose the information that is displayed by using the View|Attributes menu item. You can also change the look of the outline by using the View|Style menu item. See “View Menu” on page 45 for more information.

TPZip uses two TAbMeters at the bottom of the window to display the progress of an archive action. The meter on the left is attached to the ArchiveProgressMeter property, so it shows the progress for the entire archive. The meter on the right is attached to the ItemProgressMeter property, so it shows the progress for the current archive item (file). The lower right panel is updated using the OnArchiveItemProgress event.

## Extracting files

It is easy to extract a file from an archive. In the browser, navigate to the desired directory for extraction. Click and drag any file in the outline to the list box at the bottom of the file browser and the file is extracted into that directory.

Another way to extract a file is to right-click on it in the outline. A popup menu is displayed and you can select Extract. The Extract Selected File dialog box is displayed as shown in Figure 3.8.



Figure 3.8: The Extract Selected File dialog box

To extract the file to a specific directory, simply navigate to the directory and click OK. If the directory does not exist, navigate to its parent directory, use the Create Directory button to create a new directory, navigate to the new directory, and then click OK. The file is extracted.

You can also use the path information stored in the archive to choose the directory. The edit control at the top of the dialog box shows the path information that is stored in the archive for the selected file. If the "Restore Path" check box is not checked (the default), the file is extracted to the target directory and the path information stored in the archive is ignored. In the example shown above, the file would be extracted to C:\TEMP\CHAPTER1.WMF. If you check the "Restore Path" check box, the file is extracted to the directory indicated by the path information stored in the archive, relative to the target directory. In this example, if

Restore Path were checked, the file would be extracted to D:\ABBREVIA\REDIST, which is probably not exactly what you would like. To avoid that, you could navigate back up the directory path to D:\, so that it becomes the Target Directory.

The “Create Directories” check box is used only if “Restore Path” is checked. If “Create Directories” is checked and the directory indicated in the path information stored in the archive does not exist, it is created. If “Create Directories” is not checked and the directory indicated in the path information stored in the archive does not exist, the file is not extracted.

To extract all of the files (or multiple files) from the archive, use the Items|Extract Files menu option or the Extract Files button on the toolbar. The Extract Files With FileMask dialog box is displayed in Figure 3.9.

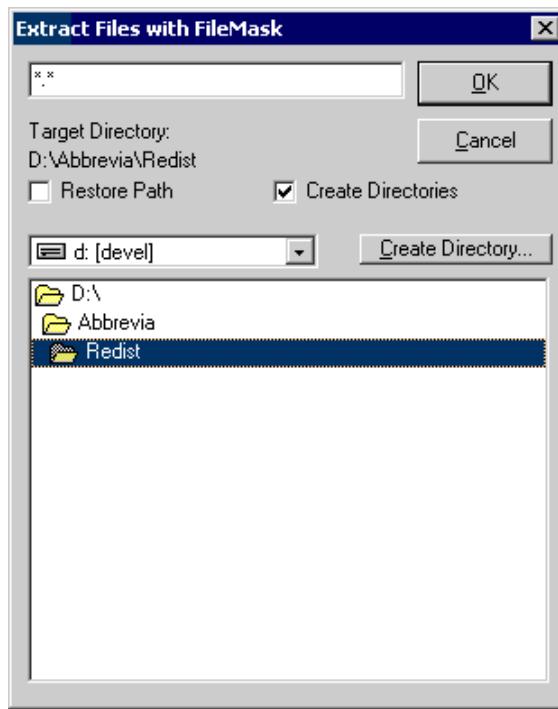


Figure 3.9: The Extract Files with FileMask dialog box

The default file mask is “\*.\*”, which means that all files will be extracted from the archive. You can modify the file mask to extract any desired group of files. To extract the files to a specific directory, simply navigate to the directory and click OK. If the directory does not exist, navigate to its parent directory, use the Create Directory button to create a new directory, navigate to the new directory, and then click OK. The files are extracted.

If the “Restore Path” check box is not checked (the default), the files are extracted to the target directory and the path information stored in the archive is ignored. In the example shown above, the files would be extracted to E:\ABBREVIA\DOC. If the “Restore Path” check box is checked and path information is stored in the archive for a file, the file is extracted to the directory indicated by the path information, relative to the target directory.

The setting of Create Directories is used only if Restore Path is checked. If Create Directories is checked and the directory indicated in the path information stored in the archive does not exist, it is created. If “Create Directories” is not checked and the directory indicated in the path information stored in the archive does not exist, the file is not extracted.

## Creating an archive

To create an archive, click on the Open button on the toolbar to display the Open a Zip Archive dialog box. Navigate to the Abbrevia examples folder, create a new archive called MYTEST.ZIP, and click Open. A message tells you that the archive does not exist and asks if you would like to create it. Click Yes. Now you can add files to your new archive. Use the file browser to find a file and drag it to the outline. The file is added to the archive. You can add as many files as desired to the archive (up to 65,535).

If you add a file to the archive by mistake and need to delete it, right-click on the file and choose Delete from the menu.

## The TPZip menu

### File menu

From the File menu, you can open, save, or close an archive, convert it to a self-extracting archive, or exit the application.

#### Open

You can open an existing PKZIP-compatible archive or a self-extracting archive (the Abbrevia components can operate on either type of archive transparently). If you select an existing archive, the contents of the archive are displayed in the outline on the left. If you specify an archive that does not exist, a new archive is created.

#### Save

The Save menu option updates the archive.

TPZip updates the archive when this menu option is used or when the archive is closed (for example, when a different archive is opened). This is the behavior when the TAbZipOutline AutoSave property is False. See the entry for the AutoSave property on page 141 for more information.

## **Close**

The Close option closes the current archive.

## **Convert (to self-extracting)**

The Convert option converts an archive to a self-extracting archive by attaching it to the end of an executable (called a stub) that is designed specifically for this purpose. Sample stubs (including the one that is used when you choose this option) are provided with Abbrevia, or you can build your own. See “TAbMakeSelfExe Component” on page 293 for more information.

## **Exit**

This option terminates TPZip.

## **Preferences Menu**

You can modify the behavior of TPZip by using the Preferences menu.

### **Confirmations**

TPZip allows you to enable confirmations of common operations. These can be useful for drag and drop and wild card operations.

### **Compression method to use**

The choices for compression method are Store, Deflate, and Best. If you choose Best (the default), TPZip attempts to use Abbrevia’s most effective compression technique (the deflate method). However, that method is abandoned if the resulting file would be smaller if it were simply stored. If you select Store or Deflate, TPZip uses the selected technique, regardless of the resulting file size.

### **Deflation options**

The deflation options allow you to select varying trade-offs between compression and speed. The choices are Maximum (most compression, slowest speed), Normal, Fast, and Super Fast (least compression, fastest speed). Normal deflation is the default.

### **Extract Options**

The Extract Options control what happens as files are extracted from an archive. The Create Directories option controls whether Abbrevia is allowed to create directories as needed to extract the files. The Restore Path option controls whether relative file path information is restored.

### **Password**

Abbrevia encryption and decryption is fully compatible with PKZIP. Use the Password option to supply a password for extracting an encrypted file, or to encrypt a file when it is added or freshened. Encryption is automatic—if the Password is set, added files are encrypted. When a Password is entered, a key icon is displayed on the status bar at the upper

right of the main window. If you try to extract an encrypted file from an archive and don't supply a Password (or supply an invalid one), TPZip prompts you for a Password using the OnNeedPassword event. This event is repeated based on the value of the PasswordRetries property. See "Encryption" on page 17 for more information on passwords.

### Store Options

The Store Options control what happens when files are added to an archive. The Recurse Tree option controls whether the directory tree is recursed during add operations. The Remove Dots option controls whether relative path information is removed—adding "..\README.TXT" to an archive would result in a stored file name of "README.TXT" if the RemoveDots options is checked. The Strip Path option controls whether path information is retained in the stored file.

## View Menu

### Attributes

Along with compressed data, PKZIP archives store a wealth of information about each file in the archive. The Attributes option allows you to select which attributes are displayed. For a description of each of the attributes, see the Attributes property on page 140.

### Font

The Fonts option invokes the Font dialog box, which allows you to select the font for the components on the TPZip main form.

### Style

The Style option controls the look of the outline.

If "Hierarchy" is checked (the default), the archive contents are displayed as if the archive is a virtual drive (files contained within folders). If "Hierarchy" is not checked, each item's file name is displayed with the path information included.

If "Only Text" is checked, no pictures or glyphs are displayed.

If "+/-" is checked, a + or - is displayed to the left of each item in the hierarchy. If the item is collapsed, a + is displayed. If the item is expanded, a - is displayed. You can toggle between collapsing and expanding the item by double-clicking on the + or -.

If "Glyphs" is checked, glyphs are displayed to the left of each item in the outline. A folder glyph is displayed for directories, a page glyph is displayed for items in the archive, and a zipper glyph is displayed for attributes.

If "+/- and Glyphs" is checked, both +/- boxes and glyphs are displayed.

If "Only Tree" is checked, lines indicating the tree structure are displayed.

If "Tree and Glyphs" is checked, both the tree structure and glyphs are displayed.

## Items Menu

Using the Items menu, you can Add, Delete, Extract, or Freshen files that match a specified FileMask.

### Add Files

The Add Files option adds files to the current archive.

### Delete Files

The Delete Files option deletes files from the current archive.

### Extract Files

The Extract Files option extracts files from the current archive.

### Freshen Files

The Freshen Files option freshens files in the current archive.

## Help Menu

### Contents

View the TPZip help file.

### About

Display information about TPZip, Abbrevia, and TurboPower.

## Popup menu

If an archive is loaded, you can right-click on any archive item to display a popup menu. If you right-click on a directory or a file attribute (such as Compressed Size), the popup menu is not displayed. From the popup menu, you can delete, extract, freshen, move (i.e., change the archive item's stored file name), or run the file (i.e., extract the file to a temporary directory and launch its associated application).

## Tour 2: ZipView

ZipView is an involved demonstration application built around the TAbZipView and TAbZipKit components. You can use ZipView to view an archive and add, delete, extract, freshen, or move items in the archive. Table 3.10 shows the ZipView running.

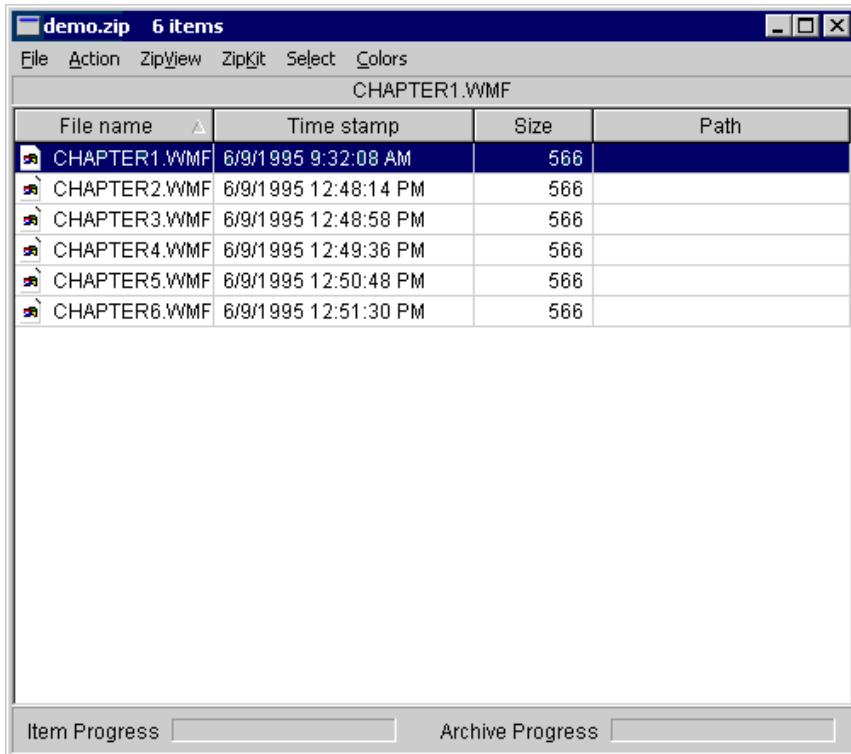


Figure 3.10: The ZipView program running

The main portion of the window is a TAbZipView component. The zip view is attached to a TAbZipKit component to display the contents of a zip file.

You can open a ZIP file by using the File|Open menu option on the toolbar. The Open a Zip Archive dialog box is displayed so you can navigate to the desired file.

For each item in the archive, the viewer displays its name, time stamp, uncompressed size, and path (if path information was saved in the archive). Other information can be selected by choosing the desired attribute using the ZipView|Attributes menu item. You can change

the look of the viewer by using the ZipView|Display menu item. You can choose which attributes to sort on via the ZipView|Sort menu item. See “ZipView Menu” on page 48 for more information.

ZipView uses two TAbMeters at the bottom of the window to display the progress of an archive action. The meter on the left is attached to the ArchiveProgressMeter property, so it shows the progress for the entire archive. The meter on the right is attached to the ItemProgressMeter property, so it shows the progress for the current archive item (file). The lower right panel is updated using the OnArchiveItemProgress event.

## Extracting files

To extract a file from an archive, select any of the items in the viewer you wish to extract and then choose the Action|Extract Selected Items menu item. You will be prompted to choose a destination directory for the extracted file.

Another way to extract a file is to right-click on it in the viewer. A popup menu is displayed and you can select Extract.

## Creating a new archive or opening an existing archive

To create or open an archive, select the File|Open menu item to display the Open a Zip Archive dialog box. Enter the name of the zip archive you wish to create or selected an existing archive, e.g. “TEST.ZIP”.

## Adding files

To add a file to an open archive, use the Action|Add menu item to display a file selection dialog box. Select the file and press OK.

## The ZipView menu

### File Menu

From the File menu, you can open, save, or close an archive, or exit the application.

#### Open

You can open an existing PKZIP-compatible archive or a self-extracting archive (the Abbrevia components can operate on either type of archive transparently). If you select an existing archive, the contents of the archive are displayed in the viewer on the left. If you specify an archive that does not exist, a new archive is created.

## Save

The Save menu option updates the archive.

ZipView updates the archive when this menu option is used or when the archive is closed (for example, when a different archive is opened). This is the behavior when the TAbZipKit.AutoSave property is False.

## Close

The Close option closes the current archive.

## Exit

This option terminates ZipView.

## Action Menu

Using the Action menu, you can add, delete, extract, freshen, move, or test items in the archive.

### Add

This option adds files to the current archive.

### Delete selected items

This option deletes all the items selected in the viewer from the archive. Deleted items will be displayed with the colors selected from the Colors menu until the archive is saved or closed.

### Extract selected items

This option extracts all items selected in the viewer.

### Freshen selected items

This option freshens all items selected in the viewer.

### Move selected item

This option moves the item selected in the viewer.

### Test selected items

This option tests all the items selected in the viewer.

## ZipView Menu

### Attributes

Along with compressed data, PKZIP archives store a wealth of information about each file in the archive. The Attributes option allows you to select which attributes are displayed.

### Display

The Display options control the look of the viewer.

If “Alternate colors” is checked, then every other row in the viewer is displayed using an alternate color. The alternate colors can be configured using the Colors menu item.

If “Column lines” is checked, then a vertical line is displayed between the attribute columns.

If “Column moving” is checked, then the columns can be reordered by dragging the column’s header button.

If “Column resizing” is checked, then the columns can be resized by dragging the left or right edge of the column’s header button.

If “Multiselect” is checked, then multiple items in the viewer can be selected.

If “Row lines” is checked, then a horizontal line is displayed between the rows.

If “Show icons” is checked, then the default icon of the item is displayed to the left of the item name.

If “ThumbTracking” is checked, then the viewer window moves along with the scroll bar.

If “Track active row” is checked, then when the rows are sorted, the viewer window is adjusted to keep the active row in view.

### Font

The Fonts option invokes the Font dialog box, which allows you to select the font for the viewer.

### Row height

The Row height option allows you to configure the height of the header row and the height of the items rows.

### Sort

The Sort option allows you to choose which attribute to allow sorting on. If sorting for a particular attribute is enabled, then the items are sorted by clicking on the attribute column’s header button. The sort order is reversed each time the header button is clicked, and a small arrow is displayed on the button to indicate the sort order.

## ZipKit menu

You can modify the compression/decompression characteristics by using the ZipKit menu. This menu item enables you to select various options for the ZipKit.

### Method

The choices for compression method are Store, Deflate, and Best. If you choose Best (the default), TPZip attempts to use Abbrevia's most effective compression technique (the deflate method); however, that method is abandoned if the resulting file would be smaller if it were simply stored. If you select Store or Deflate, ZipView uses the selected technique, regardless of the resulting file size.

### Deflation

The deflation options allow you to select varying trade-offs between compression and speed. The choices are Maximum (most compression, slowest speed), Normal, Fast, and Super Fast (least compression, fastest speed). Normal deflation is the default.

### Extract

The Extract options control what happens as files are extracted from an archive. The “Create Directories” option controls whether Abbrevia is allowed to create directories as needed to extract the files. The “Restore Path” option controls whether relative file path information is restored.

### Store

The Store options control what happens when files are added to an archive. The “Recurse Tree” option controls whether the directory tree is recursed during add operations. The “Remove Dots” option controls whether relative path information is removed—adding “..\\README.TXT” to an archive would result in a stored file name of “README.TXT” if the “Remove Dots” option is checked. The “Strip Path” option controls whether path information is retained in the stored file.

## Select menu

You can select or deselect all the items in the viewer from this menu.

## Colors menu

You can choose the colors used by the viewer with this menu. See the Colors property of the TAbZipView on page 178.

## Popup menu

If an archive is loaded, you can right-click on any item in the viewer to display a popup menu. From the popup menu, you can delete, extract, freshen, or move (i.e., change the archive item's stored file name) the file.



---

# Chapter 4: Base Browser Component

This chapter provides information on the base browser class from which most of the Abbrevia non-visual components are descended.

The TAbBaseBrowser class is the ancestor to all of the Abbrevia non-visual zip and cabinet components. The TAbBrowser provides the common functionality and references an abstract TAbArchive class.

---

# TAbBaseComponent Class

The TAbBaseComponent class is the ancestor of most of the Abbrevia components. It is descended from the VCL TComponent class and adds the Version property that identifies the current version.

## Hierarchy

TComponent

TAbBaseComponent (AbBase)

## Properties

Version

## Reference Section

### Version

### property

```
property Version : string
```

↳ Returns or displays the current version number of Abbrevia.

Version is provided so you can identify your Abbrevia version if you need technical support. If you double-click on the Version property in the IDE's object inspector, the Abbrevia About box is displayed.

Although the Version property can be written to, all changes are ignored.

# TAbBaseBrowser Class

The TAbBaseBrowser class is the ancestor to all of the Abbrevia non-visual components, both the zip archive components, and the cabinet archive components. The TAbBrowser provides the common functionality and references an abstract TAbArchive class.

The TAbBaseBrowser encapsulates an abstract archive.

## Hierarchy

TComponent

① TAbBaseComponent (AbBase) .....	54
TAbBaseBrowser (AbBrowse)	

## Properties

ArchiveProgressMeter	FileName	SpanningThreshold
BaseDirectory	ItemProgressMeter	Status
CompressionType	LogFile	TempDirectory
Count	Logging	① Version

## Methods

ClearTags	FindItem	UnTagItems
CloseArchive	OpenArchive	
FindFile	TagItems	

## Events

OnArchiveItemProgress	OnConfirmProcessItem	OnRequestImage
OnArchiveProgress	OnLoad	
OnChange	OnProcessItemFailure	

## Reference Section

---

<b>ArchiveProgressMeter</b>	<b>property</b>
-----------------------------	-----------------

---

`property ArchiveProgressMeter : TAbMeter`

↳ Specifies a synchronized meter for the OnArchiveProgress event.

The ArchiveProgressMeter property provides a mechanism to associate a TAbMeter (see page 290) with the OnArchiveProgress event. If a meter is specified, it will be updated to reflect the progress given by the OnArchiveProgress event. This occurs even if no event handler has been assigned to the event.

See also: OnArchiveProgress

---

<b>BaseDirectory</b>	<b>property</b>
----------------------	-----------------

---

`property BaseDirectory : string`

↳ Supplies the default path for add and extract operations on the archive.

**Note:** BaseDirectory is not necessarily the same as the directory of the archive.

When you add a file to an archive, if FileName is a relative file specification (e.g., “README.TXT” or “EXAMPLES\README.TXT”), BaseDirectory is used as the starting point to search for the file. When you extract a file from an archive, it is extracted to the current BaseDirectory or a subdirectory of the current BaseDirectory.

See the add, freshen, and extract operations in the non-visual components for information on how BaseDirectory is used in each case.

---

<b>ClearTags</b>	<b>method</b>
------------------	---------------

---

`procedure ClearTags;`

↳ Untags all items in the archive.

See also: TagItems, UnTagItems

---

<b>CloseArchive</b>	<b>method</b>
---------------------	---------------

---

`procedure CloseArchive;`

↳ Closes the archive.

If necessary, the current archive will be saved. The FileName property will be set to a null string.

See also: FileName

---

**CompressionType** property

```
property CompressionType: TAbCompressionTypes
TAbCompressionTypes = (
  ctZip, ctGZip, ctTar, ctGZipTar, ctZLib, ctCab);
```

Default: ctZip

↳ Specifies the compression type of the archive.

When opening an archive CompressionType is set to the type of archive as determined by examining the archive's file extension.

When creating an archive, the compression type must be set to the type of archive desired.

---

**Count** run-time, read-only property

```
property Count : Integer
```

↳ Returns the number of items in the current archive.

Count is the number of items available from the Items property.

● **Caution:** Count returns the number of items in the memory representation of the archive. For components that can modify an archive (e.g., TAbZipper, if AutoSave in TAbZipper and TAbZipOutline is False), and items have been added or deleted from the archive, the number of items in the memory representation can be different than the number of items in the disk file or stream.

See also: Items, TAbZipper.AutoSave

---

**FileName** property

```
property FileName : string
```

↳ Returns the disk file name of the archive.

FileName is the full path name of the archive. If you change FileName, the current archive is saved, if necessary, and closed. If FileName is not blank, a new archive is opened and loaded, initializing the Count and Items properties.

See also: CloseArchive, OpenArchive

**FindFile****method**


---

```
function FindFile(const aFileName : string) : Integer;
```

↳ Returns the index of the specified file in the archive's item list.

FindFile searches through the list of archive files until it finds one with an exact match for the specified file name. It then returns the index of that item. FindFile returns -1 if no match is found. No two items in an archive can have the same file name.

See also: FindItem

**FindItem****method**


---

```
function FindItem(aItem : TAbArchiveItem) : Integer;
```

↳ Returns the index of the specified archive item.

FindItem searches through the list of archive items until it finds the one specified. It then returns the index of that item. FindItem returns -1 if no match is found.

See also: FindFile

**ItemProgressMeter****property**


---

```
property ItemProgressMeter : TAbMeter
```

↳ Specifies a synchronized meter for the OnArchiveItemProgressEvent.

The ItemProgressMeter property provides a mechanism to associate a TAbMeter (see page 290) with the OnArchiveItemProgress event. If a meter is specified then it will be updated to reflect the progress given by the OnArchiveItemProgress event. This occurs even if no event handler has been assigned to the event.

See also: OnArchiveItemProgress

**LogFile****property**


---

```
property LogFile : string
```

↳ Specifies the text file to use for logging.

LogFile specifies the text file that will receive the log entries during archiving operations if logging has been enabled. If the file does not exist, it will be created. If the file does exist, the entries will be appended to the end of the file.

See also: Logging

**Logging****property**

```
property Logging : Boolean
```

Default: False

- ↳ Controls whether archive operations are recorded.

When Logging is True, an entry is logged to a text file specified by the LogFile property, for each operation (add, delete, extract, freshen, move, replace) performed on an archive. The following example shows the format of log entries:

```
D:\Test\Test.zip logging 04/27/1999 6:12:06 PM
FDI.H deleted 04/27/1999 6:12:37 PM
Ffdefine.inc added 04/27/1999 6:12:52 PM
```

If LogFile is not set, then logging will not be attempted regardless of the value of Logging.

See also: LogFile

**OnArchiveItemProgress****event**

```
property OnArchiveItemProgress : TAbArchiveItemProgressEvent
TAbArchiveItemProgressEvent = procedure(
  Sender : TObject; Item : TAbArchiveItem; Progress : Byte;
  var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called during long operations on a single archive item.

OnArchiveItemProgress is called by add, extract, and freshen operations on an archive item. You can use it to display the progress of the operation, or even to abort the operation. For example, you might want to display the name of the item being processed, the action being performed on the item, and an indication of the progress.

Progress is a percentage that indicates how far the operation has progressed through the archive item. Valid values for Progress are 0 to 100. If you set Abort to True in the event handler, the action for that item is aborted.

See also: OnArchiveProgress, OnProcessItemFailure

## OnArchiveProgress

event

```
property OnArchiveProgress : TABArchiveItemProgressEvent  
  TABArchiveProgressEvent = procedure(Sender : TObject;  
    Progress : Byte; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called once per item when a process steps through the archive.

OnArchiveProgress is called during DeleteFiles, FreshenFiles, ExtractFiles, FileName (a change causes a new archive to be loaded), and Save archive operations. You can use it to display the progress of the operation, or even to abort the operation.

Progress is a percentage that indicates how far the current operation has progressed through the entire archive. Valid values for Progress are 0 to 100. If you set Abort to True in the event handler, the entire operation is aborted and an EAbUserAbort exception is raised.

See also: OnArchiveItemProgress

## OnChange

event

```
property OnChange : TNotifyEvent
```

- ↳ Defines an event handler that is called when the archive changes.

OnChange is called when the archive is opened or closed, and when the contents of the archive change (immediately after an add, delete, freshen, or move operation).

The OnChange event can be used to update any components that display the contents of the archive or information relating to the archive (e.g., the number of files in the archive).

## OnConfirmProcessItem

event

```
property OnConfirmProcessItem : TAbArchiveItemConfirmEvent  
TAbArchiveItemConfirmEvent = procedure(Sender : TObject;  
  var Item : TAbArchiveItem; ProcessType : TAbProcessType;  
  var Confirm : Boolean) of object;  
  
TAbProcessType = (  
  ptAdd, ptDelete, ptExtract, ptFreshen, ptMove);
```

4

- ↳ Defines an event handler that is called before an action is performed on each item in the archive.

This event handler is called once for each item in an add, delete, extract, freshen, or move operation. If you set Confirm to False, the process is aborted for that item. If you do not provide a handler for this event, all processing continues as if Confirm were set to True.

The OnConfirmProcessItem event handler can be used to perform any actions that are necessary when the archive changes. For example, you might want display a confirmation dialog or save information on who modified the archive.

## OnLoad

event

```
property OnLoad : TABArchiveEvent  
TABArchiveEvent = procedure(Sender : TObject) of object;
```

- ↳ Defines an event handler that is called immediately after an archive's contents are loaded.

If FileName changes (causing a new archive to be loaded), OnLoad is called just after the archive is loaded. An OnLoad event can be used to display the name of the current archive or to update a most recently used archive list.

See also: FileName

```
property OnProcessItemFailure : TABArchiveItemFailureEvent  
  TABArchiveItemFailureEvent = procedure(Sender : TObject;  
    Item : TABArchiveItem; const ProcessType : TABProcessType;  
    ErrorClass : TABErrorClass; ErrorCode : Integer ) of object;  
  
TABProcessType = (ptAdd, ptDelete, ptExtract, ptFreshen, ptMove);  
  
TABErrorClass = (ecAbbrevia, ecInOutError, ecFilerError,  
  ecFileCreateError, ecFileOpenError, ecOther);
```

- ↳ Defines an event handler that is called when an exception is raised during an add, extract, freshen, or move process.

Abbrevia traps all exceptions that occur during add, extract, freshen, and move processes. The exception is translated to an ErrorClass and ErrorCode. The OnProcessItemFailure is called and a customized error message can be displayed. Exceptions that are defined by Abbrevia are translated to an ErrorClass of ecAbbrevia. Abbrevia's exceptions (defined in the AbExcept unit) all have an ErrorCode property that uniquely identifies them. The ErrorCode constants are defined in the AbConst unit.

An error string corresponding to the Abbrevia exception can be retrieved using AbStrRes function in the AbConst unit. The following example shows how:

```
if ErrorClass = ecAbbrevia then  
  ShowMessage(AbStrRes[ErrorCode]);
```

If you do not provide an event handler for OnProcessItemFailure, the user is not informed of the error. Abbrevia is designed to continue processing commands when an error is received without corrupting files or archives.

The following is a list of the possible values for ErrorCode:

ErrorCode	Action	Description
AbDuplicateName	Add, move	The file name already exists in the archive. You might want to display a warning to the user, but this could be inefficient if the user is attempting to add the same large set of files to a directory twice. The file is not added or moved.
AbInvalidPassword	Extract	The specified password is not valid for extracting the encrypted file. The file is not extracted.
AbNoSuchDirectory	Extract	The destination directory does not exist. The file is not extracted.
AbUnknownCompressionMethod	Extract	The file was compressed with a compression method that is not supported by Abbrevia. The file is not extracted.
AbUserAbort	Add, extract, freshen	The user aborted the process. The file is not processed.
AbZipBadCRC	Extract	The extracted file's CRC doesn't match the stored CRC. The extracted file is deleted.
AbZipVersionNeeded	Extract	The "Version needed to extract" for this file is not supported by this version of Abbrevia. This might occur in the future when a new version of PKZIP is available. The file is not extracted.

See also: AddFiles, ExtractFiles, ExtractTaggedItems, FreshenFiles, FreshenTaggedItems, OnConfirmProcessItem

## OnRequestImage

event

```
property OnRequestImage : TAbRequestImageEvent  
  TAbRequestImageEvent = procedure(  
    Sender : TObject; ImageNumber : Word;  
    var ImageName : string; var Abort : Boolean) of object;
```

↳ Defines an event handler to obtain a spanned archive file name.

OnRequestImage provides a mechanism to obtain a particular archive file name when working with a spanned set. ImageName specifies the file name of the requested archive file within a spanned set. ImageNumber identifies the file's sequence number within the spanned set.

There are two conditions where this event will be fired — when saving an archive, and when extracting files from an archive.

When saving an archive, if the SpanningThreshold is reached, the OnRequestImage event is fired to obtain the file name (via ImageName) of the next file in the spanned set. If an event handler has not been defined, the image file name is automatically generated by replacing the last character of the file extension with a sequence number, (e.g. Files.Zip, Files.Z01, Files.Z02, etc.). There is a limit of 99 files that can be auto-generated.

When extracting an item from a spanned archive set, OnRequestImage is fired to obtain the archive image containing the item. If the item spans archive images, this event is fired as often as required until the item has been extracted. If an event handler is not assigned then an EAbSpannedImageNotFound exception will be raised when trying to extract from a spanned set.

See also: SpanningThreshold

## OpenArchive

method

```
procedure OpenArchive(const aFileName : string);
```

↳ Opens or creates the specified archive.

OpenArchive sets the FileName property to the value specified. If an archive is currently open, it is saved if necessary and closed. A new archive is opened and loaded, initializing the Count and Items properties.

See also: FileName

**SpanningThreshold****property**


---

```
property SpanningThreshold : Longint
```

Default: 0

↳ Specifies the maximum archive image size.

By specifying SpanningThreshold you can restrict the size of the archive. If the archive size reaches SpanningThreshold when adding or freshening items, the archive is closed and a new archive file is created and the process continues onto the new, spanned archive. This process is repeated until the entire archive has been saved.

Setting SpanningThreshold to 0 specifies that a single archive file is created, up to MaxLongint bytes in size.

Spanning archive files in this fashion requires that a new file name be supplied each time a new archive file needs to be created. This can be done either by providing a specific file name via the OnRequestImage event, or allowing the file name to be automatically generated by appending a two-digit sequence number to the original archive file name, (e.g. Files.Zip, Files.Z01, Files.Z02, etc.). There is a limit of 99 files that can be auto-generated. For larger spanned sets, you must provide an event handler to supply the file names.

See also: OnRequestImage

**Status****run-time, read-only property**


---

```
property Status : TABArchiveStatus
TABArchiveStatus = (asInvalid, asIdle, asBusy);
```

↳ Determines whether an operation is currently being performed on the archive.

When an archive is being initialized, Status is set to asInvalid. When the initialization is finished, Status is set to asIdle. During each operation that modifies the archive, Status is set to asBusy. Status is used internally by Abbrevia to prevent re-entrant operations, but could also be used to display archive activity.

**TagItems****method**


---

```
procedure TagItems(const FileMask : string);
```

↳ Tags the specified archive items.

Abbrevia allows you to perform operations on multiple files in an archive. You can extract, freshen, or delete a group of files by first tagging them and then performing the desired operation.

`TagItems` tags each archive item whose stored name matches `FileMask`. Items can be untagged using `ClearTags` or `UnTagItems`.

See also: `ClearTags`, `TAbZipKit.ExtractTaggedItems`, `TAbZipper.DeleteTaggedItems`, `TAbUnzipper.ExtractTaggedItems`, `TAbZipper.FreshenTaggedItems`, `UnTagItems`, `TAbCabExtractor.ExtractTaggedItems`

---

<b>TempDirectory</b>		<b>property</b>
----------------------	--	-----------------

---

`property TempDirectory : string`

↳ Specifies a temporary directory to use during archive operations.

By setting `TempDirectory`, you specify where the temporary files used by Abbrevia are created. If this property is left null, the system's temporary directory will be used. If the specified directory does not exist, an `EAbNoSuchDirectory` is raised.

---

<b>UnTagItems</b>		<b>method</b>
-------------------	--	---------------

---

`procedure UnTagItems(const FileMask : string);`

↳ Sets the `Tagged` property to False for each archive item whose stored name matches `FileMask`.

See also: `ClearTags`, `TagItems`



---

# Chapter 5: Non-Visual Zip Components

Abbrevia includes four non-visual components that make it easy to implement zip archiving and compression in your application:

- TAbZipBrowser can view the contents of a zip archive (read-only access).
- TAbUnzipper can extract items from a zip archive (read-only access).
- TAbZipper can insert items into a zip archive (read-write access).
- TAbZipKit can extract and insert items in a zip archive (read/write access).

The non-visual zip components descend from the TAbCustomZipBrowser which references a TAbZipArchive, which encapsulates a PKZIP-compatible archive. Through the TAbZipArchive, the TAbZipBrowser can view (and its descendants can modify) the contents of an archive.

---

# TAbCustomZipBrowser Class

TAbCustomZipBrowser class is the immediate ancestor of the TAbZipBrowser component, and also serves as the base class for the non-visual zip archive components: TAbUnZipper, TAbZipKit, and TAbZipper. It implements all of the methods and properties used by its descendants, but no properties are published.

TAbCustomZipBrowser is provided to facilitate creation of descendent zip components. For property and method descriptions, see “TAbZipBrowser Component” on page 71.

## Hierarchy

TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomZipBrowser (AbZBrows)	

# TAbZipBrowser Component

The TAbZipBrowser component is a non-visual component that allows read-only access to a PKZIP-compatible archive. It does not include support for inserting or extracting files in the archive.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomZipBrowser (AbZBrows) .....	70
TAbZipBrowser (AbZBrows)	

## Properties

② ArchiveProgressMeter	② ItemProgressMeter	② SpanningThreshold
② BaseDirectory	Items	② Status
② CompressionType	② LogFile	② TempDirectory
② Count	② Logging	① Version
② FileName	Password	ZipFileComment

## Methods

② ClearTags	② FindItem	② UnTagItems
② CloseArchive	② OpenArchive	
② FindFile	② TagItems	

## Events

② OnArchiveItemProgress	② OnConfirmProcessItem	② OnRequestImage
② OnArchiveProgress	② OnLoad	OnRequestLastDisk
② OnChange	② OnProcessItemFailure	OnRequestNthDisk

## Reference Section

---

<b>Items</b>	<b>read-only property</b>
--------------	---------------------------

---

```
property Items[Index : Integer] : TABZipItem
```

↳ Contains file names for each item in the archive.

Each item in a zip archive is described using a TABZipItem.

Valid values for Index are 0 through Count - 1.

The following example accesses the name of each file in the archive:

5

```
if (Count > 0) then
  for I := 0 to pred(Count) do
    ListBox1.Items.Add := AbZipBrowser1.Items[I].FileName;
```

See also: Count, TABZip.Item

---

### OnRequestLastDisk

**event**

```
property OnRequestLastDisk : TABRequestDiskEvent
```

```
TABRequestDiskEvent = procedure(
  Sender : TObject; var Abort : Boolean) of object;
```

↳ Defines an event handler that is called when the last (removable) disk of a spanned archive is needed.

The directory information for PKZIP files is stored on the last disk of a spanned archive. If you do not supply an OnRequestLastDisk event handler, a dialog box is displayed to prompt the user for the last disk. If it is desired to add features such as allowing the user to verify the disk contents, that can be done in an OnRequestLastDisk event handler. The spanned archive can even be aborted by setting Abort to True inside the event handler.

**Note:** On Linux this functionality is subject to proper floppy drive mount and unmount permissions, and the floppy file system should be mounted as a DOS Compatible FAT format of some type.

See also: OnRequestNthDisk

## OnRequestNthDisk

event

```
property OnRequestNthDisk : TAbRequestNthDiskEvent  
TAbRequestNthDiskEvent = procedure(Sender : TObject;  
DiskNumber : Byte; var Abort : Boolean) of object;
```

↳ Defines an event handler that is called when a specific (removable) disk in the spanned archive is needed.

If you do not supply an OnRequestNthDisk event handler, a dialog box is displayed to prompt for the disk specified by DiskNumber. If you want to add features such as allowing the user to verify the disk contents, you can do that in an OnRequestNthDisk event handler. The spanned archive can even be aborted by setting Abort to True inside the event handler.

**Note:** On Linux this functionality is subject to proper floppy drive mount and umount permissions, and the floppy file system should be mounted as a DOS Compatible FAT format of some type.

See also: OnRequestLastDisk

## Password

property

```
property Password : string
```

↳ Supplies the password used for encrypting or decrypting a file.

When a file is added to the archive, Password is used to encrypt it. If Password is empty, the file is not encrypted.

When an attempt is made to extract an encrypted file from the archive, Password is used to decrypt the encryption header. If the decryption is successful, the remainder of the file is decrypted and extracted. If the encryption header cannot be successfully decrypted or Password is empty, OnNeedPassword in TABUnzipper or TABZipKit is called (if fewer than PasswordRetries attempts have been made). The OnNeedPassword event handler can then provide a password (possibly by prompting the user).

See also: TABUnzipper.OnNeedPassword, TABUnzipper.PasswordRetries,  
TABZipKit.OnNeedPassword, TABZipKit.PasswordRetries

## ZipFileComment

run-time property

```
property ZipFileComment : string
```

↳ Supplies a comment to stored in the Zip archive.

A comment for the archive can be stored in PKZIP-compatible archives.

---

## TAbCustomUnZipper Class

TAbCustomUnZipper class is the immediate ancestor of the TAbUnZipper component. It implements all of the methods and properties used by its descendants but no properties are published.

TAbCustomUnZipper is provided to facilitate creation of descendent components. For property and method descriptions, see “TAbUnZipper Component” on page 75.

### Hierarchy

5

TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomZipBrowser (AbZBrows) .....	70
TAbCustomUnZipper (AbUnzper)	

# TAbUnZipper Component

TAbUnZipper is a non-visual component that extends the capabilities of the TAbZipBrowser by adding the ability to extract files from an archive. TAbZipper is derived from TAbZipBrowser and therefore inherits all of its properties, events, and methods.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomZipBrowser (AbZBrows) .....	70
TAbCustomUnZipper (AbUnzper) .....	74
TAbUnZipper (AbUnzper)	

## Properties

② ArchiveProgressMeter	② ItemProgressMeter	② Status
② BaseDirectory	②LogFile	② TempDirectory
② CompressionType	② Logging	① Version
② Count	>PasswordRetries	
ExtractOptions	② SpanningThreshold	

## Methods

② ClearTags	ExtractTaggedItems	② TagItems
② CloseArchive	ExtractToStream	TestTaggedItems
ExtractAt	② FindFile	② UnTagItems
ExtractFiles	② FindItem	
ExtractFilesEx	② OpenArchive	

## Events

② OnArchiveItemProgress	OnConfirmOverwrite	OnNeedPassword
② OnArchiveProgress	② OnConfirmProcessItem	② OnProcessItemFailure
② OnChange	② OnLoad	② OnRequestImage

# Reference Section

## ExtractAt method

```
procedure ExtractAt(Index : Integer; const NewName : string);
```

↳ Extracts an item with a known item index.

Index is the zero-based item index. NewName is the file name to be given to the item, once extracted, if it is to be different than the stored file name. If NewName is left blank then the item's stored file name will be used.

5

## ExtractFiles method

```
procedure ExtractFiles(const FileMask : string);
```

↳ Extracts all files that match FileMask from the archive.

The files in the archive are not modified by extract operations. The extraction happens immediately.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is extracted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” extracts all files in the archive that have a TXT extension. If path information is included in FileMask, the directory information in the archive item's file name must also match for the file to be extracted.

The following table shows some example FileMasks and stored file names and indicates whether they would be considered a match:

FileMask	Stored File Name	Match?
*.txt	test.txt	Yes
*.txt	doc/test.txt	Yes
*.txt	bin/test.txt	Yes
doc/*.txt	test.txt	No
doc/*.txt	doc/test.txt	Yes
doc/*.txt	bin/test.txt	No
*/*.txt	test.txt	No
*/*.txt	doc/test.txt	Yes
*/*.txt	bin/test.txt	Yes

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See the ExtractOptions property for a description of how the directory is determined.

See also: ExtractFilesEx, ExtractOptions, ExtractTaggedItems

## ExtractFilesEx

method

```
procedure ExtractFilesEx(const FileMask, ExclusionMask : string);
```

↳ Extracts files except those specified by ExclusionMask.

ExtractFilesEx introduces item exception list functionality to ExtractFiles. By specifying ExclusionMask you can exclude certain files that would normally be extracted using the ExtractFiles method.

For example, you could extract all files except text files by the following:

### Windows

```
ExtractFilesEx('*.*', '*.txt');
```

### Linux

```
ExtractFilesEx('*', '*.txt');
```

See also: ExtractFiles

## ExtractOptions

property

```
property ExtractOptions : TAbExtractOptions
TAbExtractOption = (eoCreateDirs, eoRestorePath);
TAbExtractOptions = set of TAbExtractOption;
```

Default: [eoCreateDirs]

↳ Determines the options for archive extract operations.

If ExtractOptions contains eoCreateDirs, directories (including multiple levels of subdirectories) are created as needed to extract files.

If ExtractOptions contains eoRestorePath, path information stored in the archive is retained when files are extracted. The extracted file is placed in the appropriate subdirectory relative to the current BaseDirectory.

The following table shows to which directory the file is extracted for a given file in the archive and a set of ExtractOptions:

<b>ExtractOptions</b>	<b>Name in Archive</b>	<b>Directory for Extracted File</b>
[ ]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
[eoCreateDirs, eoRestorePath]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), which is created if necessary.
[eoCreateDirs]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY, which is created if necessary.
[eoRestorePath]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), if it exists. Otherwise, an exception is raised.

See also: ExtractFiles, ExtractTaggedItems, OnConfirmOverwrite,  
TAbZipBrowser.BaseDirectory

## ExtractTaggedItems

## method

procedure ExtractTaggedItems;

Extracts all files that have their Tagged property set to True.

The files in the archive are not modified by extract operations. The extraction happens immediately.

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions on page 77 for a description of how the directory is determined.

See the TagItems method of the TAbBaseBrowser class on page 66 for information about how tagging works.

See the OnProcessItemFailure event of the TAbBaseBrowser class on page 63 for a description of the exceptions that can occur during the extract process.

See also: ExtractFiles, ExtractOptions, TAbZipBrowser.BaseDirectory,  
TAbZipBrowser.OnConfirmProcessItem, TAbZipBrowser.OnProcessItemFailure,  
TAbZipBrowser.TagItems

---

<b>ExtractToStream</b>	<b>method</b>
------------------------	---------------

---

```
procedure ExtractToStream(
  const File name : string; ToStream : TStream);
```

↳ Extract the specified item directly to a stream.

ExtractToStream will extract an item directly to a TStream instance. The instance must already be created. The stream instance is not cleared and the extracted file is appended to the stream's current position.

---

<b>OnConfirmOverwrite</b>	<b>event</b>
---------------------------	--------------

---

```
property OnConfirmOverwrite : TAbConfirmOverwriteEvent
  TAbConfirmOverwriteEvent = procedure(
    var FileName : string; var Confirm : Boolean) of object;
```

↳ Defines an event handler that is called if the output file for an extract operation already exists.

This event handler is called prior to the start of the extract process if the expected output file already exists. If Confirm is set to False, the extract process is aborted for that item. If Confirm is set to True or a handler is not provided for this event, the file is overwritten without warning.

See also: ExtractFiles, ExtractTaggedItems

## OnNeedPassword

event

```
property OnNeedPassword : TABNeedPasswordEvent  
TABNeedPasswordEvent = procedure(  
  Sender : TObject; var NewPassword : string) of object;
```

↳ Defines an event handler that is called to allow entry of a password when decrypting a file.

The OnNeedPassword event handler is called when an encrypted file is being extracted and one of the following occurs:

- The Password property is empty.
- The Password is not valid for the encrypted file and the number of attempts is less than PasswordRetries.

This event causes the Password property to be updated with the value in NewPassword.

See also: PasswordRetries, Password

## PasswordRetries

property

```
property PasswordRetries: Byte
```

Default: 3

↳ Specifies the maximum number of passwords to try when attempting to extract an encrypted file.

When the number of retries is exhausted, an exception is raised. See the OnProcessItemFailure event on page 63 for a description of how exceptions that occur during the extract process are handled.

If PasswordRetries is 0 and Password is empty, encrypted files cannot be extracted.

See also: TAbZipBrowser.Password, TAbZipKit.OnNeedPassword,  
TAbZipKit.PasswordRetries

```
procedure TestTaggedItems;
```

↳ Verifies the integrity of each tagged item in the archive.

For each tagged item in the archive the following are checked:

- The central directory record.
- The local file header.
- The CRC for the file.

Some of these checks require the file to be extracted. Abbrevia extracts the file to a special stream and then, after the checks are made, the stream is destroyed.

---

# TAbCustomZipper Class

TAbCustomZipper class is the immediate ancestor of the TAbZipper component. It implements all of the methods and properties used by its descendants but no properties are published.

TAbCustomZipper is provided to facilitate creation of descendent components. For property and method descriptions, see “TAbZipper Component” on page 83.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TABBaseBrowser (AbBrowse) .....	56
TAbCustomZipBrowser (AbZBrows) .....	70
TAbCustomZipper (AbZipper)	

---

# TabZipper Component

TabZipper is a non-visual component that extends the capabilities of the TabZipBrowser by adding the ability to add, freshen, or move files in an archive. TabZipper is derived from TAbZipBrowser and therefore inherits all of its properties, events, and methods.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomZipBrowser (AbZBrows) .....	70
TAbCustomZipper (AbZipper) .....	82
TAbZipper (AbZipper)	

## Properties

② ArchiveProgressMeter	DeflationOption	② SpanningThreshold
AutoSave	DOSMode	② Status
② BaseDirectory	② FileName	StoreOptions
CompressionMethodToUse	② ItemProgressMeter	② TempDirectory
② CompressionType	②LogFile	① Version
② Count	② Logging	

## Methods

AddFiles	DeleteFilesEx	Move
AddFilesEx	DeleteTaggedItems	② OpenArchive
AddFromStream	② FindFile	Replace
② ClearTags	② FindItem	Save
② CloseArchive	FreshenFiles	② TagItems
DeleteAt	FreshenFilesEx	② UnTagItems
DeleteFiles	FreshenTaggedItems	

5

## Events

② OnArchiveItemProgress	OnConfirmSave	OnRequestImage
② OnArchiveProgress	② OnLoad	OnSave
② OnChange	② OnProcessItemFailure	
② OnConfirmProcessItem	OnRequestBlankDisk	

# Reference Section

## AddFiles

method

```
procedure AddFiles(  
  const FileMask : string; SearchAttr : Integer);
```

↳ Adds the files that match FileMask to the archive.

The wild card characters '\*' and '?' are allowed in FileMask.

When AddFiles is searching for files that match FileMask, it searches certain directories, depending on the StoreOptions and BaseDirectory properties. The following table shows which directories are searched for the file of a given FileMask and set of StoreOptions. If the file is found, it is stored in the archive with the name shown in the last column. The following table shows examples of Windows (DOS) FileMasks:

StoreOptions	DOS FileMask	Directories Searched	Name in Archive
[ ]	README . TXT	BASEDIRECTORY	README . TXT
	DOC\ README . TXT	BASEDIRECTORY\ DOC	DOC/ README . TXT
	\DOC\ README . TXT	DEFAULTDRIVE: \DOC	DOC/ README . TXT
	X:\DOC\ README . TXT	X:\DOC	DOC/ README . TXT
[ soRecurse, soStripPath ]	README . TXT	BASEDIRECTORY and all subdirectories	README . TXT
	DOC\ README . TXT	BASEDIRECTORY\ DOC	README . TXT
	\DOC\ README . TXT	DEFAULTDRIVE: \DOC	README . TXT
	X:\DOC\ README . TXT	X:\DOC	README . TXT
[ soRecurse ]	README . TXT	BASEDIRECTORY and all subdirectories	README . TXT or SUBDIR/ README . T XT
	DOC\ README . TXT	BASEDIRECTORY\ DOC	DOC/ README . TXT
	\DOC\ README . TXT	DEFAULTDRIVE: \DOC	DOC/ README . TXT
	X:\DOC\ README . TXT	X:\DOC	DOC/ README . TXT
[ soStripPath ]	README . TXT	BASEDIRECTORY	README . TXT

<b>StoreOptions</b>	<b>DOS FileMask</b>	<b>Directories Searched</b>	<b>Name in Archive</b>
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT
	X:\DOC\README.TXT	X:\DOC	README.TXT

The following table shows examples of Linux FileMasks:

<b>StoreOptions</b>	<b>Linux FileMask</b>	<b>Directories Searched</b>	<b>Name in Archive</b>
[ ]	README.TXT	BASEDIRECTORY	README.TXT
	DOC/README.TXT	BASEDIRECTORY/DOC	DOC/README.TXT
	/DOC/README.TXT	/DOC	DOC/README.TXT
[soRecurse, soStripPath]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT
	DOC/README.TXT	BASEDIRECTORY/DOC	README.TXT
	/DOC/README.TXT	/DOC	README.TXT
[soRecurse]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT or SUBDIR/README.T XT
	DOC/README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	/DOC/README.TXT	/DOC	DOC/README.TXT
[soStripPath]	README.TXT	BASEDIRECTORY	README.TXT
	DOC/README.TXT	BASEDIRECTORY/DOC	README.TXT
	/DOC/README.TXT	/DOC	README.TXT

The SearchAttr parameter allows searching for particular classes of file system entities and varies by platform. If SearchAttr is zero, only normal files that match FileMask are added. If SearchAttr is set to one of the file attribute constants defined in the SysUtils unit, the appropriate type of special files are also added. To add multiple types of special files, add the file attribute constants together.

## Windows

The options are faReadOnly, faHidden, faSysFile, faVolumeID, faDirectory, and faArchive. Example: to search for read-only and hidden files in addition to normal files, pass (faReadOnly + faHidden) as the SearchAttr parameter.

## Linux

The options are faLinuxDir, faFileLink faChrFile, faBlkFile, faFifoFile, and faSockFile. Example: to search for directories and symbolic links in addition to normal files, pass (faLinuxDir + faFileLink) as the SearchAttr parameter.

See also: AddFilesEx

AddFilesEx	method
------------	--------

```
procedure AddFilesEx(
  const FileMask, ExclusionMask : string; SearchAttr : Integer);
```

↳ Adds the files that match FileMask, except those specified by ExclusionMask, to the archive.

AddFilesEx introduces item exception list functionality to AddFiles. By specifying ExclusionMask you can exclude certain files that would normally be added using the AddFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could add all files except text files by the following:

## Windows

```
AddFilesEx('*.*', '*.txt', 0);
```

## Linux

```
AddFilesEx('*', '*.txt', 0);
```

See also: AddFiles

AddFromStream	method
---------------	--------

```
procedure AddFromStream(
  const NewName : string; FromStream : TStream);
```

↳ Create a zip item directly from a stream.

AddFromStream will create a zip item by compressing data directly from a TStream instance. A file name must be provided via NewName for the new zip item that is created. The stream's position is set to 0 and the entire stream is added and the zip item properties are set to reflect the item's compressed size, uncompressed size, CRC32, etc. Adding data directly from a stream causes the archive to be automatically saved since the stream object may be subsequently freed or modified.

**AutoSave****property**

```
property AutoSave : Boolean
```

Default: False

↳ Controls whether changes to the archive are performed immediately.

When AutoSave is True, each change to the archive forces the disk image of the archive to be updated immediately. When AutoSave is False, changes to the disk image are done when one of the following occur:

- The archive is explicitly saved using the Save method.
- Multiple operations on a single item require the archive to be saved.
- File name is changed, causing a new archive to be opened.

**Note:** When AutoSave is True, only a single archive file may be created. Multiple archive file spanning requires that AutoSave be set to False. Also, keep in mind that saving the archive each time a file is added can be very time consuming. For these reasons, we recommend that AutoSave be used only with relatively small archives.

See also: OnConfirmSave, Save, TAbZipBrowser.FileName

**CompressionMethodToUse****property**

```
property CompressionMethodToUse : TAbZipSupportedMethod
```

```
TAbZipSupportedMethod = (smStored, smDeflated, smBestMethod);
```

Default: smBestMethod

↳ Selects the compression method used when adding or freshening items in the archive.

The possible values for CompressionMethodToUse are:

Method	Description
smStored	The file is stored without any compression.
smDeflated	The file is compressed using the deflate method.
smBestMethod	The file is deflated, but if the resulting file is larger than the original, the file is simply stored.

See also: AddFiles

## DeflationOption

property

```
property DeflationOption : TAbZipDeflationOption  
TAbZipDeflationOption = (  
  doInvalid, doNormal, doMaximum, doFast, doSuperFast);
```

Default: doNormal

↳ Determines whether priority is given to compression or speed during compression.

DeflationOption allows you to select the deflation option when CompressionMethodToUse is smDeflated or smBestMethod. You can select varying trade-offs between compression and speed. The choices are Maximum (most compression, slowest speed), Normal, Fast, and Super Fast (least compression, fastest speed).

These choices correspond to the PKZIP 2.04g -ex, -en, -ef, and -es options.

See also: CompressionMethodToUse

## DeleteAt

method

```
procedure DeleteAt(Index : Integer);
```

↳ Deletes the item at the specified item index.

DeleteAt can be used to delete an item from an archive when its index is known. Index is the zero-based item index of the item to be deleted.

## DeleteFiles

method

```
procedure DeleteFiles(const FileMask : string);
```

↳ Deletes the specified files from the archive.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is marked to be deleted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” marks all files in the archive that have a TXT extension to be deleted. If path information is included in FileMask, the directory information in the archive item's file name must match for the file to be deleted.

The next time the archive is saved, all marked files and their corresponding information are deleted from the archive.

See also: DeleteTaggedItems, Save, TAbZipBrowser.OnChange

**DeleteFilesEx****method**

```
procedure DeleteFilesEx(const FileMask, ExclusionMask : string);
```

- Deletes files except those specified by ExclusionMask.

DeleteFilesEx introduces item exception list functionality to DeleteFiles. By specifying ExclusionMask you can exclude certain files that would normally be deleted using the DeleteFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could delete all files from the archive except text files by the following:

**Windows**

```
DeleteFilesEx('*.*', '*.txt');
```

**Linux**

```
DeleteFilesEx('*', '*.txt');
```

See also: DeleteFiles

**DeleteTaggedItems****method**

```
procedure DeleteTaggedItems;
```

- Finds all files that have their Tagged property set to True and marks them to be deleted.

The next time the archive is saved, all marked files and their corresponding information are deleted from the archive.

See the TagItems method of the TAbBaseBrowser class on page 66 for information about how tagging works.

See also: DeleteFiles, Save, TAbZipBrowser.ClearTags, TAbZipBrowser.OnChange,  
TAbZipBrowser.TagItems, TAbZipBrowser.UnTagItems

**DOSMode****property**

```
property DOSMode : Boolean
```

Default: False

- Forces all new files stored in an archive to have DOS-compatible file names.

Windows 95 and Windows NT introduced long file names, which were not available under older versions of Windows or DOS. The older operating systems cannot handle the long file names; they require a file name no longer than eight characters (8.3 format). If the archive must be read by an older operating system, short file names can be forced by setting DOSMode to True.

## Windows

The shortened version of the file name is determined by the Windows GetShortPathName API call.

If DOSMode is True, a file named C:\PROGRAM FILES\README.TXT is stored in the archive as PROGRA~1/README.TXT.

## Linux

The shortened version of the file name is determined by a simple algorithm similar to what is used on Windows to generate short file names.

If DOSMode is True, a file named /home/mydir/longdirname/SomeLongFile name will be stored in the archive as something like home/mydir/longd~01/SomeL~01.

### FreshenFiles

### method

```
procedure FreshenFiles(const FileMask : string);
```

↳ Freshens the specified files in the archive.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is marked to be freshened. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” marks all files in the archive that have a TXT extension to be freshened. If path information is included in FileMask, the directory information in the archive item's file name must match for the file to be freshened.

The next time the archive is saved, the marked files are freshened. If the file is found on disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file.

The directories that are searched to find the file depend on the StoreOptions and BaseDirectory properties. The following table shows, for a given file in the archive and a set of StoreOptions, what directories are searched to find the file to use to freshen the file in the archive.

The following table shows examples of Windows directories searched:

<b>StoreOptions</b>	<b>Name in Archive</b>	<b>Directories Searched</b>
[ ]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY\DOC
[soRecurse, soStripPath]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY\DOC
[soRecurse]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY\DOC
[soStripPath]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY\DOC

The following table shows examples of Linux directories searched:

<b>StoreOptions</b>	<b>Name in archive</b>	<b>Directories Searched</b>
[ ]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY/DOC
[soRecurse, soStripPath]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY/DOC
[soRecurse]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY/DOC
[soStripPath]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY/DOC

See also: FreshenTaggedItems, Save, StoreOptions, TAbZipBrowser.OnChange

```
procedure FreshenFilesEx(const FileMask, ExclusionMask : string);
```

- ↳ Freshen files except those specified by ExclusionMask.

FreshenFilesEx introduces item exception list functionality to FreshenFiles. By specifying ExclusionMask you can exclude certain files that would normally be freshened by using the FreshenFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, all files could be freshened except text files by the following:

**Windows**

```
FreshenFilesEx('*.*', '*.txt');
```

**Linux**

```
FreshenFilesEx('*', *.txt');
```

See also: FreshenFiles

```
procedure FreshenTaggedItems;
```

- ↳ Finds all files that have their Tagged property set to True and marks them to be freshened.

The next time the archive is saved, all marked files are freshened. If the file is found on the disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file.

The directories that are searched to find the file depend on the StoreOptions and BaseDirectory properties. See the table in FreshenFiles for details of determining the directory to search.

See the TagItems method of the TAbBaseBrowser class on page 66 for information about how tagging works.

See the OnProcessItemFailure event of the TAbBaseBrowser class on page 63 for a description of the exceptions that can occur during the freshen process.

See also: FreshenFiles, FreshenFilesEx

**Move****method**

```
procedure Move(
  aItem : TABArchiveItem; const NewStoredPath : string);
```

↳ Renames an existing archive item.

Move modifies the stored file name of the archive item only in the memory representation of the archive. It marks the file to be moved (renamed). The next time the archive is saved, the stored file name is physically changed in the archive on disk. NewStoredPath can include path information.

See the OnProcessItemFailure event of the TAbBaseBrowser class on page 63 for a description of the exceptions that can occur during the move process.

See also: AutoSave, Save, TAbZipBrowser.OnChange

**OnConfirmSave****event**

```
property OnConfirmSave : TABArchiveConfirmEvent
TAbArchiveConfirmEvent = procedure (
  Sender : TObject; var Confirm : Boolean) of object;
```

↳ Defines an event handler that is called before the archive is saved.

This event handler is called immediately before an archive is updated. If Confirm is set to False, the update is aborted. If True, the update continues. If a handler is not provided for this event, all processing continues as if Confirm were set to True.

The OnConfirmSave event can be used to allow the user to close an archive without saving changes.

This is most useful when the AutoSave property is False. When the component's File name property is changed, the Save method is automatically called, which fires the OnConfirmSave event. If the OnConfirmSave event handler sets Confirm to False, then the archive is not updated (all pending changes are discarded).

See also: AutoSave, Save, TAbBaseBrowser.File name

**OnRequestBlankDisk****event**

```
property OnRequestBlankDisk : TABRequestDiskEvent
TABRequestDiskEvent = procedure(
  Sender : TObject; var Abort : Boolean) of object;
```

↳ Defines an event handler that is called when a removable blank disk is needed for a spanned archive.

If you do not supply an OnRequestBlankDisk event handler, a dialog box is displayed to prompt the user for a blank disk. If you want to add features such as formatting the disk, scanning the disk for errors, or allowing the user to verify the disk contents, you can do that in an OnRequestBlankDisk event handler. Spanning can be aborted by setting Abort to True inside the event handler.

**Note:** On Linux this functionality is subject to proper floppy drive mount and umount permissions, and the floppy file system should be mounted as a DOS Compatible FAT format of some type.

**OnSave****event**

```
property OnSave : TABArchiveEvent
TABArchiveEvent = procedure(Sender : TObject) of object;
```

↳ Defines an event handler that is called immediately after the archive's contents are saved.

The OnSave event can be used along with the OnConfirmProcessItem event to display status information for the user. For example, in the OnConfirmProcessItem event handler, the status could be set to "change pending", and in the OnSave event handler, set the status to "saved". This would be most useful when the AutoSave property is False.

See also: OnConfirmProcessItem, OnConfirmSave, Save

## Replace

## method

---

```
procedure Replace(aItem : TAbArchiveItem);
```

- ⌚ Replaces the specified item in the archive.

The next time the archive is saved, the item is replaced. If the file specified by the item's `FileName` property is found on disk, the compressed data in the archive is replaced with the compressed representation of the current file. If the file cannot be found an `EAbFileNotFoundException` exception is raised.

See [OnProcessItemFailure](#) on page 63 for a description of the exceptions that can occur during the replacement process.

5

See also: [OnChange](#), [Save](#), [StoreOptions](#)

## Save

## method

---

```
procedure Save;
```

- ⌚ Save updates the archive on disk.

If the archive has not been modified since it was last opened or saved, `Save` returns immediately without modifying the archive.

See also: [AutoSave](#), [OnConfirmSave](#)

```
property StoreOptions : TAbStoreOptions  
  TABStoreOption = (soStripDrive, soStripPath, soRemoveDots,  
    soRecurse, soFreshen, soReplace);  
  
  TABStoreOptions = set of TABStoreOption;  
  
  Default: [soStripDrive, soRemoveDots]
```

↳ Determines the options for archive add and freshen operations.

The following table lists the results of the available options:

Option	Result
soStripDrive	Drive letter information is removed from the stored file name. ( <b>Note:</b> This option is ignored in Linux.)
soStripPath	All path information is removed from the stored file name.
soRemoveDots	All relative path information is removed from the stored file name. For example, if you call AddFiles with a FileMask of "..\TEST.TXT" ("../TEST.TXT" in Linux), the parent of the current BaseDirectory is searched for a file named "TEST.TXT". If the file is found, it is stored as "TEST.TXT".
soRecurse	Subdirectories of the search path are included in the search for files to add or freshen.
soFreshen	When adding an existing item to the archive, the item is freshened.
soReplace	When adding an existing item to the archive, the item is replaced.

See also: AddFiles, FreshenFiles, FreshenTaggedItems

---

# TAbCustomZipKit Class

TAbCustomZipKit class is the immediate ancestor of the TAbZipKit component. It implements all of the methods and properties used by its descendants but no properties are published.

TAbCustomZipKit is provided to facilitate creation of descendent components. For property and method descriptions, see “TAbZipKit Component” on page 99.

## Hierarchy

5

### TComponent

TAbBaseComponent (AbBase) . . . . .	54
TAbBaseBrowser (AbBrowse) . . . . .	56
TAbCustomZipBrowser (AbZBrows) . . . . .	70
TAbCustomZipper (AbZipper) . . . . .	82
TAbCustomZipKit (AbZipKit)	

---

# TAbZipKit Component

TAbZipKit is a non-visual component that gives you everything—with it you can browse an archive, add files, and extract files. It combines the abilities of TAbUnzipper and TAbZipper. TAbZipKit is derived from TAbCustomZipper, and adds all the functionality of the TAbUnZipper.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomZipBrowser (AbZBrows) .....	70
TAbCustomZipper (AbZipper) .....	82
TAbCustomZipKit (AbZipKit) .....	98
TAbZipKit (AbZipKit)	

## Properties

② ArchiveProgressMeter	② ItemProgressMeter	② Status
② BaseDirectory	②LogFile	② TempDirectory
② CompressionType	② Logging	① Version
② Count	PasswordRetries	
ExtractOptions	② SpanningThreshold	

## Methods

- |                |                    |                 |
|----------------|--------------------|-----------------|
| ② ClearTags    | ExtractTaggedItems | ② TagItems      |
| ② CloseArchive | ExtractToStream    | TestTaggedItems |
| ExtractAt      | ② FindFile         | ② UnTagItems    |
| ExtractFiles   | ② FindItem         |                 |
| ExtractFilesEx | ② OpenArchive      |                 |

## Events

- |                         |                        |                        |
|-------------------------|------------------------|------------------------|
| ② OnArchiveItemProgress | OnConfirmOverwrite     | OnNeedPassword         |
| ② OnArchiveProgress     | ② OnConfirmProcessItem | ② OnProcessItemFailure |
| ② OnChange              | ② OnLoad               | ② OnRequestImage       |

# Reference Section

## ExtractAt

method

```
procedure ExtractAt(Index : Integer; const NewName : string);
```

↳ Extracts an item with a known item index.

ExtractAt can be used to extract an item from a zip archive when its item index is known. Index is the zero-based item index. NewName is the file name to be given to the item, once extracted, if different than the stored file name. If NewName is left blank then the item's stored file name will be used.

5

## ExtractFiles

method

```
procedure ExtractFiles(const FileMask : string);
```

↳ Extracts all files that match FileMask from the archive.

The files in the archive are not modified by extract operations. The extraction happens immediately.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is extracted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” extracts all files in the archive that have a TXT extension. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be extracted.

The following table shows some example FileMasks and stored file names and indicates whether they would be considered a match:

FileMask	Stored File Name	Match?
*.txt	test.txt	Yes
*.txt	doc/test.txt	Yes
*.txt	bin/test.txt	Yes
doc/*.txt	test.txt	No
doc/*.txt	doc/test.txt	Yes
doc/*.txt	bin/test.txt	No
*/*.txt	test.txt	No
*/*.txt	doc/test.txt	Yes
*/*.txt	bin/test.txt	Yes

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See the ExtractOptions property on page 77 for a description of how the directory is determined.

## **ExtractFilesEx**

**method**

```
procedure ExtractFilesEx(const FileMask, ExclusionMask : string);
```

- ↳ Extracts files except those specified by ExclusionMask.

ExtractFilesEx introduces item exception list functionality to ExtractFiles. By specifying ExclusionMask you can exclude certain files that would normally be extracted using the ExtractFiles method.

For example, all of the files could be extracted except text files by using the following code:

### **Windows**

```
ExtractFilesEx('*.*', '*.txt');
```

### **Linux**

```
ExtractFilesEx('*', '*.txt');
```

See also: ExtractFiles

## **ExtractOptions**

**property**

```
property ExtractOptions : TAbExtractOptions
TAbExtractOption = (eoCreateDirs, eoRestorePath);
TAbExtractOptions = set of TAbExtractOption;
```

Default: [eoCreateDirs]

- ↳ Determines the options for archive extract operations.

If ExtractOptions contains eoCreateDirs, directories (including multiple levels of subdirectories) are created as needed to extract files.

If ExtractOptions contains eoRestorePath, path information stored in the archive is retained when files are extracted. The extracted file is placed in the appropriate subdirectory relative to the current BaseDirectory.

For a given file in the archive and a set of ExtractOptions, the following table shows, to which directory the file is extracted:

<b>ExtractOptions</b>	<b>Name in Archive</b>	<b>Directory for Extracted File</b>
[ ]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
[ eoCreateDirs, eoRestorePath]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), which is created if necessary.
[ eoCreateDirs]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY, which is created if necessary.
[ eoRestorePath]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), if it exists. Otherwise, an exception is raised.

See also: [ExtractFiles](#), [ExtractTaggedItems](#), [OnConfirmOverwrite](#),  
[TAbZipBrowser.BaseDirectory](#)

**ExtractTaggedItems****method**

```
procedure ExtractTaggedItems;
```

- ↳ Extracts all files that have their Tagged property set to True.

The files in the archive are not modified by extract operations. The extraction happens immediately.

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions on page 77 for a description of how the directory is determined.

See the TagItems method of the TAbBaseBrowser class on page 66 for information about how tagging works.

See the OnProcessItemFailure event of the TAbZipBrowser class on page 63 for a description of the exceptions that can occur during the extract process.

See also: ExtractFiles, ExtractOptions, TAbZipBrowser.BaseDirectory,  
 TAbZipBrowser.OnConfirmProcessItem, TAbZipBrowser.OnProcessItemFailure,  
 TAbZipBrowser.TagItems

**ExtractToStream****method**

```
procedure ExtractToStream(
  const File name : string; ToStream : TStream);
```

- ↳ Extracts the specified item directly to a stream.

ExtractToStream will extract an item directly to a TStream instance. The instance must already be created. The stream instance is not cleared and the extracted file is appended to the stream's current position.

**OnConfirmOverwrite****event**

```
property OnConfirmOverwrite : TAbConfirmOverwriteEvent
TAbConfirmOverwriteEvent = procedure(
  var File name : string; var Confirm : Boolean) of object;
```

- ↳ Defines an event handler that is called if the output file for an extract operation already exists.

This event handler is called prior to the start of the extract process if the expected output file already exists. If Confirm is set to False, the extract process is aborted for that item. If a handler is not provided for this event, the file is overwritten without warning.

See also: ExtractFiles, ExtractTaggedItems

## OnNeedPassword

event

```
property OnNeedPassword : TAbNeedPasswordEvent  
  TAbNeedPasswordEvent = procedure(  
    Sender : TObject; var NewPassword : string) of object;
```

↳ Defines an event handler that is called to allow entry of a password when decrypting a file.

The OnNeedPassword event handler is called when an encrypted file is being extracted and one of the following occurs:

- The Password property is empty.
- The Password is not valid for the encrypted file and the number of attempts is less than PasswordRetries.

The event causes the Password property to be updated with the value of NewPassword.

See also: PasswordRetries, Password

## PasswordRetries

property

```
property PasswordRetries: Byte
```

Default: 3

↳ Specifies the maximum number of passwords to try when attempting to extract an encrypted file.

When the number of retries is exhausted, an exception is raised. See OnProcessItemFailure event on page 63 for a description of how exceptions that occur during the extract process are handled.

If PasswordRetries is 0 and Password is empty, encrypted files cannot be extracted.

See also: TAbBaseBrowser.Password, OnNeedPassword, PasswordRetries

```
procedure TestTaggedItems;
```

- ⌚ Verifies the integrity of each tagged item in the archive.

For each tagged item in the archive the following things are checked:

- The central directory record
- The local file header
- The CRC for the file

5

Some of these checks require the file to be extracted. Abbrevia extracts the file to a special stream, and then, after the checks are made, the stream is destroyed.

# Chapter 6: Cabinet Components (Windows Only)

Abbrevia provides several components that make it easy to add CAB file support to your application. The Abbrevia cabinet components provide a friendly interface between your application and the Microsoft CABINET.DLL. The cabinet components are modeled after their sibling zip archive components and provide much of the same functionality in addition to their cabinet specific functions. Files can be extracted from a cabinet in exactly the same manner that a file is extracted from a zip, firing similar events. Thus, adding cabinet archiving capability to an application with existing Abbrevia zip components can be as simple as dropping a component on the form and hooking into the existing event handlers.

Cabinet file processing is more restricted than zip file processing. A cabinet file can be used in two modes: browsing and extracting (read-only); or creating a new cabinet (write-only). There is no equivalent to delete, freshen, and replace operations with zip files.

The Abbrevia cabinet components require the Microsoft CABINET.DLL and are available to 32-bit programs only.

The Abbrevia cabinet components consist of four non-visual components:

- TAbCabBrowser provides access to the contents of a cabinet (read-only access).
- TAbCabExtractor can extract files from a cabinet (read-only access).
- TAbMakeCab can create a cabinet (write access).
- TAbCabKit combines the functionality of TAbCabExtractor and TAbMakeCab into one component.

These components all descend from the TAbBaseBrowser class which provides their common functionality and reference to the abstract TAbArchive class. The cabinet components descend from the TAbCustomCabBrowser class which contains a reference to a TAbCabArchive, which in turn encapsulates a cabinet file archive. Through the TAbCabArchive, the TAbCabBrowser can view the contents of an archive. The other non-visual components are derived from the TAbCustomCabBrowser, so they can also access the contained TAbCabArchive object.

---

## TAbCustomCabBrowser Class

TAbCustomCabBrowser class is the immediate ancestor of the TAbCabBrowser component and also serves as the base class for the non-visual cabinet components, TAbCabExtractor and TAbMakeCab. It implements all of the methods and properties used by its descendants but no properties are published.

TAbCustomCabBrowser is provided to facilitate creation of descendent cabinet components. For property and method descriptions, see “TAbCabBrowser Component” on page 109.

### Hierarchy

6

TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbCBrows)	

---

# TAbCabBrowser Component

TAbCabBrowser is a non-visual component that provides read-only access to a cabinet archive. It does not include support for building a cabinet or extracting files from a cabinet.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbCBrows) .....	108
TAbCabBrowser (AbCBrows)	

6

## Properties

② ArchiveProgressMeter	FolderCount	SetID
② BaseDirectory	HasNext	② SpanningThreshold
CabSize	HasPrev	② Status
② CompressionType	② ItemProgressMeter	② TempDirectory
② Count	Items	① Version
CurrentCab	②LogFile	
② FileName	②Logging	

## Methods

② ClearTags	② FindItem	② UnTagItems
② CloseArchive	② OpenArchive	
② FindFile	② TagItems	

## Events

② OnArchiveItemProgress	② OnConfirmProcessItem	② OnRequestImage
② OnArchiveProgress	② OnLoad	
② OnChange	② OnProcessItemFailure	

## Reference Section

---

<b>CabSize</b>	<b>run-time, read-only property</b>
----------------	-------------------------------------

---

property CabSize : Longint

- ↳ Contains the size of the cabinet file in bytes.

For existing cabinets, CabSize is static since the cabinet contents cannot be modified. When building a new cabinet, CabSize is updated after the folder (compression block) in progress has been completed and flushed to disk. This occurs either by a call to NewFolder or automatically when the FolderThreshold value has been reached.

See also: TAbMakeCab.FolderThreshold, TAbMakeCab.NewFolder

6

---

<b>CurrentCab</b>	<b>run-time, read-only property</b>
-------------------	-------------------------------------

---

property CurrentCab : Word

- ↳ The number of the current cabinet within a cabinet set.

CurrentCab contains the zero-based sequence number for the current cabinet within a cabinet set. CurrentCab will be zero if the cabinet is not part of a spanned set, or is the first cabinet in a spanned set.

---

<b>FolderCount</b>	<b>run-time, read-only property</b>
--------------------	-------------------------------------

---

property FolderCount : Word

- ↳ The number of folders (compression blocks) in the cabinet.

FolderCount also includes any partial folders if the cabinet is part of a spanned cabinet set.

---

<b>HasNext</b>	<b>run-time, read-only property</b>
----------------	-------------------------------------

---

property HasNext : Boolean

- ↳ Indicates whether the cabinet is chained to the next cabinet or not.

HasNext is True if file data spans from the current cabinet into another. HasNext is False if the cabinet is the last cabinet in a spanned set, or if the cabinet is not part of a spanned set.

See also: HasPrev

## **HasPrev**

**run-time, read-only property**

```
property HasPrev : Boolean
```

↳ Indicates whether the first file in the cabinet spans from the previous cabinet of a set.

If HasPrev is True, then the current cabinet is part of a spanned cabinet set, and its first item is a continuation from the previous cabinet.

See also: HasNext

## **Items**

**run-time, read-only property**

```
property Items[Index : Integer] : TAbCabItem
```

↳ Contains file names for each item in the current cabinet.

Each item in a cabinet is described using a TAbCabItem. Valid values for Index are 0 through Count - 1.

The following example accesses the name of each file in the archive:

```
if (Count > 0) then
  for I := 0 to pred(Count) do
    ListBox1.Items.Add := AbCabBrowser1.Items[I].FileName;
```

See also: TAbBaseBrowser.Count

## **SetID**

**property**

```
property SetID : Word
```

↳ Contains the application-defined set identification number.

SetID is determined by the application and can be any word value. Setting SetID when the cabinet is opened for decompression does not change the value.

---

# TAbCustomCabExtractor Class

TAbCustomCabExtractor class is the immediate ancestor of the TAbCabExtractor component. It implements all of the methods and properties used by the TAbCabExtractor component and is identical to the TabCabExtractor component except that no properties are published.

TAbCustomCabExtractor is provided to facilitate creation of descendent cabinet file extracting components. For property and method descriptions, see “TAbCabExtractor Component” on page 113.

## Hierarchy

6

TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbCBrows).....	108
TAbCustomCabExtractor (AbCabExt)	

---

# TAbCabExtractor Component

TAbCabExtractor is a non-visual component that extends the capabilities of the TAbCabBrowser by adding the ability to extract files from a cabinet archive.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbCBrows).....	108
TAbCustomCabExtractor (AbCabExt) .....	112
TAbCabExtractor (AbCabExt)	

## Properties

② ArchiveProgressMeter	② FileName	② Status
② BaseDirectory	② ItemProgressMeter	② TempDirectory
② CompressionType	②LogFile	① Version
② Count	② Logging	
ExtractOptions	② SpanningThreshold	

## Methods

② ClearTags	ExtractAt	② OpenArchive
② CloseArchive	ExtractFiles	② TagItems
② FindFile	ExtractFilesEx	② UnTagItems
② FindItem	ExtractTaggedItems	

## Events

② OnArchiveItemProgress	OnConfirmOverwrite	② OnProcessItemFailure
② OnArchiveProgress	② OnConfirmProcessItem	② OnRequestImage
② OnChange	② OnLoad	

## Reference Section

### ExtractAt method

```
procedure ExtractAt(Index : Integer; NewName : string);
```

- ↳ Extracts an item with known item index.

Index is the zero-based index of the item. NewName is the file name to be given to the item if different than the stored file name. If NewName is left blank then the item's stored file name will be used.

### ExtractFiles method

6

```
procedure ExtractFiles(const FileMask : string);
```

- ↳ Extracts all files that match FileMask from the cabinet.

Each file in the cabinet is compared to FileMask. If a cabinet item's stored file name matches FileMask, the file is extracted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” extracts all files in the archive that have a TXT extension. If path information is included in FileMask, the directory information in the cabinet item's file name must match for the file to be extracted.

The following table shows some example FileMasks and stored file names and indicates whether they would be considered a match:

FileMask	Stored File Name	Match?
*.txt	test.txt	Yes
*.txt	doc/test.txt	Yes
*.txt	bin/test.txt	Yes
doc/*.txt	test.txt	No
doc/*.txt	doc/test.txt	Yes
doc/*.txt	bin/test.txt	No
*/*.txt	test.txt	No
*/*.txt	doc/test.txt	Yes
*/*.txt	bin/test.txt	Yes

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions for a description of how the directory is determined.

See also: ExtractFilesEx, ExtractOptions, ExtractTaggedItems, BaseDirectory, OnConfirmProcessItem, OnProcessItemFailure

### ExtractFilesEx

method

```
procedure ExtractFilesEx(const FileMask, ExclusionMask : string);
```

↳ Extracts all files that match FileMask but not ExclusionMask.

ExtractFilesEx introduces item exception list functionality to ExtractFiles. By specifying ExclusionMask you can exclude certain files that would normally be extracted using the ExtractFiles method.

For example, you could extract all files except text files using the following code:

```
ExtractFilesEx('*.*', '*.txt');
```

See also: ExtractFiles

### ExtractOptions

property

```
property ExtractOptions : TAbExtractOptions  
TAbExtractOption = (eoCreateDirs, eoRestorePath);  
TAbExtractOptions = set of TAbExtractOption;
```

Default: [eoCreateDirs]

↳ Determines the options for archive extract operations.

If ExtractOptions contains eoCreateDirs, directories (including multiple levels of subdirectories) are created as needed to extract files.

If ExtractOptions contains eoRestorePath, path information stored in the archive is retained when files are extracted. The extracted file is placed in the appropriate subdirectory relative to the current BaseDirectory.

For a given file in the cabinet and a set of ExtractOptions, the following table shows to which directory the file is extracted:

<b>ExtractOptions</b>	<b>Name in Archive</b>	<b>Directory for Extracted File</b>
[ ]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
[eoCreateDirs, eoRestorePath]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY\DOC, which is created if necessary.
[eoCreateDirs]	README.TXT	BASEDIRECTORY, which is created if necessary.
<hr/>		
<b>ExtractOptions</b>	<b>Name in archive</b>	<b>Directory for extracted file</b>
	DOC/README.TXT	BASEDIRECTORY, which is created if necessary.
[eoRestorePath]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY\DOC, if it exists. Otherwise, an exception is raised.

See also: [ExtractFiles](#), [ExtractTaggedItems](#), [OnConfirmOverwrite](#), [TAbCabBrowser.BaseDirectory](#)

## **ExtractTaggedItems**

**method**

`procedure ExtractTaggedItems;`

↳ Extracts all tagged files from the cabinet.

`ExtractTaggedItems` extracts all files that have their Tagged property set to True.

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions for a description of how the directory is determined.

See also: [ExtractFiles](#), [ExtractOptions](#), [BaseDirectory](#), [OnConfirmProcessItem](#), [TagItems](#)

## OnConfirmOverwrite

event

```
property OnConfirmOverwrite : TAbConfirmOverwriteEvent  
  TAbConfirmOverwriteEvent = procedure(  
    var FileName : string; var Confirm : Boolean) of object;
```

- ↳ Defines an event handler that is called if the output file for an extract operation already exists.

This event handler is called prior to the start of the extract process if the expected output file already exists. You can specify a different file name if desired, or set Confirm to False and abort the extract process for that item. If you do not provide a handler for this event, the file is overwritten without warning.

See also: ExtractFiles, ExtractTaggedItems

---

# TAbCustomMakeCab Class

TAbCustomMakeCab class is the immediate ancestor of the TAbMakeCab component. It implements all of the methods and properties used by the TAbMakeCab component and is identical to the TAbMakeCab component except that no properties are published.

TAbCustomMakeCab is provided to facilitate creation of descendent cabinet archive building components. For property and method descriptions, see “TAbMakeCab Component” on page119.

## Hierarchy

TComponent

6

① TAbBaseComponent (AbBase) .....	54
② TABaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbCBrows).....	108
TAbCustomMakeCab (AbCabMak)	

# TabMakeCab Component

TabMakeCab is a non-visual component that extends the capabilities of the TAbCabBrowser by adding the ability to create a cabinet archive and add files to it.

## Hierarchy

### TComponent

① TAbBaseComponent (AbBase) .....	54
② TAbBaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbCBrows).....	108
TAbCustomMakeCab (AbCabMak) .....	118
TAbMakeCab (AbCabMak)	

6

## Properties

② ArchiveProgressMeter	FolderThreshold	② SpanningThreshold
② BaseDirectory	② ItemProgressMeter	② Status
CompressionType	②LogFile	StoreOptions
② Count	② Logging	② TempDirectory
② FileName	LZXWindowSize	① Version

## Methods

AddFiles	② FindFile	② OpenArchive
AddFilesEx	② FindItem	② TagItems
② ClearTags	NewCabinet	② UnTagItems
② CloseArchive	NewFolder	

## Events

② OnArchiveItemProgress	② OnConfirmProcessItem	② OnRequestImage
② OnArchiveProgress	② OnLoad	OnSave
② OnChange	② OnProcessItemFailure	

# Reference Section

## AddFiles method

```
procedure AddFiles(const FileMask : string; SearchAttr : Integer);
```

↳ Adds files that match FileMask to a cabinet under construction.

The wild card characters '\*' and '?' are allowed in the FileMask. When AddFiles is searching for files that match FileMask, it searches certain directories, depending on the StoreOptions and BaseDirectory properties. The following table shows, for a given FileMask and StoreOptions, which directories are searched for the file. If the file is found, it is stored in the archive with the name shown in the last column.

6

StoreOptions	FileMask	Directories Searched	Name in Archive
[ ]	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
[ soRecurse, soStripPath ]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT
	X:\DOC\README.TXT	X:\DOC	README.TXT
[ soRecurse ]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT or SUBDIR/README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
[ soStripPath ]	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TEXT
	\DOC\README.TEXT	DEFAULTDRIVE:\DOC	README.TEXT
	X:\DOC\README.TEXT	X:\DOC	README.TEXT

Use the SearchAttr parameter if you want to include special files, such as system and hidden files in the cabinet. If SearchAttr is 0, only normal files that match FileMask are added. If SearchAttr is set to one of the file attribute constants: faReadOnly, faHidden, faSysFile, faVolumeID, faDirectory, faArchive, defined in the SysUtils unit, the appropriate type of special files are also added. To add multiple types of special files, add the file attribute constants together. For example, to search for read-only and hidden files in addition to normal files, pass (faReadOnly + faHidden) as the SearchAttr parameter.

See also: StoreOptions, BaseDirectory

### AddFilesEx

method

```
procedure AddFilesEx(  
  const FileMask, ExclusionMask : string; SearchAttr : Integer);
```

↳ Add files matching FileMask but not ExclusionMask.

AddFilesEx introduces item exception list functionality to AddFiles. By specifying ExclusionMask you can exclude certain files that would normally be added using the AddFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could add all files except text files using the following code:

```
AddFilesEx('*.*', '*.txt');
```

See also: AddFiles

### CompressionType

property

```
property CompressionType : TAbCabCompressionType  
TAbCabCompressionType = (ctNone, ctMSZIP, ctLZX);
```

Default: ctMSZIP

↳ Specifies the type of compression used when adding files to a cabinet.

CompressionType indicates which type of compression will be used when compressing a cabinet folder. CompressionType can vary from one folder to the next, unless the folder is a continuation from a previous cabinet.

By setting CompressionType, you can specify which type of compression to use during subsequent add operations. Setting CompressionType to ctNone specifies that files will be stored with no compression.

**FolderThreshold****property**

```
property FolderThreshold : Longint
```

Default: MaxLongint

☞ The maximum cabinet folder (compression block) size.

Items in a cabinet archive can be compressed across their file boundaries into a single compression block. Such compression blocks are called folders. Compression ratios improve significantly when items are compressed together as opposed to individually. However, there is a trade-off between random access time to an individual item and compression ratio since an entire folder must be decompressed to extract an item from it.

Use FolderThreshold to specify the maximum size of a folder (compression block). When compressing files, this value will be used to determine when to stop the current compression block and start a new one. You can also start a new folder by calling NewFolder.

See also: NewFolder

**LZXWindowSize****property**

```
property LZXWindowSize : Integer;
```

☞ Sets the window size for the LZX compression engine.

Default: 18

The LZX window size must be a power of 2, from  $2^{15}$  to  $2^{21}$ . The greater the window size, the greater the compression. This property sets the exponent to which the base (2) will be raised and must be between 15 and 21. The following table illustrates the effect of this property setting on the LZX window size:

<b>LZXWindowSize Property Setting</b>	<b>LZX Window Size</b>
15	32K
16	64K
17	128K
18	256K
19	512K
20	1MB
21	2MB

See also: CompressionType

## NewCabinet

method

```
procedure NewCabinet;
```

- ☞ Flush the current cabinet to disk and start a new one.

Use NewCabinet to force the current cabinet under construction to be written to disk and start a new one. This is done automatically when the cabinet size reaches the value specified by SpanningThreshold. NewCabinet will fire the OnRequestImage event to obtain the new cabinet file name. If an event handler is not defined for OnRequestImage, the new cabinet file name will be auto-generated. See OnRequestImage for more information about auto-generated file names.

See also: TAbBaseBrowser.OnRequestImage, TAbBaseBrowser.SpanningThreshold

## NewFolder

method

```
procedure NewFolder;
```

- ☞ Flush the current folder and start a new one.

Use NewFolder to force the current folder (compression block) to be completed and start a new one. The compression history is reset and a new compression block is started using the compression type specified by CompressionType. This is done automatically when the folder size reached the value specified by FolderThreshold.

See also: CompressionType, FolderThreshold

## OnSave

event

```
property OnSave : TAbArchiveEvent
```

```
TAbArchiveEvent = procedure(Sender : TObject) of object;
```

- ☞ Defines an event handler that is called immediately after the cabinet is saved.

You can use the OnSave event along with the OnConfirmProcessItem event to display status information for the user. For example, in the OnConfirmProcessItem event handler, you could set the status to “change pending”, and in the OnSave event handler, set the status to “saved”. This would be most useful when the AutoSave property is False.

See also: OnConfirmProcessItem, OnConfirmSave, Save

## StoreOptions property

---

```
property StoreOptions : TAbStoreOptions  
  
TAbStoreOption = (  
  soStripDrive, soStripPath, soRemoveDots, soRecurse)  
  
TAbStoreOptions = set of TAbStoreOption;  
  
Default: [soStripDrive, soRemoveDots]
```

↳ Determines the options for archive add and freshen operations.

The following table lists the results of the available options:

6

Option	Result
soStripDrive	Drive letter information is removed from the stored file name.
soStripPath	All path information is removed from the stored file name.
soRemoveDots	All relative path information is removed from the stored file name. For example, if you call AddFiles with a FileMask of "..\TEST.TXT", the parent of the current BaseDirectory is searched for a file named "TEST.TXT". If the file is found, it is stored as "TEST.TXT".
soRecurse	Subdirectories of the search path are included in the search for files to add or freshen.

See also: [AddFiles](#), [FreshenFiles](#), [FreshenTaggedItems](#)

---

## TAbCustomCabKit Class

TAbCustomCabKit class is the immediate ancestor of the TAbCabKit component. It implements all of the methods and properties used by the TAbCabKit component and is identical to the TAbCabKit component except that no properties are published.

TAbCustomCabKit is provided to facilitate creation of descendent cabinet archive building components. For property and method lists, see “TAbCabKit Component” on page 126.

### Hierarchy

#### TComponent

❶ TAbBaseComponent (AbBase) .....	54
❷ TAbBaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbCBrows).....	108
TAbCustomMakeCab (AbCabMak) .....	118
TAbCustomCabKit (AbCabKit)	

# TAbCabKit Component

TAbCabKit is a non-visual component that extends the capabilities of the TAbMakeCab by adding the ability to open an existing cabinet archive and extract files from it.

It is important to keep in mind that a cabinet archive can be opened as write-only, or read-only, but not both. Opening a new cabinet archive opens it as write-only and items can be added. Opening an existing cabinet archive opens it as read-only and items can be extracted.

## Hierarchy

### TComponent

6

① TAbBaseComponent (AbBase) .....	54
② TABBaseBrowser (AbBrowse) .....	56
TAbCustomCabBrowser (AbC brows).....	108
TAbCustomMakeCab (AbCabMak) .....	118
TAbCustomCabKit (AbCabKit) .....	125
TAbCabKit (AbCabKit)	

## Properties

② ArchiveProgressMeter	ExtractOptions	② Status
② BaseDirectory	② ItemProgressMeter	② TempDirectory
② CompressionType	②LogFile	① Version
② Count	② Logging	
② FileName	② SpanningThreshold	

## Methods

- |                |                    |               |
|----------------|--------------------|---------------|
| ② ClearTags    | ExtractAt          | ② OpenArchive |
| ② CloseArchive | ExtractFiles       | ② TagItems    |
| ② FindFile     | ExtractFilesEx     | ② UnTagItems  |
| ② FindItem     | ExtractTaggedItems |               |

## Events

- |                         |                        |                        |
|-------------------------|------------------------|------------------------|
| ② OnArchiveItemProgress | ② OnConfirmProcessItem | ② OnProcessItemFailure |
| ② OnArchiveProgress     | OnConfirmOverwrite     | ② OnRequestImage       |
| ② OnChange              | ② OnLoad               |                        |

# Reference Section

## ExtractAt method

```
procedure ExtractAt(Index : Integer; NewName : string);
```

↳ Extracts an item with a known item index.

Index is the zero-based index of the item. NewName is the file name to be given to the item if different than the stored file name. If NewName is left blank then the item's stored file name will be used.

## ExtractFiles method

6

```
procedure ExtractFiles(const FileMask : string);
```

↳ Extracts all files that match FileMask from the cabinet.

Each file in the cabinet is compared to FileMask. If a cabinet item's stored file name matches FileMask, the file is extracted. If no path information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” extracts all files in the archive that have a TXT extension. If directory information is included in FileMask, the directory information in the cabinet item's file name must match for the file to be extracted.

The following table shows some example FileMasks and stored file names and indicates whether they would be considered a match:

FileMask	Stored File Name	Match?
*.txt	test.txt	Yes
*.txt	doc/test.txt	Yes
*.txt	bin/test.txt	Yes
doc/*.txt	test.txt	No
doc/*.txt	doc/test.txt	Yes
doc/*.txt	bin/test.txt	No
*/*.txt	test.txt	No
*/*.txt	doc/test.txt	Yes
*/*.txt	bin/test.txt	Yes

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions for a description of how the directory is determined.

See also: ExtractFilesEx, ExtractOptions, ExtractTaggedItems, BaseDirectory, OnConfirmProcessItem, OnProcessItemFailure

### ExtractFilesEx

method

```
procedure ExtractFilesEx(const FileMask, ExclusionMask : string);
```

↳ Extracts all files that match FileMask but not ExclusionMask.

ExtractFilesEx introduces item exception list functionality to ExtractFiles. By specifying ExclusionMask you can exclude certain files that would normally be extracted using the ExtractFiles method.

For example, you could extract all files except text files using the following code:

```
ExtractFilesEx('*.*', '*.txt');
```

See also: ExtractFiles

### ExtractOptions

property

```
property ExtractOptions : TAbExtractOptions  
TAbExtractOption = (eoCreateDirs, eoRestorePath);  
TAbExtractOptions = set of TAbExtractOption;
```

Default: [eoCreateDirs]

↳ Determines the options for archive extract operations.

If ExtractOptions contains eoCreateDirs, directories (including multiple levels of subdirectories) are created as needed to extract files.

If ExtractOptions contains eoRestorePath, path information stored in the archive is retained when files are extracted. The extracted file is placed in the appropriate subdirectory relative to the current BaseDirectory.

The following table shows, for a given file in the cabinet and a set of ExtractOptions, to which directory the file is extracted:

<b>ExtractOptions</b>	<b>Name in Archive</b>	<b>Directory for Extracted File</b>
[ ]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
[eoCreateDirs, eoRestorePath]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY\DOC, which is created if necessary.
[eoCreateDirs]	README.TXT	BASEDIRECTORY, which is created if necessary.
<hr/>		
ExtractOptions	Name in archive	Directory for extracted file
	DOC/README.TXT	BASEDIRECTORY, which is created if necessary.
[eoRestorePath]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY\DOC, if it exists. Otherwise, an exception is raised.

See also: [ExtractFiles](#), [ExtractTaggedItems](#), [OnConfirmOverwrite](#), [TAbCabBrowser.BaseDirectory](#)

## [ExtractTaggedItems](#)

**method**

`procedure ExtractTaggedItems;`

↳ Extracts all tagged files from the cabinet.

`ExtractTaggedItems` extracts all files that have their Tagged property set to True.

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions for a description of how the directory is determined.

See also: [ExtractFiles](#), [ExtractOptions](#), [BaseDirectory](#), [OnConfirmProcessItem](#), [TagItems](#)

## OnConfirmOverwrite

event

```
property OnConfirmOverwrite : TAbConfirmOverwriteEvent  
  TAbConfirmOverwriteEvent = procedure(  
    var FileName : string; var Confirm : Boolean) of object;
```

- ↳ Defines an event handler that is called if the output file for an extract operation already exists.

This event handler is called prior to the start of the extract process if the expected output file already exists. You can specify a different file name if desired, or set Confirm to False and abort the extract process for that item. If you do not provide a handler for this event, the file is overwritten without warning.

See also: ExtractFiles, ExtractTaggedItems



# Chapter 7: TAbZipOutline Visual Component

---

Abbrevia includes a visual outline component, TAbZipOutline, which combines the archive access capabilities of a TAbZipKit with the display capabilities of the TTreeView component.

TAbZipOutline contains a reference to a TABArchive, which encapsulates the logical organization of an archive. Through the TABArchive, the TAbZipOutline can view and modify the contents of an archive (either PKZIP compatible, or a GZip or Tar archive). TAbZipOutline also contains a reference to a specialized descendant of the TTreeView component. This combination allows the TAbZipOutline to provide the power of the non-visual TAbZipKit component to manipulate an archive and the ease-of-use of a visual display of the archive.

There are several ways to customize the display of the TAbZipOutline. Any of the standard visual options that are available in the TTreeView component (e.g., Color, BorderStyle, Font, ShowButtons) can be altered. In addition, TAbZipOutline adds properties that can further modify the visual display: Attributes, Hierarchy, Options, and OutlineStyle.

The Attributes property determines which attributes to display about any item in an archive. You can turn the attributes on or off as desired and see the result in the outline. The attributes are described in the Attributes property on page 140.

The Hierarchy property determines whether the archive contents are displayed in a directory hierarchy or the directory information for each item is displayed as part of the file name. You can toggle the Hierarchy property to see how each display looks.

The Options property determines how the nodes in the ZipOutline are drawn: whether the root node is visible, whether a focus rectangle is displayed, and whether the icons used in display are stretched to fit the current text size. The Style property determines how the tree hierarchy is drawn: text only, with or without icons, with or without lines indicating the tree layout.

---

# TAbCustomZipOutline Class

TAbCustomZipOutline class is the immediate ancestor of the TAbZipOutline component. It implements all of the methods and properties used by TAbZipOutline except no properties are published.

TAbCustomZipOutline is provided to facilitate creation of descendent zip outline components. For property and method descriptions, see “TAbZipOutline Component” on page 135.

## Hierarchy

TWinControl (Windows); TWidgetControl (Linux)

TAbCustomZipOutline (AbZipOut)

---

# TAbZipOutline Component

The TAbZipOutline component is a visual component that allows read/write access to a PKZIP- compatible file, including full support for adding and extracting files.

TAbZipOutline displays the contents of the archive in a tree view format. It contains a descendant of the VCL TTreeView component, which is used to provide the display. Several of the properties, events, and methods of TAbZipOutline are simply the surfaced properties of the contained TTreeView. Such properties, events, and methods are documented in this section only if they are critical to the use of TAbZipOutline or if they have been modified.

## Hierarchy

TWinControl (Windows); TWidgetControl (Linux)

TAbCustomZipOutline (AbZipOut).....	134
TAbZipOutline (AbZipOut)	

## Properties

ArchiveProgressMeter	Hierarchy	PictureZipAttribute
Attributes	Items	SelectedZipItem
AutoSave	ItemProgressMeter	SpanningThreshold
BaseDirectory	LogFile	Status
CompressionMethodToUse	Logging	StoreOptions
Count	Options	Style
DeflationOption	Password	TempDirectory
DOSMode	PasswordRetries	Version
FileName	PictureFile	ZipFileComment

## Methods

AddFiles	ExtractAt	FreshenTaggedItems
AddFilesEx	ExtractFiles	Move
AddFromStream	ExtractFilesEx	Replace
ClearTags	ExtractTaggedItems	Save
CloseArchive	ExtractToStream	TagItems
DeleteAt	FindFile	TestTaggedItems
DeleteFiles	FindItem	UnTagItems
DeleteFilesEx	FreshenFiles	
DeleteTaggedItems	FreshenFilesEx	

## Events

7

OnArchiveItemProgress	OnLoad	OnRequestLastDisk
OnArchiveProgress	OnMouseWheel	OnRequestNthDisk
OnChange	OnNeedPassword	OnSave
OnConfirmOverwrite	OnProcessItemFailure	OnWindowsDrop
OnConfirmProcessItem	OnRequestBlankDisk	
OnConfirmSave	OnRequestImage	

# Reference Section

## AddFiles

method

```
procedure AddFiles(const FileMask : string; SearchAttr : Integer);
```

↳ Adds the files that match FileMask to the archive.

The wild card characters '\*' and '?' are allowed in the FileMask.

When AddFiles is searching for files that match FileMask, it searches certain directories, depending on the StoreOptions and BaseDirectory properties. The following table shows, for a given FileMask and set of StoreOptions, which directories are searched for the file. If the file is found, it is stored in the archive with the name shown in the last column.

StoreOptions	FileMask	Directories Searched	Name in Archive
[ ]	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
[ soRecurse, soStripPath ]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT
	X:\DOC\README.TXT	X:\DOC	README.TXT
[ soRecurse ]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT or SUBDIR/README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
[ soStripPath ]	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT
	X:\DOC\README.TXT	X:\DOC	README.TXT

The following table shows Linux FileMasks:

<b>StoreOptions</b>	<b>Linux FileMask</b>	<b>Directories Searched</b>	<b>Name in Archive</b>
[ ]	README .TXT	BASEDIRECTORY	README .TXT
	DOC / README .TXT	BASEDIRECTORY / DOC	DOC / README .TXT
	/DOC / README .TXT	/DOC	DOC / README .TXT
[ soRecurse, soStripPath ]	README .TXT	BASEDIRECTORY and all subdirectories	README .TXT
	DOC / README .TXT	BASEDIRECTORY / DOC	README .TXT
	/DOC / README .TXT	/DOC	README .TXT
[ soRecurse ]	README .TXT	BASEDIRECTORY and all subdirectories	README .TXT or SUBDIR / README .TXT
	DOC / README .TXT	BASEDIRECTORY \ DOC	DOC / README .TXT
	/DOC / README .TXT	/DOC	DOC / README .TXT
[ soStripPath ]	README .TXT	BASEDIRECTORY	README .TXT
	DOC / README .TXT	BASEDIRECTORY / DOC	README .TXT
	/DOC / README .TXT	/DOC	README .TXT

The SearchAttr parameter allows searching for particular classes of file system entities and varies by platform. If SearchAttr is zero, only normal files that match FileMask are added. If SearchAttr is set to one of the file attribute constants defined in the SysUtils unit, the appropriate type of special files are also added. To add multiple types of special files, add the file attribute constants together.

## Windows

The options are faReadOnly, faHidden, faSysFile, faVolumeID, faDirectory, and faArchive. Example: to search for read-only and hidden files in addition to normal files, pass (faReadOnly + faHidden) as the SearchAttr parameter.

## Linux

The options are faLinuxDir, faFileLink, faChrFile, faBlkFile, faFifoFile, and faSockFile. Example: to search for directories and symbolic links in addition to normal files, pass (faLinuxDir + faFileLink) as the SearchAttr parameter.

See the OnProcessItemFailure event on page 159 for a description of the exceptions that can occur during the add process.

See also: BaseDirectory, OnChange, StoreOptions

## AddFilesEx

method

```
procedure AddFilesEx(  
  const FileMask, ExclusionMask : string; SearchAttr : Integer);
```

↳ Adds files matching FileMask but not ExclusionMask.

AddFilesEx introduces item exception list functionality to AddFiles. By specifying ExclusionMask you can exclude certain files that would normally be added using the AddFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

### Windows

```
AddFilesEx('*.*', '*.txt', 0);
```

### Linux

```
AddFilesEx('*', '*.txt', 0);
```

See also: AddFiles

## AddFromStream

method

```
procedure AddFromStream(  
  const NewName : string; FromStream : TStream)
```

↳ Create a zip item directly from a stream.

AddFromStream will create a zip item by compressing data directly from a TStream instance. A file name must be provided via NewName for the new zip item that is created. The stream's position is set to 0 and the entire stream is added. Adding data directly from a stream causes the archive to be automatically saved since the stream object may be subsequently freed or modified. The item's properties such as CompressedSize, UnCompressedSize, CRC32, etc, are set accordingly.

## ArchiveProgressMeter

## property

property ArchiveProgressMeter : TAbMeter

- ↳ Specifies a synchronized meter for the OnArchiveProgressEvent.

The ArchiveProgressMeter property provides a mechanism to associate a TAbMeter (see page 290) with the OnArchiveProgress event. If a meter is specified then it will be updated to reflect the progress given by the OnArchiveProgress event. This occurs even if no event handler has been assigned to the event.

See also: OnArchiveProgress, ItemProgressMeter

## Attributes

## property

property Attributes : TAbZipAttributes

TAbZipAttribute = (zaCompressedSize, zaCompressionMethod,  
zaCompressionRatio, zaCRC, zaExternalFileAttributes,  
zaInternalFileAttributes, zaEncryption, zaTimeStamp,  
zaUncompressedSize, zaVersionMade, zaVersionNeeded,  
zaComment);

TAbZipAttributes = set of TAbZipAttribute;

Default: [zaCompressedSize, zaCompressionMethod, zaCompressionRatio, zaCRC,  
zaExternalFileAttributes, zaEncryption, zaTimeStamp, zaUncompressedSize]

- ↳ Determines which attributes are displayed in the outline.

PKZIP-compatible archives contain extensive information about each archive item. The following information can be displayed in the outline:

Attribute	Description
zaCompressedSize	Size of the file before it was compressed.
zaCompressionMethod	The possible values are stored, shrunk, reduced, imploded, and deflated.
zaCompressionRatio	(uncompressed size - compressed size) * 100 / uncompressed size.
zaCRC	A 32-bit Cyclical Redundancy Check for the uncompressed file.
zaExternalFileAttributes	Indicates whether the file has one of the system file attributes (i.e., system, read-only, archive, hidden).
zaInternalFileAttributes	Indicates whether the file is text or binary.

<b>Attribute</b>	<b>Description</b>
zaEncryption	Indicates whether the file is encrypted or not.
zaTimeStamp	Date and time that the uncompressed file was last changed.
zaUncompressedSize	Size of the file after it was compressed.
zaVersionMade	Version of the compression program used to compress the file (PKZIP compatibility version). Abbrevia stores 2.0 here to indicate that the file was compressed with a version of Abbrevia that is compatible with PKZIP version 2.0.
zaVersionNeeded	Minimum version of the compression program that is required to extract the file. This is a PKZIP compatibility version. For example, a 2.0 here indicates that a version of Abbrevia that is compatible with PKZIP version 2.0 is required to extract the file.
zaComment	A comment for the file.

The following example displays only the Compression Ratio and CRC of each file:

```
AbZipOutline1.Attributes := [zaCompressionRatio, zaCRC];
```

See also: [Hierarchy](#)

### AutoSave property

property AutoSave : Boolean

Default: False

↳ Controls whether changes to the archive are performed immediately or not.

When AutoSave is True, each change to the archive forces the disk image of the archive to be updated immediately. When AutoSave is False, changes to the disk image are done when one of the following occur:

- The archive is explicitly saved using the Save method.
- Multiple operations on a single item require the archive to be saved.
- FileName is changed, causing a new archive to be opened.

**Note:** When AutoSave is True, only a single archive file may be created. Multiple archive file spanning requires that AutoSave is False. Also, keep in mind that saving the archive each time a file is added can be very time consuming. For these reasons, we recommend that AutoSave be used only with relatively small archives.

See also: [FileName](#), [OnConfirmSave](#), [Save](#)

---

**BaseDirectory** **property**

**property** BaseDirectory : string

↳ Specifies the default path for add and extract operations on the archive.

BaseDirectory is not necessarily the same as the directory of the archive.

When you add a file to an archive, if FileName is a relative file specification (e.g., "README.TXT" or "EXAMPLES\README.TXT"), BaseDirectory is used as the starting point to search for the file. When you extract a file from an archive, it is extracted to the current BaseDirectory or a subdirectory of the current BaseDirectory.

See the [AddFiles](#), [FreshenFiles](#), [FreshenTaggedItems](#), [ExtractFiles](#), and [ExtractTaggedItems](#) methods for information on how BaseDirectory is used in each case.

---

**ClearTags** **method**

**procedure** ClearTags;

↳ Untags all items in the archive.

See also: [TagItems](#), [UnTagItems](#)

---

**CloseArchive** **method**

**procedure** CloseArchive;

↳ Closes the archive.

Use CloseArchive to close the current archive. If necessary, the current archive will be saved. The FileName property will be set to an empty string.

See also: [FileName](#)

## CompressionMethodToUse

property

```
property CompressionMethodToUse : TAbZipSupportedMethod  
TAbZipSupportedMethod = (smStored, smDeflated, smBestMethod);  
Default: smBestMethod
```

↳ Selects the compression method.

CompressionMethodToUse selects the compression method used when adding or freshening items in the archive. The possible values for CompressionMethodToUse are:

Value	Description
smStored	The file is stored without any compression.
smDeflated	The file is compressed using the deflate method.
smBestMethod	The file is deflated, but if the resulting file is larger than the original, the file is simply stored.

See also: AddFiles

## Count

run-time, read-only property

```
property Count : Integer
```

↳ Returns the number of items in the current archive.

Count is the number of items available from the Items property.

☞ **Caution:** Count returns the number of items in the memory representation of the archive. If AutoSave is False, and items have been added or deleted from the archive, the number of items in the memory representation can be different than the number of items in the disk file or stream.

The following example illustrates how to iterate through the items of a TAbZipOutline:

```
var
  I : Integer;
  TotalSize : LongInt;
begin
  TotalSize := 0;
  with AbZipOutline1 do
    if Count > 0 then
      for I := 0 to Pred(Count) do
        TotalSize := TotalSize + Items[I].CompressedSize;
end;
```

See also: AutoSave, Items

## DeflationOption

## property

---

property DeflationOption : TAbZipDeflationOption

7

TAbZipDeflationOption = (doInvalid, doNormal, doMaximum, doFast, doSuperFast);

Default: doNormal

↳ Determines whether priority is given to compression or speed in the deflation.

DeflationOption allows you to select the deflation option when CompressionMethodToUse is smDeflated or smBestMethod. You can select varying trade-offs between compression and speed. The choices are Maximum (most compression, slowest speed), Normal, Fast, and Super Fast (least compression, fastest speed). The choices correspond to the PKZIP 2.04g -ex, -en, -ef, and -es options.

See also: CompressionMethodToUse

## DeleteAt

## method

---

procedure DeleteAt(Index : Integer);

↳ Deletes the item at the specified item index.

DeleteAt can be used to delete an item from an archive when its index is known. Index is the zero-based item index of the item to be deleted.

## DeleteFiles

method

```
procedure DeleteFiles(const FileMask : string);
```

- ↳ Deletes the specified files from the archive.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is marked to be deleted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” marks all files in the archive that have a TXT extension to be deleted. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be deleted.

The next time the archive is saved, all marked files and their corresponding information are deleted from the archive.

See also: DeleteTaggedItems, OnChange, Save

## DeleteFilesEx

method

```
procedure DeleteFilesEx(const FileMask, ExclusionMask : string);
```

- ↳ Delete files that match FileMask but not ExclusionMask.

DeleteFilesEx introduces item exception list functionality to DeleteFiles. By specifying ExclusionMask you can exclude certain files that would normally be deleted using the DeleteFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could delete all files from the archive except text files using the following code:

```
DeleteFilesEx('*.*', '*.txt');
```

## DeleteTaggedItems

method

```
procedure DeleteTaggedItems;
```

- ↳ Finds all files that have their Tagged property set to True and marks them to be deleted.

The next time the archive is saved, all marked files and their corresponding information are deleted from the archive.

See also: ClearTags, DeleteFiles, OnChange, Save, TagItems, UnTagItems

## DOSMode

## property

property DOSMode : Boolean

Default: False

- ↳ Forces all new files stored in an archive to have DOS-compatible file names.

Windows 95 and Windows NT introduced long file names, which were not available under older versions of Windows or DOS. The older operating systems cannot handle the long file names; they require a file name no longer than eight characters (8.3 format). If your archive must be read by an older operating system, you can force short file names by setting DOSMode to True.

### Windows

The shortened version of the file name is determined by the Windows GetShortPathName API call.

7

If DOSMode is True, a file named C:\PROGRAM FILES\README.TXT is stored in the archive as PROGRA~1/README.TXT.

### Linux

The shortened version of the file name is determined by a simple algorithm similar to what is used on Windows to generate short file names.

If DOSMode is True, a file named /home/mydir/longdirname/SomeLongFileName will be stored in the archive as something like home/mydir/longd~01/SomeL~01.

## ExtractAt

## method

procedure ExtractAt(Index : Integer; NewName : string);

- ↳ Extracts an item with a known item index.

ExtractAt can be used to extract an item from a zip archive when its item index is known. Index is the zero-based item index. NewName is the file name to be given to the item, once extracted, if different than the stored file name. If NewName is the empty string then the item's stored file name will be used.

```
procedure ExtractFiles(const FileMask : string);
```

↳ Extracts the specified files from the archive.

ExtractFiles extracts all files that match FileMask. The files in the archive are not modified by extract operations. The extraction happens immediately—it is not affected by the setting of the AutoSave property.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is extracted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” extracts all files in the archive that have a TXT extension. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be extracted.

The following table shows example FileMasks and stored file names, and indicates whether they would be considered a match:

FileMask	Stored File Name	Match?
*.txt	test.txt	Yes
*.txt	doc/test.txt	Yes
*.txt	bin/test.txt	Yes
doc/*.txt	test.txt	No
doc/*.txt	doc/test.txt	Yes
doc/*.txt	bin/test.txt	No
*/*.txt	test.txt	No
*/*.txt	doc/test.txt	Yes
*/*.txt	bin/test.txt	Yes

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties.

See the OnProcessItemFailure event on page 159 for a description of the exceptions that can occur during the extract process.

See also: BaseDirectory, ExtractOptions, ExtractTaggedItems, OnConfirmProcessItem, OnProcessItemFailure

## ExtractFilesEx

method

```
procedure ExtractFilesEx(const FileMask, ExclusionMask : string);
```

- ↳ Extracts files that match FileMask but not ExclusionMask.

ExtractFilesEx introduces item exception list functionality to ExtractFiles. By Specifying ExclusionMask you can exclude certain files that would normally be extracted using the ExtractFiles method.

For example, you could extract all files except text files using the following code:

### Windows

```
ExtractFilesEx('*.*', '*.txt');
```

### Linux

```
ExtractFilesEx('*', '*.txt');
```

See also: ExtractFiles

7

## ExtractOptions

property

```
property ExtractOptions : TAbExtractOptions  
TAbExtractOption = (eoCreateDirs, eoRestorePath);  
TAbExtractOptions = set of TAbExtractOption;
```

Default: [eoCreateDirs]

- ↳ Determines the options for archive extract operations.

If ExtractOptions contains eoCreateDirs, directories (including multiple levels of subdirectories) are created as needed to extract files.

If ExtractOptions contains eoRestorePath, path information stored in the archive is retained when files are extracted. The extracted file is placed in the appropriate subdirectory relative to the current BaseDirectory.

The following table shows, for a given file in the archive and a set of ExtractOptions, to which directory the file is extracted:

<b>ExtractOptions</b>	<b>Name in Archive</b>	<b>Directory for Extracted File</b>
[ ]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
[ eoCreateDirs, eoRestorePath]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), which is created if necessary.
[ eoCreateDirs]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY, which is created if necessary.
[ eoRestorePath]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), if it exists. Otherwise, an exception is raised.

See the OnProcessItemFailure event on page 159 for a description of how exceptions that occur during the extract process are handled.

See also: BaseDirectory, ExtractFiles, OnConfirmOverwrite

## ExtractTaggedItems

## **method**

`procedure ExtractTaggedItems;`

↳ Extracts all tagged files from the archive.

ExtractTaggedItems extracts all files that have their Tagged property set to True. The files in the archive are not modified by extract operations. The extraction happens immediately—it is not affected by the setting of the AutoSave property.

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties.

See the OnProcessItemFailure event on page 159 for a description of the exceptions that can occur during the extract process.

See also: BaseDirectory, ExtractFiles, ExtractOptions, OnConfirmProcessItem, OnProcessItemFailure, TagItems

---

<b>ExtractToStream</b>	<b>method</b>
------------------------	---------------

---

```
procedure ExtractToStream(
  const FileName : string; ToStream : TStream);
```

↳ Extracts the specified item directly to a stream.

ExtractToStream will extract an item directly to a TStream instance. The instance must already be created. The stream instance is not cleared and the extracted file is appended at the stream's current position.

---

<b>FileName</b>	<b>property</b>
-----------------	-----------------

---

7

```
property FileName : string
```

↳ Returns the name of the archive.

If you change FileName, the current archive is saved, if necessary, and closed. If FileName is not blank, a new archive is opened and loaded, initializing the Count and Items properties. If the specified file is not a PKZIP-compatible archive, an EAbZipInvalid exception is raised.

See also: BaseDirectory, Count, Items

---

<b>FindFile</b>	<b>method</b>
-----------------	---------------

---

```
function FindFile(const aFileName : string) : Integer;
```

↳ Returns the index of the specified file in the archive's item list.

FindFile searches through the list of archive files until it finds one with an exact match for the specified file name. It then returns the index of that item. FindFile returns -1 if no match is found. No two items in an archive can have the same file name.

---

<b>FindItem</b>	<b>method</b>
-----------------	---------------

---

```
function FindItem(aItem : TABArchiveItem) : Integer;
```

↳ Returns the index of the specified archive item.

FindItem searches through the list of archive items until it finds the one specified. It then returns the index of that item. FindItem returns -1 if no match is found.

```
procedure FreshenFiles(const FileMask : string);
```

↳ Freshens the specified files in the archive.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is marked to be freshened. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” marks all files in the archive that have a TXT extension to be freshened. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be freshened.

The next time the archive is saved, the marked files are freshened. If the file is found on disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file.

The directories that are searched to find the file depend on the StoreOptions and BaseDirectory properties. The following table shows, for a given file in the archive and a set of StoreOptions, what Windows directories are searched to find the file to use to freshen the file in the archive:

StoreOptions	Name in Archive	Directories Searched
[ ]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY\DOC
[ soRecurse, soStripPath ]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY\DOC
[ soRecurse ]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY\DOC
[ soStripPath ]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY\DOC

The following table shows examples of Linux directories searched:

StoreOptions	Name in Archive	Directories Searched
[ ]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY/DOC
[soRecurse, soStripPath]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY/DOC
[soRecurse]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY/DOC
[soStripPath]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY/DOC

7

See the OnProcessItemFailure event on page 159 for a description of the exceptions that can occur during the freshen process.

See also: FreshenTaggedItems, OnChange, Save, StoreOptions

## **FreshenFilesEx** method

```
procedure FreshenFilesEx(const FileMask, ExclusionMask : string);
```

↳ Freshen files that match FileMask but not ExclusionMask.

FreshenFilesEx introduces item exception list functionality to FreshenFiles. By specifying ExclusionMask you can exclude certain files that would normally be freshened by using the FreshenFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could freshen all files except text files using the following code:

### **Windows**

```
FreshenFilesEx('*.*', '*.txt');
```

### **Linux**

```
FreshenFilesEx('*', *.txt');
```

See also: FreshenFiles

## **FreshenTaggedItems**

**method**

```
procedure FreshenTaggedItems;
```

- ↳ Finds all files that have their Tagged property set to True and marks them to be freshened.

The next time the archive is saved, all marked files are freshened. If the file is found on the disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file.

The directories that are searched to find the file depend on the StoreOptions and BaseDirectory properties. See the table in FreshenFiles for details of determining the directory to search.

See the OnProcessItemFailure event on page 159 for a description of the exceptions that can occur during the freshen process.

See also: FreshenFiles, OnChange, Save, StoreOptions, TagItems

## **Hierarchy**

**property**

```
property Hierarchy : Boolean
```

Default: True

- ↳ Determines whether items are displayed in a directory hierarchy format in the outline.

If Hierarchy is True, the items in the outline are displayed in a directory hierarchy. If Hierarchy is False, the directory information for each item is displayed as part of the file name.

See also: Attributes

## **ItemProgressMeter**

**property**

```
property ItemProgressMeter : TABMeter
```

- ↳ Specifies a synchronized meter for the OnArchiveItemProgress event.

The ItemProgressMeter property provides a mechanism to associate a TABMeter (see page 290) with the OnArchiveItemProgress event. If a meter is specified then it will be updated to reflect the progress given by the OnArchiveItemProgress event. This occurs even if no event handler has been assigned to the event.

See also: ArchiveProgressMeter, OnArchiveItemProgress

## Items

run-time, read-only property

```
property Items[Index : Integer] : TAbZipItem
```

- ↳ Contains file names for each item in the archive.

Each item in a zip archive is described using a TAbZipItem. See the “TAbZipItem Class” on page 242.

Valid values for Index are 0 through Count - 1.

The following example accesses the name of each file in the archive:

```
if (Count > 0) then
  for I := 0 to pred(Count) do
    ListBox1.Items.Add := AbZipOutline1[I].FileName;
```

See also: Count

## LogFile

property

```
property LogFile : string
```

- ↳ Specifies the text file to use for logging.

LogFile specifies the text file that will receive the log entries during archiving operations if logging has been enabled. If the file does not exist, it will be created. If the file does exist, the entries will be appended to the end of the file.

See also: Logging

## Logging

property

```
property Logging : Boolean
```

Default: False

- ↳ Determines whether archive operations are recorded.

When Logging is True, an entry is logged to a text file specified by the LogFile property, for each operation (add, delete, extract, freshen, move, replace) performed on an archive. The following example shows the format of log entries:

```
D:\Test\Test.zip logging 04/27/1999 6:12:06 PM
FDI.H deleted 04/27/1999 6:12:37 PM
Ffdefine.inc added 04/27/1999 6:12:52 PM
```

If LogFile is not set, or the file cannot be used, an exception is raised.

See also: LogFile

```
procedure Move(  
  aItem : TAbArchiveItem; const NewStoredPath : string);
```

↳ Renames an existing archive item.

Move modifies the stored file name of the archive item only in the memory representation of the archive. It marks the file to be moved (renamed). The next time the archive is saved, the stored file name is physically changed in the archive on disk.

See the OnProcessItemFailure event on page 159 for a description of the exceptions that can occur during the move process.

The following example renames the first item in an archive to NEWPATH/NAME.EXT:

```
Move(ABZipper1.Items[0], 'NEWPATH/NAME.EXT');
```

See also: AutoSave, OnChange, Save

## OnArchiveItemProgress

event

```
property OnArchiveItemProgress : TAbArchiveItemProgressEvent  
TAbArchiveItemProgressEvent = procedure(  
  Sender : TObject; Item : TAbArchiveItem; Progress : Byte;  
  var Abort : Boolean) of object;
```

↳ Defines an event handler that is called during long operations on a single archive item.

OnArchiveItemProgress is called by add, extract, and freshen operations on an archive item. You can use it to display the progress of the operation, or to abort the operation. For example, you might want to display the name of the item being processed, the action being performed on the item, and an indication of the progress.

Progress is a percentage that indicates how far the operation has progressed through the archive item. Valid values for Progress are 0 to 100. If you set Abort to True in the event handler, the current operation is aborted.

See also: ItemProgressMeter, OnArchiveProgress

## OnArchiveProgress

event

```
property OnArchiveProgress : TAbArchiveProgressEvent  
TAbArchiveProgressEvent = procedure(Sender : TObject;  
Progress : Byte; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called once per item when a process steps through the archive.

OnArchiveProgress is called during DeleteFiles, FreshenFiles, ExtractFiles, FileName (a change causes a new archive to be loaded), and Save archive operations. You can use it to display the progress of the operation, or even to abort the operation.

Progress is a percentage that indicates how far the current operation has progressed through the entire archive. Valid values for Progress are 0 to 100. If you set Abort to True in the event handler, the current operation is aborted.

See also: ArchiveProgressMeter, DeleteFiles, ExtractFiles, FileName, FreshenFiles, OnArchiveItemProgress, Save

7

## OnChange

event

```
property OnChange : TNotifyEvent
```

- ↳ Defines an event handler that is called when the archive changes.

OnChange is called when the archive is opened or closed, or when the contents of the archive change (immediately after an add, delete, freshen, or move operation).

The OnChange event can be used to update any components that display the contents of the archive or information relating to the archive (e.g., the number of files in the archive).

## OnConfirmOverwrite

event

```
property OnConfirmOverwrite : TAbConfirmOverwriteEvent  
TAbConfirmOverwriteEvent = procedure(  
var FileName : string; var Confirm : Boolean) of object;
```

- ↳ Defines an event handler that is called if the output file for an extract operation already exists.

This event handler is called prior to the start of the extract process if the expected output file already exists. If you set Confirm to False, the extract process is aborted for that item. If Confirm is True or you do not provide a handler for this event, the file is overwritten.

See also: ExtractFiles, ExtractTaggedItems

## OnConfirmProcessItem

event

```
property OnConfirmProcessItem : TAbArchiveItemConfirmEvent  
TAbArchiveItemConfirmEvent = procedure(  
  Sender : TObject; var Item : TAbArchiveItem;  
  ProcessType : TabProcessType; var Confirm : Boolean) of object;  
  
TABProcessType = (  
  ptAdd, ptDelete, ptExtract, ptFreshen, ptMove, ptReplace);
```

- ↳ Defines an event handler that is called before a process is performed on each item in the archive.

This event handler is called once for each item in an add, delete, extract, freshen, or move operation. If you set Confirm to False, the process for that item is aborted. If you do not provide a handler for this event, all processing continues as if Confirm were set to True.

The OnConfirmProcessItem event handler can be used to perform any actions that are necessary when the archive changes. For example, you might want display a confirmation dialog or save information on who modified the archive.

See also: AddFiles, DeleteFiles, DeleteTaggedItems, ExtractFiles, ExtractTaggedItems, FreshenFiles, FreshenTaggedItems, Move, ZipfileComment

## OnConfirmSave

event

```
property OnConfirmSave : TAbArchiveConfirmEvent  
TAbArchiveConfirmEvent = procedure(  
  Sender : TObject; var Confirm : Boolean) of object;
```

- ↳ Defines an event handler that is called before the archive is saved.

This event handler is called immediately before an archive is updated. If you set Confirm to False, the update is aborted. If you do not provide a handler for this event, all processing continues as if Confirm were set to True.

The OnConfirmSave event can be used to allow the user to close an archive without saving changes. This is most useful when the AutoSave property is False. When you change the component's FileName property, the Save method is automatically called, which fires the OnConfirmSave event. If the OnConfirmSave event handler sets Confirm to False, then the archive is not updated (all pending changes are discarded).

See also: AutoSave, Save, FileName

## OnLoad

event

```
property OnLoad : TABArchiveEvent  
TABArchiveEvent = procedure(Sender : TObject) of object;
```

↳ Defines an event handler that is called immediately after an archive's contents are loaded.

If FileName changes (causing a new archive to be loaded), OnLoad is called just after the archive is loaded. An OnLoad event could be used to display the name of the current archive or to update a most-recently-used archive list.

See also: FileName

## OnMouseWheel

event

```
property OnMouseWheel : TMouseWheelEvent  
TMouseWheelEvent = procedure(Sender : TObject;  
      Shift : TShiftState; Delta, XPos, YPos : Word) of object;
```

7

↳ Defines an event handler that is called when the wheel on a Microsoft IntelliMouse is rotated.

TAbZipOutline responds to the mouse wheel (if present and supported by the operating system) by changing the SelectedItem. The SelectedItem moves up if the mouse wheel is rotated away from the user and down if the mouse wheel is rotated toward the user. If the current SelectedItem is not already expanded, rotating the mouse wheel forces expansion before changing the SelectedItem.

Delta is the distance that the wheel is rotated, expressed in multiples or divisions of WHEEL\_DELTA, which is 120. A positive value indicates that the wheel was rotated forward (away from the user). A negative value indicates that the wheel was rotated backward (toward the user). Each notch on the wheel of the Microsoft IntelliMouse corresponds to a rotation of WHEEL\_DELTA.

⌚ **Caution:** This feature is only available on operating systems that support the mouse wheel. It is known to work with Windows NT 4.0 and Windows 2000, and on some installations of Windows 9x (with proper mouse drivers installed). Since the availability of this method is not guaranteed, you should only use OnMouseWheel to provide bonus features, rather than critical operations.

## OnNeedPassword

event

```
property OnNeedPassword : TABNeedPasswordEvent  
  TABNeedPasswordEvent = procedure(  
    Sender : TObject; var NewPassword : string) of object;
```

Defines an event handler that is called to allow entry of a password when decrypting a file.

The OnNeedPassword event handler is called when an encrypted file is being extracted and one of the following occurs:

- The Password property is empty.
- The Password is not valid for the encrypted file and the number of attempts is less than PasswordRetries.

The event causes the Password property to be updated with the value of NewPassword.

See also: Password, PasswordRetries

7

## OnProcessItemFailure

event

```
property OnProcessItemFailure : TABArchiveItemFailureEvent  
  TABArchiveItemFailureEvent = procedure(  
    Sender : TObject; Item : TABArchiveItem;  
    ProcessType : TABProcessType; ErrorClass : TABErrorClass;  
    ErrorCode : Integer) of object;  
  
TABProcessType = (  
  ptAdd, ptDelete, ptExtract, ptFreshen, ptMove, ptReplace);  
  
TABErrorClass = (ecAbbrevia, ecInOutError, ecFilerError,  
  ecFileCreateError, ecFileOpenError, ecOther);
```

Defines an event handler that is called when an exception is raised during an add, extract, freshen, or move operation.

Abbrevia traps all exceptions that occur during add, extract, freshen, and move operations. The exception is translated to an ErrorClass and ErrorCode. The OnProcessItemFailure is called and you can display a customized error message. Exceptions that are defined by Abbrevia are translated to an ErrorClass of ecAbbrevia. Abbrevia's exceptions (defined in the AbExcept unit) all have an ErrorCode property which uniquely identifies them. The ErrorCode constants are defined in the AbConst unit. You can retrieve an error string corresponding to the Abbrevia exception using the global AbStrRes variable. The following example shows how:

```
if ErrorClass = ecAbbrevia then  
  ShowMessage(AbStrRes[ErrorCode]);
```

If an event handler is not provided for OnProcessItemFailure, the user is not informed of the error. Abbrevia is designed to continue processing commands when an error is received without corrupting files or archives.

The following is a list of the possible ErrorCode:

<b>ErrorCode</b>	<b>Action</b>	<b>Description</b>
AbDuplicateName	Add, move	The file name already exists in the archive. You might want to display a warning to the user, but it can get ridiculous if the user is attempting to add the same large set of files to a directory twice. The file is not added or moved.
AbInvalidPassword	Extract	The specified password is not valid for extracting the encrypted file. The file is not extracted.
AbNoSuchDirectory	Extract	The destination directory does not exist. The file is not extracted.
AbUnknownCompressionMethod	Extract	The file was compressed with a compression method that is not supported by Abbrevia. The file is not extracted.
AbUserAbort	Add, extract freshen	The user aborted the process. The file is not processed.
AbZipBadCRC	Extract	The extracted file's CRC doesn't match the stored CRC. The extracted file is deleted.
AbZipVersionNeeded	Extract	The "Version needed to extract" for this file is not supported by this version of Abbrevia. This might occur in the future when a new version of PKZIP is available. The file is not extracted.

See also: AddFiles, ExtractFiles, ExtractTaggedItems, FreshenFiles, FreshenTaggedItems, OnConfirmProcessItem

## OnRequestBlankDisk

event

```
property OnRequestBlankDisk : TABRequestDiskEvent  
  TABRequestDiskEvent = procedure(Sender : TObject;  
    var Abort : Boolean) of object;
```

↳ Defines an event handler that is called when a blank disk is needed for a spanned archive.

If you do not supply an OnRequestBlankDisk event handler, a dialog box is displayed to prompt the user for a blank disk. If you want to add features such as formatting the disk, scanning the disk for errors, or allowing the user to verify the disk contents, you can do that in an OnRequestBlankDisk event handler. You can abort spanning by setting Abort to True inside the event handler.

## OnRequestImage

event

```
property OnRequestImage : TABRequestImageEvent  
  TABRequestImageEvent = procedure(  
    Sender : TObject; ImageNumber : Word;  
    var ImageName : string; var Abort : Boolean) of object;
```

↳ Defines an event handler to obtain a spanned archive file name.

OnRequestImage provides a mechanism to obtain a particular archive file name when working with a spanned set. ImageName specifies the file of the request archive file within a spanned set. ImageNumber identifies the file's sequence number within the spanned set.

There are two conditions where this event will be fired: when saving an archive, and when extracting files from an archive.

When saving an archive, if the SpanningThreshold is reached, then the OnRequestImage event is fired to obtain the file name (via ImageName) of the next file in the spanned set. If an event handler has not been defined, then the image file name is automatically generated by replacing the last character of the file extension with a sequence number, (e.g. Files.Zip, Files.Z01, Files.Z02, etc.) A maximum of 99 image files can be automatically generated in this fashion. If you require a larger number of image files, then you must define an event handler for the OnRequestImage event.

When extracting an item from a spanned archive set, OnRequestImage is fired to obtain the archive image containing the item. If the item spans archive images, then OnRequestImage is fired as often as required until the item has been extracted. If an event handler is not assigned then an exception will be raised when trying to extract from a spanned set.

See also: SpanningThreshold

## OnRequestLastDisk

event

```
property OnRequestLastDisk : TABRequestDiskEvent  
TABRequestDiskEvent = procedure(  
  Sender : TObject; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called when the last disk of a spanned archive is needed.

The directory information for PKZIP files is stored on the last disk of a spanned archive. If you do not supply an OnRequestLastDisk event handler, a dialog box is displayed to prompt the user for the last disk. If you want to add features such as allowing the user to verify the disk contents, you can do that in an OnRequestLastDisk event handler. You can abort reading the spanned archive by setting Abort to True inside the event handler.

See also: OnRequestBlankDisk, OnRequestNthDisk

## OnRequestNthDisk

event

7

```
property OnRequestNthDisk : TABRequestNthDiskEvent  
TABRequestNthDiskEvent = procedure(Sender : TObject;  
  DiskNumber : Byte; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called when a specific disk in the spanned archive is needed.

If an OnRequestNthDisk event handler is not supplied, a dialog box is displayed to prompt for the disk specified by DiskNumber. If you want to add features such as allowing the user to verify the disk contents, you can do that in an OnRequestNthDisk event handler. You can even abort reading the spanned archive by setting Abort to True inside the event handler.

See also: OnRequestBlankDisk, OnRequestLastDisk

## OnSave

event

```
property OnSave : TABArchiveEvent  
TABArchiveEvent = procedure(Sender : TObject) of object;
```

- ↳ Defines an event handler that is called immediately after the archive's contents are saved.

The OnSave event can be used along with the OnConfirmProcessItem event to display status information for the user. For example, in the OnConfirmProcessItem event handler, you could set the status to "change pending", and in the OnSave event handler, set the status to "saved". This would be most useful when the AutoSave property is False.

See also: OnConfirmProcessItem, OnConfirmSave, Save

```
property OnWindowsDrop : TWindowsDropEvent  
  TWindowsDropEvent = procedure(  
    Sender : TObject; FileName : string) of object;
```

- ↳ Defines an event handler that is called when a file is dropped on the TAbZipOutline from another Windows programs.

This is a Windows only feature.

OnWindowsDrop provides a way to accept files dragged from applications such as Windows Explorer. When an event handler is assigned to OnWindowsDrop, the TAbZipOutline calls DragAcceptFiles, telling Windows that it is a drop target. If files are then dragged onto the component, the component receives a WM\_DROPFILES message and the OnWindowsDrop event is called once for each file dropped on the component.

Drag and drop within the application can be implemented using the VCL's support mechanisms. See the Delphi documentation for TTreeView for more information.

The following example adds the dropped file to the archive. If multiple files are dropped, this event is called for each item dropped on the TAbZipOutline:

```
procedure TForm1.AbZipOutline1WindowsDrop(  
  Sender: TObject; FileName: ShortString);  
begin  
  AbZipOutline1.AddFiles(FileName, 0);  
end;
```

## Options property

---

```
property Options : TAbZipOutlineOptions
TAbZipOutlineOptions = set of TabZipOutlineOption;
TAbZipOutlineOption = (
  zooDrawTreeRoot, zooDrawFocusRect, zooStretchBitmaps);
Default: [zooDrawTreeRoot, zooDrawFocusRect]
```

↳ Determines how the nodes in the TAbZipOutline are drawn.

Set Options to any combination of the following values:

Value	Meaning
zooDrawTreeRoot	The first item (Index value of 1) is connected to the root item by the outline tree. This means that the tree will extend from the top of the outline to all the first level items. Without ooDrawTreeRoot, all first level items appear leftmost in the outline, not connected by the tree.
zooDrawFocusRect	The outline draws a focus rectangle around the selected item.
zooStretchBitmaps	The outline stretches the standard bitmaps (PictureLeaf, PictureOpen, PictureClosed, PicturePlus, PictureMinus) to fit in the size of the item, determined by the size of the Font of the Text. Without ooStretchBitmap, the bitmaps will be cropped if larger than the height of the item text, or won't fill up the entire item space if smaller than the text.

## Password property

---

```
property Password : string
```

↳ The password used for encrypting or decrypting a file.

When a file is added to the archive, Password is used to encrypt it. If Password is empty, the file is not encrypted.

When an attempt is made to extract an encrypted file from the archive, Password is used to decrypt the encryption header. If the decryption is successful, the remainder of the file is decrypted and extracted. If the encryption header cannot be successfully decrypted or

Password is empty, OnNeedPassword is called (if fewer than PasswordRetries attempts have been made). The OnNeedPassword event handler can then provide a password (possibly by prompting the user).

See also: OnNeedPassword, PasswordRetries

### **PasswordRetries**

**property**

**property** PasswordRetries: Byte

**Default:** 3

- ↳ Specifies the maximum number of passwords attempts allowed when extracting an encrypted file.

When the number of retries is exhausted, an exception is raised. See the OnProcessItemFailure event on page 159 for a description of how exceptions that occur during the extract process are handled.

If PasswordRetries is 0 and Password is empty, encrypted files cannot be extracted.

See also: Password

### **PictureFile**

**property**

**property** PictureFile : TBitmap

- ↳ The glyph displayed to the left of each file in the outline.

In the TAbZipOutline, glyphs are displayed to the left of each item in the directory hierarchy. Directories are displayed with a folder glyph on the left. If the directory is expanded, the glyph shows an open folder. If the directory is collapsed, the glyph shows a closed folder.

Files are displayed with a page glyph on the left. If you want to change the glyph that is displayed, assign a new glyph to PictureFile.

See also: PictureZipAttribute

### **PictureZipAttribute**

**property**

**property** PictureZipAttribute : TBitmap

- ↳ The glyph displayed to the left of each item attribute in the outline.

In the TAbZipOutline, item attributes are displayed with a zipper glyph on the left. If you want to change the glyph that is displayed, assign a new glyph to PictureZipAttribute.

See also: PictureFile

## Replace

## method

```
procedure Replace(aItem : TAbArchiveItem);
```

- ↳ Replaces the specified item in the archive.

The next time the archive is saved, the item is replaced. If the file specified by the item's `FileName` property is found on disk, the compressed data in the archive is replaced with the compressed representation of the current file. If the file cannot be found, an `EAbFileNotFoundException` exception is raised.

See the `OnProcessItemFailure` event on page 159 for a description of the exceptions that can occur during the replacement process.

See also: `OnChange`, `Save`, `StoreOptions`

## Save

## method

```
procedure Save;
```

- ↳ Saves updates the archive.

If the archive has not been modified since it was last opened or saved, `Save` returns immediately without modifying the archive.

See also: `AutoSave`, `OnConfirmSave`

## SelectedZipItem

## run-time, read-only property

```
property SelectedZipItem : TAbZipItem
```

- ↳ Returns a reference to the selected item in the outline.

If a folder or an attribute is selected in the outline, `SelectedZipItem` returns nil. If a file is selected in the outline, `SelectedZipItem` returns a valid reference.

The following example calls BeginDrag if the user clicks on the representation of a file in a TAbZipOutline:

```
procedure TForm1.AbZipOutline1MouseDown(
  Sender: TObject; Button: TMouseButton; Shift: TShiftState;
  X, Y: Integer);
var
  i : Integer;
begin
  if Button = mbLeft then begin
    if Assigned(AbZipOutline1.SelectedZipItem) then
      AbZipOutline1.BeginDrag(False);
  end
end;
```

## SpanningThreshold

**property**

**property SpanningThreshold : Longint**

**Default: 0**

↳ Specifies the maximum archive image size.

By specifying SpanningThreshold you can restrict the size of the archive. If the archive size reaches SpanningThreshold when adding or freshening items, the archive is closed and a new archive file is created and the process continues onto the new, spanned archive. This process is repeated until the entire archive has been saved.

Setting SpanningThreshold to 0 specifies that a single archive file is created, up to MaxLongint bytes in size.

Spanning archive files in this fashion requires that a new file name be supplied each time a new archive file needs to be created. This can be done either by a file name via the OnRequestImage event, or allowing the file name to be automatically generated by appending a two digit sequence number to the original archive file name, (e.g. Files.Zip, Files.Z01, Files.Z02, etc.). There is a limit of 99 files that can be auto-generated. For larger spanned sets, you must provide an event handler to supply the file names.

See also: OnRequestImage

## Status

run-time, read-only property

```
property Status : TABArchiveStatus  
TABArchiveStatus = (asInvalid, asIdle, asBusy);
```

- ↳ Determines whether an operation is currently being performed on the archive.

When an archive is being initialized, Status is set to asInvalid. When the initialization is finished, Status is set to asIdle. During each operation that modifies the archive, Status is set to asBusy. Status is used internally by Abbrevia to prevent re-entrant operations, but could also be used to display archive activity.

## StoreOptions

property

```
property StoreOptions : TABStoreOptions  
TABStoreOption = (soStripDrive, soStripPath, soRemoveDots,  
    soRecurse, soFreshen, soReplace);  
TABStoreOptions = set of TABStoreOption;  
Default: [soStripDrive, soRemoveDots]
```

- ↳ Determines the options for archive add and freshen operations.

The following table describes the various options:

Option	Result
soStripDrive	Drive letter information is removed from the stored file name. ( <b>Note:</b> This option is ignored in Linux.)
soStripPath	All path information is removed from the stored file name.
soRemoveDots	All relative path information is removed from the stored file name. For example, if you call AddFiles with a FileMask of "...\\TEST.TXT" ("../TEST.TXT" in Linux), the parent of the current BaseDirectory is searched for a file named "TEST.TXT". If the file is found, it is stored as "TEST.TXT".
soRecurse	Subdirectories of the search path are included in the search for files to add or freshen.
soFreshen	When adding an existing item to the archive, the item is freshened.
soReplace	When adding an existing item to the archive, the item is replaced.

See also: AddFiles, FreshenFiles, Replace

**Style****property**

```
property Style: TAbOutlineStyle
TAbZipOutlineStyle = (zosTextOnly, zosPlusMinusText,
zosPictureText, zosTreeText, zosTreePictureText)
```

**Default:** zosTreePictureText

↳ Specifies how the TAbZipOutline tree is displayed.

Set Style to one of the following values:

<b>Value</b>	<b>Meaning</b>
zosText	Displays node text (specified in the Text property of the TTreenode item) only.
zosPlusMinusText	Displays plus picture, minus picture (both as specified in the StateImages list), and node text.
zosPictureText	Displays open picture, closed picture (both as specified in the StateImages list), leaf picture (as specified in the Images list), and the node text (specified in the Text property of the TTreenode item).
zosPlusMinusPictureText	Displays plus picture, minus picture (both as specified in the StateImages list), open picture, closed picture, leaf picture, (as specified in the Images list), and node text (specified in the Text property of the TTreenode item).
zosTreeText	Displays lines connecting outline nodes and node text (specified in the Text property of the TTreenode item).
zosTreePictureText	Displays lines connecting outline nodes, open picture, closed picture, leaf picture (as specified in the Images list), and node text (specified in the Text property of the TTreenode item).

**Note:** TZipOutline provides default images and text for all items.

## TagItems method

---

```
procedure TagItems(const FileMask : string);
```

↳ Tags the specified archive items.

Abbrevia allows you to perform operations on multiple files in an archive. You can extract, freshen, or delete a group of files by first tagging them and then performing the desired operation.

TagItems tags each archive item whose stored name matches FileMask. Items can be untagged using ClearTags or UnTagItems. Operations can be performed on the tagged items using the DeleteTaggedItems, ExtractTaggedItems, and FreshenTaggedItems methods.

The following example shows how to tag and then extract all files that are larger than 2048 bytes in size and are Pascal source files:

```
7
var
  i : Integer;
begin
  with AbZipOutline1 do begin
    {clear all tagged items}
    ClearTags;
    {tag all items which match '*.pas'}
    TagItems('*.*.pas');
    if Count > 0 then
      for i := 0 to pred(Count) do
        if Items[i].UncompressedSize <= 2048 then
          {if it's too small, untag it}
          Items[i].Tagged := False;
    {extract each file that's left}
    ExtractTaggedItems;
  end;
end;
```

See also: ClearTags, ExtractTaggedItems, DeleteTaggedItems, ExtractTaggedItems, FreshenTaggedItems, UnTagItems

## TempDirectory property

---

```
property TempDirectory : string
```

↳ Specifies a temporary directory to use during archive operations.

By setting TempDirectory, you change specify where the temporary files used by Abbrevia are created. If this property is the empty string, the system's temporary directory will be used.

## TestTaggedItems

method

```
procedure TestTaggedItems;
```

↳ Performs integrity test on tagged items.

For each tagged item in the archive the following things are checked:

- The central directory record.
- The local file header.
- The CRC for the file.

Some of these checks require the file to be extracted. Abbrevia extracts the file to a special stream, and then, after the checks are made, the stream is destroyed.

## UnTagItems

method

```
procedure UnTagItems(const FileMask : string);
```

↳ Untags the specified archive items.

UnTagItems sets the Tagged property to False for each archive item whose stored name matches FileMask.

See also: ClearTags, TagItems

## Version

read-only property

```
property Version : string
```

↳ Returns the current version number of Abbrevia.

Version is provided so you can identify your Abbrevia version if you need technical support. If you double-click on the Version property in the IDE's object inspector, the Abbrevia About box is displayed.

## ZipFileComment

run-time property

```
property ZipFileComment : string
```

↳ Returns the comment stored in the Zip archive.

A comment for the archive can be stored in PKZIP-compatible archives.



---

# Chapter 8: Archive Viewer Components

Abbrevia includes two visual components, TAbZipView and TAbCabView, which when attached to an appropriate non-visual component, display the items contained in an archive in a grid with a rich assortment of viewing options.

The grid has a number of features that allow you to customize its function and appearance extensively. These features include:

- Sizeable, movable columns to display (or not) various attributes of the archive items.
- Sort by item name, file size, compressed size, or timestamp by clicking the column header.
- Complete control over the color and size of the windowed display.
- Display icons registered to the type of files in the archive.
- Easily add to existing applications to enhance the non-visual Abbrevia components.

The TAbZipView and TAbCabView components descend from the TAbBaseViewer class which provides their common functionality.

---

## TAbColors Class

TAbColors class encapsulates the colors used by the viewer components to simplify the Object Inspector display in the IDE.

### Hierarchy

TPersistent

  TAbColors (AbView)

### Properties

  Alternate

  Deleted

  Selected

  AlternateText

  DeletedText

  SelectedText

## Reference Section

---

<b>Alternate</b>	<b>property</b>
------------------	-----------------

---

property Alternate : TColor

Default: clAqua

- ↳ The background row color for alternate row coloring.

---

<b>AlternateText</b>	<b>property</b>
----------------------	-----------------

---

property AlternateText : TColor

Default: clRed

- ↳ The font color for alternate row coloring.

---

<b>Deleted</b>	<b>property</b>
----------------	-----------------

---

property Deleted : TColor

Default: clYellow

- ↳ The background row color for items marked to be deleted.

---

<b>DeletedText</b>	<b>property</b>
--------------------	-----------------

---

property DeletedText : TColor

Default: clNavy

- ↳ The font color for items marked to be deleted.

---

<b>Selected</b>	<b>property</b>
-----------------	-----------------

---

property Selected : TColor

Default: clHighlight

- ↳ The background row color for selected items.

---

<b>SelectedText</b>	<b>property</b>
---------------------	-----------------

---

property SelectedText : TColor

Default: clHighlightText

- ↳ The font color for selected items.

---

## TAbBaseViewer Class

The TAbBaseViewer component is the immediate ancestor for the TAbZipView and TAbCabView components and provides the common functionality and references an abstract TAbArchive class.

### Hierarchy

TCustomGrid

  TAbBaseViewer (AbView)

### Properties

ActiveRow	DefaultColWidth	SelCount
Attributes	DefaultRowHeight	Selected
Colors	DisplayOptions	SortAttributes
Count	HeaderRowHeight	Version
ColWidths	Headings	

8

### Methods

BeginUpdate	EndUpdate
ClearSelections	SelectAll

### Events

OnChange	OnSorted
----------	----------

# Reference Section

## ActiveRow

run-time property

property ActiveRow : Longint

- ↳ Contains the table row number of the currently selected item.

ActiveRow must be between 0 and Count-1. The column header row has no row number, thus it is never referenced by ActiveRow. If no item is currently selected, ActiveRow is set to -1.

ActiveRow gives the table row number of the selected item, not the item's index in the attached archive's item list. This value should only be used to access the corresponding archive item via the Items property of the viewer. For example:

```
Caption := AbZipView1.Items[AbZipView1.ActiveRow].Filename;
```

See also: Selected

## Attributes

property

property Attributes : TAbViewAttributes

```
TAbViewAttribute = (
  vaItemName, vaPacked, vaMethod, vaRatio, vaCRC,
  vaFileAttributes, vaFileType, vaEncryption, vaTimeStamp,
  vaFileSize, vaVersionMade, vaVersionNeeded, vaPath);
```

TAbViewAttributes = set of TAbViewAttribute;

Default: [vaItemName, vaPacked, vaTimeStamp, vaFileSize, vaPath]

- ↳ Specifies which item attributes to display.

Attributes is used to select which attributes (file size, CRC, etc.) to display for the items in the archive. Omitting an attribute from Attributes will hide the corresponding column from the table.

The possible values for Attributes are:

Attribute	Description
vaItemName	The item's file name.
vaPacked	The compressed size of the item.
vaMethod	The compression method used.
vaRatio	The compression ratio.

<b>Attribute</b>	<b>Description</b>
vaCRC	The 32-bit CRC (cyclic redundancy checksum).
vaFileAttributes	The item's FAT-style file attributes.
vaFileType	0 for binary file, 1 for text file.
vaEncryption	Indicates whether the item is encrypted.
vaTimeStamp	The item's last modified date and time.
vaFileSize	The uncompressed size of the item.
vaVersionMade	The version of PKZIP that created the item.
vaVersionNeeded	The version of PKZIP needed to extract the item.
vaPath	The item's DiskPath name.

## **BeginUpdate** **method**

`procedure BeginUpdate;`

**8**

↳ Prevents the viewer from updating until EndUpdate is called.

Use BeginUpdate to prevent the screen from being repainted when adding items to, or deleting items from an archive.

## **ClearSelections** **method**

`procedure ClearSelections;`

↳ Unselects any selected items.

ClearSelections resets the selection list and sets SelCount to zero. The ActiveRow is set to -1.

See also: Selected

## **Colors** **property**

`property Colors : TABColors`

↳ The set of colors used to paint the rows.

See TABColors Class on page 174 for details on the colors used by the viewer.

**ColWidths****property****property ColWidths[ Index: Longint ]: Integer**

↳ Determines the width of an attribute column in pixels.

Use ColWidths to override DefaultColWidth for a particular attribute column.

See also: Selected

**Count****run-time, read-only property****property Count : Longint**

↳ Specifies the number of items in the archive.

Use Count to determine the maximum valid row number to use when accessing the items displayed by the viewer. For example:

```
for i := Pred(ABZipView1.Count) downto 0 do
```

See also: Selected

**DefaultColWidth****property****property DefaultColWidth : Integer**

**Default:** 150

↳ Determines the width of all the columns within the viewer.

Use DefaultColWidth to set the width of all columns in the viewer to a common value. To set the width of an individual column, use the ColWidths property.

See also: Selected

**DefaultRowHeight****property****property DefaultRowHeight : Integer**

**Default:** 24

↳ Determines the height of all item rows within the viewer.

Use DefaultRowHeight to adjust the height of the item rows in the viewer.

Changing font size will cause DefaultRowHeight to be adjusted to accommodate the text in the row. If you do not wish to have DefaultRowHeight changed, then simply reset DefaultRowHeight to the desired value after changing font properties.

```
property DisplayOptions : TAbDisplayOptions  
  
TAbDisplayOption = (  
  doAlternateColors, doColLines, doColMove, doColSizing,  
  doMultiSelect, doRowLines, doShowIcons, doThumbTrack,  
  doTrackActiveRow);  
  
TAbDisplayOptions = set of TAbDisplayOption;
```

Default: [doColSizing]

↳ Determines the behavior of the viewer.

Use DisplayOptions to select the following available viewer options:

Option	Description
doAlternateColors	Alternate rows are displayed with different colors as determined by Color, Font.Color, Colors.Alternate, and Colors.AlternateText.
doColLines	Display vertical lines between the attribute columns.
doColMove	Allow columns to be moved by clicking and dragging on the column header button.
doColSizing	Allow columns to be individually resized.
doMultiSelect	Allow multiple items in the display to be selected. With this option enabled, use the SelCount property to determine the number of items selected, and the Selected array property to determine the selection status of an individual item.
doRowLines	Display horizontal lines between the item rows.
doShowIcons	Display the icon registered to the item file name to the left of the file name. (Windows only)
doThumbTrack	The viewer scrolls at the same time as the horizontal or vertical scrollbar thumb is moved.
doTrackActiveRow	Forces the active row item to always remain in view when sorting rows.

## **EndUpdate**

**method**

```
procedure EndUpdate;
```

↳ Re-enables screen repainting that was turned off with BeginUpdate.

## **HeaderRowHeight**

**property**

```
property HeaderRowHeight : Integer
```

↳ Specifies the height of the column header buttons in pixels.

Use HeaderRowHeight to set the height of the header buttons.

## **Headings**

**property**

```
property Headings : TAbColHeadings
```

```
TAbColHeadings = class(TStringList)
```

↳ Changes the heading text displayed by the column header buttons.

The default heading strings are as follows:

<b>Option</b>	<b>Text</b>
vaItemName	"Name"
vaPacked	"Packed"
vaMethod	"Method"
vaRatio	"Ratio"
vaCRC	"CRC32"
vaFileAttributes	"Attributes"
vaFileType	"Format"
vaEncryption	"Encrypted"
vaTimeStamp	"Time Stamp"
vaFileSize	"Size"
vaVersionMade	"Version Made"
vaVersionNeeded	"Version Needed"
vaPath	"Path"

The column heading strings are indexed by view attribute. For example, assuming the viewer in question is a TAbZipView component, you could change “Packed” to “Compressed” at run time by the following:

```
AbZipView1.Headings[vaPacked] := 'Compressed';
```

See also: Selected

---

**OnChange** event

```
property OnChange : TNotifyEvent
```

↳ Defines an event handler that is called when the archive changes.

OnChange is called when the attached archive is opened or closed, or when the contents of the archive change (immediately after an add, delete, freshen, or move operation).

---

**OnSorted** event

```
property OnSorted : TAbSortedEvent
```

```
TAbSortedEvent = procedure(  
  Sender : TObject; Attr : TAbViewAttribute) of object;
```

↳ Defines an event handler that is called when the item rows are sorted.

OnSorted is called when the display has been sorted in response to a mouse click on one of the column header buttons. Attr identifies which item attribute was used to sort.

See also: Selected

---

**SelCount** run-time, read-only property

```
property SelCount : Longint
```

↳ Returns the number of items selected.

Use SelCount to determine the number of items that have been selected when doMultiSelect is in DisplayOptions. If no items are selected, SelCount is 0.

See also: Selected

```
procedure SelectAll;
```

↳ Selects all items in the viewer.

When multi-selection is enabled, SelectAll marks all items in the viewer as selected and sets SelCount equal to count. When multi-selection is not enabled, SelectAll does nothing.

See also: Selected

---

**Selected****run-time property**

```
property Selected[RowNum : Longint] : Boolean
```

↳ Contains the selection status of the individual items.

The Selected property determines whether a particular item is selected in the viewer. RowNum is a value between 0 and Count -1. Selected item rows are displayed according to the Colors.Selected and Colors.SelectedText properties.

Assuming the viewer is a TAbZipView component, the following example shows how the Selected property could be used to compute the sum of the uncompressed file sizes of selected items:

```
var
  SizeSum : Longint;
  i        : Longint;
begin
  SizeSum := 0;
  with AbZipView1 do begin
    if (SelCount > 0) then
      for i := Pred(Count) downto 0 do begin
        if Selected[i] then
          Inc(SizeSum, Items[i].UncompressedSize);
      end;
  end;
  Result := SizeSum;
```

See also: Selected

## SortAttributes

## property

```
property SortAttributes : TAbSortAttributes  
TAbSortAttribute = (  
  saItemName, saPacked, saRatio, saTimeStamp, saFileSize);  
TAbSortAttributes = set of TAbSortAttribute;  
Default: []
```

↳ Determines which item attributes may be sorted.

If a sort attribute is included in the set, then when the corresponding column header button is clicked, the items are sorted according by that attribute. Sorting alternates between ascending order and descending order each time a sort is performed. If doTrackActiveRow display option is selected, the viewer is automatically scrolled to keep the active row in view. After the display has been sorted, the OnSorted event is fired.

Assuming that the viewer is a TAbZipView component, you could allow the user to sort the rows either by file name or by time stamp by the following:

```
AbZipView1.SortAttributes := [saItemName, saTimeStamp];
```

See also: Selected

8

## Version

## property

```
property Version : string
```

↳ Shows the current version of Abbrevia.

Version is provided so you can identify your Abbrevia version if you need technical support. The Abbrevia About box can be displayed by double-clicking this property or selecting the dialog button to the right of the property value in the IDE's object inspector. Setting this property has no effect.

# TAbZipView Component

The TAbZipView component provides a grid display of a PKZIP, GZip, Tar, or Gzipped Tar compatible archive.

Simply attach it to a non-visual Abbrevia zip component via the ZipComponent property. The TAbZipView component will display the current contents of a zip archive and keeps its display updated automatically with any additions to, or deletions from the archive.

## Hierarchy

TCustomGrid

- ① TAbBaseViewer (AbView) ..... 176
- TAbZipView (AbZView)

## Properties

- |                   |                    |                  |
|-------------------|--------------------|------------------|
| ① ActiveRow       | ① DefaultRowHeight | ① Selected       |
| ① Attributes      | ① DisplayOptions   | ① SortAttributes |
| ① Colors          | ① HeaderRowHeight  | ① Version        |
| ① ColWidths       | ① Headings         | ZipComponent     |
| ① Count           | Items              |                  |
| ① DefaultColWidth | ① SelCount         |                  |

## Methods

- |                   |             |
|-------------------|-------------|
| ① BeginUpdate     | ① EndUpdate |
| ① ClearSelections | ① SelectAll |

## Events

- |            |            |
|------------|------------|
| ① OnChange | ① OnSorted |
|------------|------------|

## Reference Section

Items	run-time property
-------	-------------------

property Items[RowNum : Longint] : TAbZipItem

↳ Provides access to the items in the attached archive.

Use Items to access the individual archive items of the attached archive rather than the latter's Items property. The Items property differs from the non-visual component's Items property in that the row number is mapped internally to the index of the item in the archive. Thus, when the items are sorted and row numbers change, the current row number of an item in the display can still be used to access the item.

You must still use the attached non-visual component to perform any operation with the item, such as extracting or deleting. Assuming the attached non-visual component is a TAbZipKit, the following example shows how to extract the currently selected item:

```
withAbZipView1 do  
  TABZipKit(ZipComponent).  
    ExtractFile(Item[ActiveRow].FileName);
```

8

See also: Selected

ZipComponent	property
--------------	----------

property ZipComponent : TABCustomZipBrowser

↳ Specifies the attached non-visual zip component.

The TABZipView requires a TABCustomBrowser instance (TABZipBrowser, TABUnZipper, TABZipper, or TABZipKit) to provide access to the zip archive. Set ZipComponent to specify which non-visual component to attach to the viewer. Unless a component is attached, the viewer cannot access the archive. Any actions upon the archive, such as opening, closing, adding files, deleting files, etc., must be done via the attached component.

# TAbCabView Component

The TAbCabView component provides a grid display of a cabinet archive. Simply attach it to a non- visual Abbrevia cabinet component via the CabComponent property. The TAbCabView component will display the current contents of a cabinet archive and keeps its display updated automatically with any additions to the archive.

## Hierarchy

TCustomGrid

① TAbBaseViewer (AbView) .....	176
TAbCabView (AbCView)	

## Properties

① ActiveRow	① DefaultColWidth	① SelCount
① Attributes	① DefaultRowHeight	① Selected
CabComponent	① DisplayOptions	① SortAttributes
① Colors	① HeaderRowHeight	① Version
① ColWidths	① Headings	
① Count	Items	

## Methods

① BeginUpdate	① EndUpdate
① ClearSelections	① SelectAll

## Events

① OnChange	① OnSorted
------------	------------

## Reference Section

<b>CabComponent</b>	<b>property</b>
---------------------	-----------------

property CabComponent : TAbCustomCabBrowser

- ↳ Specifies the attached non-visual cab component.

The TAbCabView requires a TAbCustomCabBrowser descendent (TAbCabBrowser, TAbCabExtractor, TAbMakeCab, or TAbCabKit) to provide access to the cabinet archive. Set CabComponent to specify which non-visual component to attach the viewer to. Unless a component is attached here, the viewer cannot “see” the archive. Any actions upon the archive, such as opening, closing, adding files, or extracting files, must be done via the attached component.

<b>Items</b>	<b>run-time property</b>
--------------	--------------------------

property Items[RowNum : Longint] : TAbCabItem

- ↳ Provides access to the items in the attached cabinet archive.

8

Use Items to access the individual archive items of the attached cabinet rather than the latter's Items property. The Items property differs from the non-visual component's Items property in that the row number is mapped internally to the index of the item in the archive. Thus, when the items are sorted and row numbers change, the current row number of an item in the display can still be used to access the item.

You must still use the attached non-visual component to perform any operation with the item, such as extracting or deleting. Assuming the attached non-visual component is a TAbCabKit, the following example illustrates by showing how to extract the currently selected item:

```
withAbZipView1 do
  TAbZipKit(ZipComponent).
    ExtractFile(Item[ActiveRow].FileName);
```

See also: Selected

# Chapter 9: Compound File Classes

The TAbCompoundFile class introduced in Abbrevia 3 provides the user the ability to store multiple, compressed files as part of a single file. This functionality emulates the functionality of Microsoft's Structured Storage mechanism, but does not rely on COM interfaces to achieve this functionality. Abbrevia's compound file class encapsulates a miniature file system within the single compound file. The mechanism employed to achieve this end result is similar to the FAT file system. Figure 9.1 depicts the internal structure of Abbrevia's compound files.

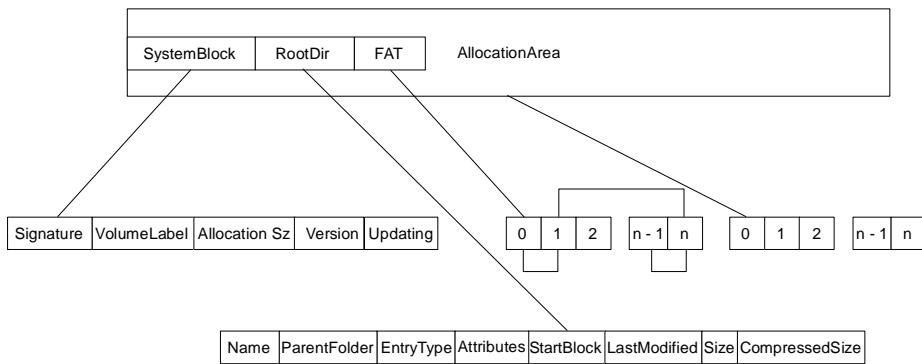


Figure 9.1: Abbrevia's compound file internal structure

There are three fundamental sections of Abbrevia's TAbCompoundFile structure that are maintained internally: the System block, the Root Directory block, and the File Allocation Table (FAT). Abbrevia's TAbCompoundFile class manages each of these sections and insulates the user from the intricacies of dealing with each section. Despite the fact that the user needs to know nothing of these structures, some background information may be helpful. Each of these sections are described in the following paragraphs.

## System Block

The System Block is a relatively small data block (89 bytes) that contains the file signature, volume label, allocation block size, version, and an updating flag. The system block consumes one single allocation unit and thus restricts the minimum allocation size to 128 bytes (the smallest power of 2 that is larger than the system block).

### Signature

The signature is used to uniquely identify the compound file structure as one that was created by the TAbCompoundFile component (40 bytes).

## Volume label

The volume label is a simple string that the user can identify the file with in addition to the disk file name (40 bytes).

## Allocation size

The allocation size indicates the size of each block (4 bytes—power of 2).

## Version

The version specifies the version of the component that created the compound file (4 bytes).

## Updating flag

The updating flag is used internally to indicate that internal processing is in progress. This flag will be reset to 0 when processing is complete to allow subsequent processing (1 byte).

## Root Directory block

The Root Directory block defines a file system hierarchy in contiguous storage. 64 bytes of data are required for each entry in the Root Directory, and one entry is required for every file or folder added to the compound file. This block contains the following fields:

### Name

Name identifies the file or folder name (32 bytes).

### ParentFolder

ParentFolder contains the index within the Root Directory block of the parent (next highest level) folder. This value will always be a positive integer since all folders and files are ultimately contained in the root folder (4 bytes).

### EntryType

EntryType indicates whether the entry is a file or folder (4 bytes).

### Attributes

Attributes define the file attributes pertaining to this entry of the Root Directory block (4 bytes).

### StartBlock

StartBlock contains an index into the FAT table, which points to the first allocated block of data in which the file is stored (4 bytes).

### LastModified

LastModified contains a date-time stamp indicating when the file was last modified (TDateTime 8 bytes).

## Size

Size contains the uncompressed file size of the file (4 bytes).

## Compressed size

Compressed size contains the compressed file size of the file (4 bytes).

## File Allocation Table (FAT)

The File Allocation Table (FAT) simply contains an array of 32 bit integers that are indexes into individual blocks of the allocation area. When reading file data from the allocation area, the FAT entries comprise a linked list to where the physical data resides.

## Allocation Area

The Allocation Area is simply comprised of same sized (default of 512 bytes) data blocks that serve as persistent storage for each of the files and folders added to the compound file.

## Hierarchy

TObject

  TAbCompoundFile

9

## Properties

AllocationSize	FileName	VolumeLabel
CurrentDirectory	SizeOnDisk	
DirectoryEntries	Version	

## Methods

AddFile	DeleteFile	OpenFile
AddFolder	DeleteFolder	PopulateTreeView
Create	Destroy	RenameFile
Defrag	Open	RenameFolder

## Events

OnAfterOpen	OnBeforeDirDelete	OnBeforeFileDelete
OnBeforeClose	OnBeforeDirModified	OnBeforeFileModified

## Reference Section

### AddFile method

---

```
procedure AddFile(  
  FileName : string; FileData : TStream; FileSize : Integer);
```

↳ Adds a single file to the current directory of the compound file.

AddFile creates an entry in the compound file's root directory, and adds the data contained in the FileData parameter to the compound file. The data is compressed and an OnBeforeDirModified event will be fired for the current directory prior to allocating storage blocks and writing to the compound file.

If a file of the same name already exists in the current directory, the AddFile method does nothing and instead returns immediately.

The DirectoryEntries and SizeOnDisk properties will be updated after calling the AddFile method.

See also: AddFolder, OnBeforeDirModified, CurrentDirectory, DirectoryEntries, SizeOnDisk

9

### AddFolder method

---

```
procedure AddFolder(FolderName : string);
```

↳ Adds the specified folder to the current directory of the compound file.

AddFolder creates a new directory entry in the compound file's current directory. An OnBeforeDirModified event will be fired for the current directory prior to adding the directory entry.

If a folder of the same name already exists in the current directory, the AddFolder method does nothing and instead returns immediately.

The DirectoryEntries property will be updated after calling the AddFolder method. It is also possible that the SizeOnDisk property will also be increased by AllocationSize bytes if the compound file's directory structure requires additional storage for the new directory entry.

See also: CurrentDirectory, DirectoryEntries, OnBeforeDirModified, SizeOnDisk

**AllocationSize****property**

```
property AllocationSize : Integer
```

↳ Contains the size in bytes of the allocation units.

Default: 512

The compound file's allocation area is simply comprised of same sized data blocks that serve as persistent storage for each of the files and folders added to the compound file. This value is determined by the AllocSize value that is passed in the constructor and cannot be modified after creation.

On average, smaller block sizes result in less wasted storage space, but longer access times for files that span several allocation units.

The AllocationSize must be set to a power of two between 128 and 65536 bytes. The following table illustrates the possible allocation sizes and the corresponding number of directory entries that will fit into a single allocation unit:

<b>AllocationSize</b>	<b>Directory Entries</b>
128	2
256	4
512	8
1024	16
2048	32
4096	64
8192	128
16384	256
32768	512
65536	1024

See also: Create, DirectoryEntries

## Create

## method

```
constructor Create(  
  AOwner : TComponent; AllocSize : Integer; VolumeLabel : string);
```

Creates and opens an instance of a TAbCompoundFile object.

The AOwner parameter defines the TComponent descendant that owns the compound file object.

The Create method creates and initializes the File Allocation Table (FAT) and creates a directory entry for the root folder. The name of the root directory entry is specified in the VolumeLabel parameter. If the VolumeLabel parameter is empty, the root directory name defaults to “Root”.

The AllocSize parameter determines the size of each of the compound file’s allocation blocks. The compound file’s allocation area is simply comprised of same sized data blocks that serve as persistent storage for each of the files and folders added to the compound file. This value is set within this constructor and cannot be changed later. See AllocationSize on page 193 for more information

An OnAfterOpen event is fired after creation and initialization.

See also: Destroy, OnAfterOpen, Open

9

## CurrentDirectory

## property

```
property CurrentDirectory : string
```

Returns a string indicating the current directory within the compound file.

This property is used to navigate the directory structure of the compound file in an identical manner to the DOS command ChDir (or CD). When setting this property, the new property value or directory must exist within the context of the current directory. If an invalid path is specified, the CurrentDirectory property will not be modified.

To create a new directory, use the CreateFolder method.

See also: AddFolder

**Defrag****method**

```
procedure Defrag;
```

- ↳ Defragments the compound file.

Just as in any file system, frequent file modifications may result in a fragmented internal file structure, meaning that files are stored within several non-contiguous allocation units. File access times are negatively impacted by such fragmentation. The Defrag method reorganizes the internal file structure so that the individual files that comprise the compound file are stored in contiguous allocation blocks.

**DeleteFile****method**

```
procedure DeleteFile(FileName : string);
```

- ↳ Deletes the specified file from the compound file.

If the file specified in the FileName parameter is found in the current directory, the file will be deleted from the compound file and the File Allocation Table (FAT) will be updated to indicate that the space formerly occupied by the file is now available for use. If the file is not found, no action will be taken.

An OnBeforeFileDelete event will be fired for the file that is about to be deleted and a corresponding OnBeforeDirModified event will be fired for the file's parent folder.

Additionally, the DirectoryEntries and SizeOnDisk properties will be updated to reflect the file deletion.

See also: OnBeforeFileDelete, OnBeforeDirModified, DirectoryEntries, SizeOnDisk, AddFile, DeleteFolder

**DeleteFolder****method**

```
procedure DeleteFolder(FolderName : string);
```

- ↳ Deletes the specified folder from the compound file.

If the folder specified in the FolderName parameter is found in the current directory, the folder will be deleted from the compound file and the File Allocation Table (FAT) and directory entry structure will be updated. If the folder is not found or the folder is not empty, no action will be taken.

An OnBeforeDirDelete event will be fired for the folder that is about to be deleted and a corresponding OnBeforeDirModified event will be fired for the folder's parent folder.

Additionally, the DirectoryEntries property will be updated to reflect the folder deletion.

**Note:** It is not possible to delete a folder that contains files or subfolders. The folder must be completely empty before it can be deleted.

See also: [OnBeforeDirDelete](#), [OnBeforeDirModified](#), [DirectoryEntries](#), [AddFolder](#), [DeleteFile](#)

---

<b>Destroy</b>	<b>method</b>
----------------	---------------

---

`destructor Destroy;`

↳ Frees the memory occupied by the compound file object.

The `Destroy` method does not delete the compound file from disk. Rather, it persists any pending changes to the compound file structure and releases the memory held by the directory entry structure and the File Allocation Table (FAT).

See also: [Create](#)

---

<b>DirectoryEntries</b>	<b>read-only property</b>
-------------------------	---------------------------

---

`property DirectoryEntries : Integer`

↳ Returns the number of directory entries in the compound file.

Each file and each folder stored within the compound file occupy a single entry within the compound file's directory structure. This property simply contains the number of directory entries currently contained within the compound file.

**Note:** All compound files contain at least one directory entry for the root directory.

See also: [AddFile](#), [AddFolder](#), [DeleteFile](#), [DeleteFolder](#)

---

<b>FileName</b>	<b>property</b>
-----------------	-----------------

---

`property FileName : string`

↳ Contains the disk file name of the compound file.

This property contains the name of the compound file on disk and should not be confused with the internal files stored within the compound file.

**OnAfterOpen****event**

```
property OnAfterOpen : TNotifyEvent
TNotifyEvent = procedure(Sender: TObject) of object;
```

Defines an event that is called immediately after opening a compound file.

This event is fired whenever a compound file is opened. The Sender parameter contains a reference to the TAbCompoundFile instance that was just opened.

See also: [OnBeforeClose](#), [OnBeforeDirDelete](#), [OnBeforeDirModified](#), [OnBeforeFileDelete](#), [OnBeforeFileModified](#)

**OnBeforeClose****event**

```
property OnBeforeClose : TNotifyEvent
TNotifyEvent = procedure(Sender: TObject) of object;
```

Defines an event that is called prior to closing a compound file.

This event is fired just prior to closing a compound file. The Sender parameter contains a reference to the TAbCompoundFile instance that is about to close.

See also: [OnAfterOpen](#), [OnBeforeDirDelete](#), [OnBeforeDirModified](#), [OnBeforeFileDelete](#), [OnBeforeFileModified](#)

**OnBeforeDirDelete****event**

```
property OnBeforeDirDelete : TBeforeDirDeleteEvent
TBeforeDirDeleteEvent = procedure(Sender : TAbCompoundFile;
Dir : string; var AllowDelete : Boolean) of object;
```

Defines an event that is called prior to deleting a directory.

The Sender parameter contains a reference to the TAbCompoundFile instance containing the folder that is about to be deleted.

The Dir parameter contains the name of the folder that is about to be deleted.

To cancel the deletion, set the AllowDelete parameter to False within your event handler. To allow the deletion to proceed, no action is required.

See also: [OnAfterOpen](#), [OnBeforeClose](#), [OnBeforeDirModified](#), [OnBeforeFileDelete](#), [OnBeforeFileModified](#)

## OnBeforeDirModified

event

```
property OnBeforeDirModified : TBeforeDirModifiedEvent  
TBeforeDirModifiedEvent = procedure(Sender : TAbCompoundFile;  
          Dir : string; var AllowModify : Boolean) of object;
```

Defines an event that is called prior to modifying a directory.

The Sender parameter contains a reference to the TAbCompoundFile instance containing the folder whose contents are about to be modified.

The Dir parameter contains the name of the folder whose contents are about to be modified.

To cancel the modification, set the AllowModify parameter to False within your event handler. To allow the modification to proceed, no action is required.

See also: [OnAfterOpen](#), [OnBeforeClose](#), [OnBeforeDirDelete](#), [OnBeforeFileDelete](#), [OnBeforeFileModified](#)

## OnBeforeFileDelete

event

```
property OnBeforeFileDelete : TBeforeFileDeleteEvent  
TBeforeFileDeleteEvent = procedure(Sender : TAbCompoundFile;  
          FileName : string; var AllowDelete : Boolean) of object;
```

Defines an event that is called prior to deleting a file from the compound file.

The Sender parameter contains a reference to the TAbCompoundFile instance containing the file that is about to be deleted.

The FileName parameter contains the name of the file that is about to be deleted.

To cancel the deletion, set the AllowDelete parameter to False within your event handler. To allow the deletion to proceed, no action is required.

See also: [OnAfterOpen](#), [OnBeforeClose](#), [OnBeforeDirDelete](#), [OnBeforeDirModified](#), [OnBeforeFileModified](#)

## OnBeforeFileModified

event

```
property OnBeforeFileModified : TBeforeFileModifiedEvent  
TBeforeFileModifiedEvent = procedure(Sender : TAbCompoundFile;  
          FileName : string; var AllowModify : Boolean) of object;
```

Defines an event that is called prior to modifying a file.

The Sender parameter contains a reference to the TAbCompoundFile instance containing the file whose contents are about to be modified.

The `FileName` parameter contains the name of the file whose contents are about to be modified.

To cancel the modification, set the `AllowModify` parameter to `False` within your event handler. To allow the modification to proceed, no action is required.

See also: `OnAfterOpen`, `OnBeforeClose`, `OnBeforeDirDelete`, `OnBeforeDirModified`, `OnBeforeFileDelete`

## Open

method

```
procedure Open(FileName : string);
```

↳ Opens an existing compound file.

Opens the compound file specified by the `FileName` parameter. The directory structure and the File Allocation Table (FAT) are loaded into memory. After opening the file, the `OnAfterOpen` event is fired.

If the `TAbCompoundFile` class determines that the file is not a valid Abbrevia 3 compound file structure, the file is closed and no events are fired.

See also: `OpenFile`

## OpenFile

method

```
function OpenFile(FileName : string; var Strm : TStream) : Integer;
```

↳ Opens the file specified and writes the file contents to the `Strm` parameter.

The `OpenFile` function opens the file specified by the `FileName` parameter from within the compound file. The file is then uncompressed and written to the `Strm` parameter. The return value indicates the uncompressed size of the file in bytes.

If the file is not found within the context of the `CurrentDirectory`, the `OpenFile` method returns zero and nothing is written to the `Strm` parameter.

See also: `CurrentDirectory`

## PopulateTreeView

method

```
function PopulateTreeView(TreeView : TTreeView) : Integer;
```

↳ Populates the TTreeView with the directory entries of the compound file.

This method populates the nodes of the TreeView parameter with the directory structure of the compound file. All folder entries within the directory structure of the compound file will have an ImageIndex property value of zero, and all file entries within the directory structure of the compound file will have an ImageIndex property value of one.

The PopulateTreeView method returns the number of nodes added to the TreeView parameter.

● Caution: The TreeView parameter is initially cleared by this method.

See also: DirectoryEntries

## RenameFile

method

```
function RenameFile(OrigName, NewName : string) : Boolean;
```

↳ Renames the file specified by OrigName to the value passed in the NewName parameter.

If the file specified by the OrigName parameter is found within the context of the CurrentDirectory and a file with the same name as the NewName parameter is not found in the CurrentDirectory, the file will be renamed to the value specified by NewName and the result will be True.

If either the file specified by OrigName cannot be found, or an existing file with the same name as that specified by the NewName parameter is found, the original file will not be renamed and the result of the function will be False.

See also: CurrentDirectory, RenameFolder

## RenameFolder

method

```
function RenameFolder(OrigName, NewName : string) : Boolean;
```

↳ Renames the folder specified by OrigName to the value passed in the NewName parameter.

If the folder specified by the OrigName parameter is found within the context of the CurrentDirectory and a folder with the same name as the NewName parameter is not found in the CurrentDirectory, the folder will be renamed to the value specified by NewName and the result will be True.

If either the folder specified by OrigName cannot be found, or an existing folder with the same name as that specified by the NewName parameter is found, the original folder will not be renamed and the result of the function will be False.

See also: CurrentDirectory, RenameFile

---

**SizeOnDisk****read-only property**

```
property SizeOnDisk : Integer
```

↳ Returns the disk size of the compound file.

The SizeOnDisk property contains the number of used File Allocation Table (FAT) entries times the allocation size. This value is reflective of the sum of the individual compressed file sizes plus any unused space within partially occupied allocation blocks.

See also: AllocationSize, DirectoryEntries

---

**Version****read-only property**

```
property Version : string
```

↳ Contains the version of the compound file engine used to create the compound file.

The Version property is stored within the compound file to indicate that the compound file was created using the TAbCompoundFile class within Abbrevia. This property is informational only.

See also: VolumeLabel

---

**VolumeLabel****property**

```
property VolumeLabel : string
```

↳ Contains the volume label of the compound file.

The VolumeLabel is used to name the root directory entry of the compound file. This property value is generally passed into the constructor of the TAbCompoundFile class, but can be modified later without adverse effect.

See also: Create, Version



---

# Chapter 10: Low-Level Compression Classes

Abbrevia's components interact with an archive via a set of classes that make up the Abbrevia application programming interface (API). This section provides a brief description of the classes. They are documented fully in the Abbrevia help file.

The AbArcTyp unit implements the abstract classes that describe an archive and the contents of an archive. TAbArchiveItem describes a single item in the archive and TAbArchive describes the archive as a whole. These classes are designed to act as ancestor classes for more specific archive representations.

The AbZipTyp unit implements the classes that describe a PKZIP-compatible archive and the contents of that archive. TAbZipItem is derived from TAbArchiveItem and describes a single item in the archive. TAbZipArchive is derived from TAbArchive and describes the archive as a whole.

The AbZipTyp unit also defines several low-level classes that encapsulate portions of a PKZIP-compatible archive. These classes describe the local file header, data descriptor, central directory header, and central directory footer portions of a zip file. For more information on the structure of a zip file, see APPNOTE.TXT, which is distributed with the PKZIP distribution files.

The AbGZTyp unit implements the classes that describe a GZip-compatible archive and the contents of that archive. TAbGZipItem is derived from TAbArchiveItem and describes a single item in the archive. TAbGZipArchive is derived from TAbArchive and describes the archive as a whole. TABGZipArchive assumes there is only one item stored in a GZip (the common practice).

The AbTarTyp unit implements the classes that describe a Tar-compatible archive and the contents of that archive. TAbTarItem is derived from TAbArchiveItem and describes a single item in the archive. TAbTarArchive is derived from TAbArchive and describes the archive as a whole.

The AbZLTyp unit implements the classes that describe a ZLib compatible data stream and the contents of that stream. TAbZLibStream is derived from TStream and understands the internal layout of ZLib compressed data; it can create a new ZLib formatted stream from uncompressed data; or decompress a pre-existing ZLib formatted stream.

The AbCabTyp unit implements the classes that describe a cabinet archive and the files contained in a cabinet. TAbCabItem is derived from TAbArchiveItem and describes a single file in the cabinet. TAbCabArchive is derived from TAbArchive and describes the cabinet archive as a whole.

---

## TAbArchiveItem Class

The TAbArchiveItem class describes a single item in an archive.

### Hierarchy

TObject

TAbArchiveItem (AbArcTyp)

### Properties

Action	ExternalFileAttributes	LastModTimeAsDateTime
CompressedSize	FileName	StoredPath
CRC32	IsEncrypted	Tagged
DiskFileName	LastModFileDialog	UncompressedSize
DiskPath	LastModFileType	

### Methods

MatchesDiskName                    MatchesStoredName

## Reference Section

---

Action	run-time property
--------	-------------------

```
property Action : TAbArchiveAction  
TAbArchiveAction = (  
  aaFailed, aaNone, aaAdd, aaDelete,  
  aaFreshen, aaMove, aaStreamAdd);
```

↳ Describes the pending process for this archive item.

If the TAbArchive that includes this TAbArchiveItem has its AutoSave property set to False, then processing of archive actions is deferred until either:

- The archive is explicitly saved using TAbArchive's Save method.
- Multiple operations on a single item require the archive to be saved.

The Action property describes the processing that needs to occur on this archive. During the Save operation on the archive, each TAbArchiveItem's Action property is checked, and pending operations occur. The archive's action is then reset to aaNone.

See also: TAbArchive.Save

---

CompressedSize	run-time property
----------------	-------------------

```
property CompressedSize : LongInt
```

↳ The size in bytes of the file in its compressed state.

CompressedSize is the actual size of the file within the archive. It must be set by the routine that writes the archive. If the file is encrypted, the size of the encryption header is included in Compressed Size.

See also: UncompressedSize

---

CRC32	run-time property
-------	-------------------

```
property CRC32 : Longint
```

↳ A 32-bit CRC for the original file represented by this item in the archive.

The CRC of the original file is stored in the header information for a zipped file to provide a verification check that the item was properly dearchived. Two files with the same CRC and the same size are almost certainly identical.

See also: LastModFileDate, LastModFileTime, UnCompressedSize

## DiskFileName

run-time property

```
property DiskFileName : string
```

- ☞ The file's complete file name as stored on the disk.

This may differ from the FileName property, which contains the file specification stored in the archive.

● **Caution:** DiskFileName may not be stored in the archive. PKZIP files do not store this information. When a PKZIP archive is first opened, the DiskFileName of each item in the archive will contain the same information found in the FileName property.

See also: DiskPath, MatchesDiskName

## DiskPath

run-time property

```
property DiskPath : string
```

- ☞ The path information of a file that is to be stored in the archive.

DiskPath is obtained by extracting the path information from DiskFileName.

See also: DiskFileName, MatchesDiskName

## ExternalFileAttributes

run-time property

```
property ExternalFileAttributes : LongInt
```

- ☞ Supplies the external file system attributes for the item.

ExternalFileAttributes is made up from the SysUtils unit file attribute constants.

### Windows

The options are faReadOnly, faHidden, faSysFile, faVolumeID, faDirectory, and faArchive. These attributes can be combined by adding their constants or values. For example: (faReadOnly + faHidden) indicates a file with the read-only and hidden attributes set.

### Linux

The options are faLinuxDir, faFileLink, faChrFile, faBlkFile, faFifoFile, and faSockFile. These attributes can be combined by adding their constants or values. For example the Kylix definition of the faSysFile constant specifies "faChrFile + faBlkFile + faFifoFile + faSockFile" which is useful for searching for various types, however, for an individual file stored in an archive, these values are mutually exclusive.

**FileName**run-time property

---

```
property FileName : string
```

↳ The name under which the archived file is stored.

See also: MatchesStoredName, StoredPath

**IsEncrypted**run-time property

---

```
property IsEncrypted : Boolean
```

↳ Indicates whether the item is encrypted.

**LastModFileDialog**run-time property

---

```
property LastModFileDialog : Word
```

↳ Indicates the last date the compressed file was modified.

The LastModFileDialog property supplies the DOS file date (as is stored in a ZIP compatible file) for the archived file.

This property is provided for backward compatibility with earlier versions of Abbrevia. To access the file date in a platform independent manner use the LastModTimeAsDateTime property.

See also: LastModFileType

**LastModFileType**run-time property

---

```
property LastModFileType : Word
```

↳ Indicates the last time the compressed file was modified.

The LastModFileDialog property supplies the DOS file time (as is stored in a ZIP compatible file) for the archived file.

The last modified date and time for the compressed file can be converted to a string by:

```
DateTimeToStr(  
  FileDateToDateTime(LastModFileDialog shl 16 + LastModFileType));
```

This property is provided for backward compatibility with earlier versions of Abbrevia. To access the file time in a platform independent manner use the LastModTimeAsDateTime property.

See also: LastModFileDialog

## LastModTimeAsDateTime

run-time property

```
property LastModTiemAsDateTime : TDateTime
```

- ↳ Indicates the date and time of the last modification to the compressed file.

The LastModTimeAsDateTime property specifies a TDateTime value indicating the file system date and time of last modification as was reported at the time of archiving.

See also: LastModFileDialog, LastModFileTime

## MatchesDiskName

method

```
function MatchesDiskName(const FileMask : string) : Boolean;
```

- ↳ Returns True if the item's DiskFileName matches the specified FileMask.

FileMask can contain '\*' and '?' wild characters, and may contain a path as well as a file name. If the FileMask includes a directory, then the DiskFilePath must match the directory portion of the FileMask for MatchesDiskName to return True.

See also: DiskFileName, DiskPath

## MatchesStoredName

method

```
function MatchesStoredName(const FileMask : string) : Boolean;
```

- ↳ Returns True if the item's Filename matches the specified FileMask.

FileMask can contain '\*' and '?' wild characters, and may contain a path as well as a file name. If the FileMask includes a directory, then the FilePath must match the directory portion of the FileMask for MatchesStoredName to return True.

See also: Filename, StoredPath

## StoredPath

run-time property

```
property StoredPath : string
```

- ↳ Returns the path stored in the compressed file's header information.

As each file is stored in an archive, information relative to that file is also stored. This can optionally include path information. If path information is included, it can be accessed via StoredPath. If no path information is included, StoredPath returns an empty string.

See also: Filename, MatchesStoredName

**Tagged****run-time property**

```
property Tagged : Boolean
```

↳ Selects items for Archive operations, such as Delete, Extract and Freshen.

Tagged archive items are ones that have been selected.

See also: TAbArchive.ClearTags, TAbArchive.TagItems, TAbArchive.UnTagItems,  
TAbArchive.DeleteTaggedItems, TAbArchive.ExtractTaggedItems,  
TAbArchive.FreshenTaggedItems.

**UncompressedSize****run-time property**

```
property UncompressedSize : LongInt
```

↳ The size in bytes of the file in its original, uncompressed state.

UncompressedSize is the actual size of the file when it is extracted from the archive.

See also: CompressedSize

# TAbArchive Class

The TAbArchive class is an abstract encapsulation of an archive.

## Hierarchy

TObject

TAbArchive (AbArcTyp)

## Properties

ArchiveName	ExtractOptions	Spanned
AutoSave	IsDirty	SpanningThreshold
BaseDirectory	LogFile	Status
Count	Logging	StoreOptions
DOSMode	Mode	TempDirectory

## Methods

Add	DeleteTaggedItems	FreshenFiles
AddFiles	Destroy	FreshenFilesEx
AddFilesEx	Extract	FreshenTaggedItems
AddFromStream	ExtractAt	Load
ClearTags	ExtractFiles	Move
Create	ExtractFilesEx	Replace
CreateFromStream	ExtractTaggedItems	Save
Delete	ExtractToStream	TagItems
DeleteAt	FindFile	TestTaggedItems
DeleteFiles	FindItem	UnTagItems
DeleteFilesEx	Freshen	

10

## Events

OnArchiveItemProgress	OnConfirmProcessItem	OnProcessItemFailure
OnArchiveProgress	OnConfirmSave	OnRequestImage
OnConfirmOverwrite	OnLoad	OnSave

# Reference Section

## Add

## method

```
procedure Add(aItem : TAbArchiveItem);
```

↳ Adds an ArchiveItem to the archive.

The archive assumes responsibility for destroying the archive item.

See also: AddFiles, AddFilesEx

## AddFiles

## method

```
procedure AddFiles(const FileMask : string; SearchAttr : Integer);
```

↳ Adds the files that match FileMask to the archive.

The wild card characters '\*' and '?' are allowed in the FileMask.

When AddFiles is searching for files that match FileMask, it searches certain directories, depending on the StoreOptions and BaseDirectory properties. The following table shows, for a given FileMask and set of StoreOptions, which directories are searched for the file. If the file is found, it is stored in the archive with the name shown in the last column.

The following table shows example of FileMask on Windows:

StoreOptions	FileMask	Directories Searched	Name in Archive
[ ]	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
[ soRecurse, soStripPath ]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TEXT
	X:\DOC\README.TEXT	X:\DOC	README.TEXT
[ soRecurse ]	README.TEXT	BASEDIRECTORY and all subdirectories	README.TEXT or SUBDIR/README.TEXT
	DOC\README.TEXT	BASEDIRECTORY\DOC	DOC/README.TEXT

<b>StoreOptions</b>	<b>FileMask</b>	<b>Directories Searched</b>	<b>Name in Archive</b>
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
[soStripPath]	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT
	X:\DOC\README.TXT	X:\DOC	README.TXT

The following table shows example of FileMask on Linux:

<b>StoreOptions</b>	<b>Linux FileMask</b>	<b>Directories Searched</b>	<b>Name in Archive</b>
[]	README.TXT	BASEDIRECTORY	README.TXT
	DOC/README.TXT	BASEDIRECTORY/DOC	DOC/README.TXT
	/DOC/README.TXT	/DOC	DOC/README.TXT
[soRecurse, soStripPath]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT
	DOC/README.TXT	BASEDIRECTORY/DOC	README.TXT
	/DOC/README.TXT	/DOC	README.TXT
[soRecurse]	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT or SUBDIR/README.TXT
	DOC/README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	/DOC/README.TXT	/DOC	DOC/README.TXT
[soStripPath]	README.TXT	BASEDIRECTORY	README.TXT
	DOC/README.TXT	BASEDIRECTORY/DOC	README.TEXT
	/DOC/README.TEXT	/DOC	README.TEXT

Use the SearchAttr parameter if you want to include special files, such as system and hidden files in the archive. If SearchAttr is zero, only normal files that match FileMask are added. If SearchAttr is set to one of the file attribute constants (faReadOnly, faHidden, faSysFile, faVolumeID, faDirectory, faArchive) defined in SysUtils, the appropriate type of special files are also added. To add multiple types of special files, add the file attribute constants together. For example, to search for read-only and hidden files in addition to normal files, pass (faReadOnly + faHidden) as the SearchAttr parameter.

See also: BaseDirectory, OnProcessItemFailure, StoreOptions

## AddFilesEx

method

```
procedure AddFilesEx(const FileMask, ExclusionMask : string;
  SearchAttr : Integer);
```

↳ Adds files matching FileMask except those matching ExclusionMask.

AddFilesEx introduces item exception list functionality to AddFiles. By specifying ExclusionMask you can exclude certain files that would normally be added using the AddFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could add all files except text files by the following:

### Windows

```
AddFilesEx('*.*', '*.txt');
```

### Linux

```
AddFilesEx('*', '*.txt');
```

See also: AddFiles

10

## AddFromStream

method

```
procedure AddFromStream(const NewName : string; aStream : TStream);
```

↳ Creates an archive item directly from a stream.

AddFromStream will create a zip item by compression data directly from a TStream instance. A file name must be provided via NewName for the new zip item that is created. The stream's position is set to 0 and the entire stream is added. Adding data directly from a stream causes the archive to be automatically saved since the stream object may be subsequently freed or modified.

---

<b>ArchiveName</b>	<b>run-time property</b>
--------------------	--------------------------

---

`property ArchiveName : string`

☞ The stored file name of the complete archive.

The archive's name is stored as ArchiveName. For new archives, the ArchiveName can be set using the Create constructor.

See also: Create

---

<b>AutoSave</b>	<b>run-time property</b>
-----------------	--------------------------

---

`property AutoSave : Boolean`

Default: False

☞ Controls whether changes to the archive are performed immediately.

When AutoSave is True, each change to the archive forces the disk image of the archive to be updated immediately. When AutoSave is False, changes to the disk image are done when one of the following occur:

- The archive is explicitly saved using the Save method.
- Multiple operations on a single item require the archive to be saved.
- The archive is closed and the archive has changed since it was last saved.

10

**Note:** When AutoSave is True, only a single archive file may be created. Multiple archive file spanning requires that AutoSave is False. Also, keep in mind that saving the archive each time a file is added can be very time consuming. For these reasons, it is recommended that AutoSave be used only with relatively small archives.

See also: OnConfirmSave, Save

## **BaseDirectory**

**run-time property**

```
property BaseDirectory : string
```

☞ The default path for add and extract operations on the archive.

BaseDirectory is not necessarily the same as the directory of the archive.

When you add a file to an archive, if FileName is a relative file specification (e.g., “README.TXT” or “EXAMPLES\README.TXT”), BaseDirectory is used as the starting point to search for the file. When you extract a file from an archive, it is extracted to the current BaseDirectory or a subdirectory of the current BaseDirectory.

See the AddFiles, FreshenFiles, FreshenTaggedItems, ExtractFiles, and ExtractTaggedItems methods for information on how BaseDirectory is used in each case.

## **ClearTags**

**method**

```
procedure ClearTags;
```

☞ Clears tags from all items in the archive.

See also: TagItems

## **Count**

**run-time property**

```
property Count : Integer
```

☞ Returns the number of items available from the Items property.

☞ **Caution:** Count returns the number of items in the memory representation of the archive. If AutoSave is False, and items have been added or deleted from the archive, the number of items in the memory representation can be different than the number of items in the disk file or stream.

See also: AutoSave

## **Create**

**constructor**

```
constructor Create(const FileName : string; Mode : Word);
```

☞ Constructs an archive that internally creates a TFileStream object with the given FileName and Mode.

Mode can be set to any of the file mode constants (fmOpenRead, fmOpenWrite, etc.) defined in the SysUtils unit.

See also: CreateFromStream, Mode

## CreateFromStream

constructor

```
constructor CreateFromStream(  
  aStream : TStream ; const aArchiveName : string);
```

↳ Creates the archive from an existing TStream instance and provides a name for the Archive.

See also: Create, Mode

## Delete

method

```
procedure Delete(aItem : TABArchiveItem);
```

↳ Marks the archive item as deleted.

During the next Save operation, the related information is deleted from the archive, and the ArchiveItem is destroyed.

See also: DeleteFiles, DeleteTaggedItems

## DeleteAt

method

```
procedure DeleteAt(Index : Integer);
```

↳ Deletes the item at the specified item index.

DeleteAt can be used to delete an item from an archive when its index is known. Index is the zero-based item index.

10

## DeleteFiles

method

```
procedure DeleteFiles(const FileMask : string);
```

↳ Deletes the specified files from the archive.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is marked to be deleted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” marks all files in the archive that have a TXT extension to be deleted. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be deleted.

The next time the archive is saved, all marked files and their corresponding information are deleted from the archive.

See also: DeleteTaggedItems, Save

## DeleteFilesEx

method

```
procedure DeleteFilesEx(const FileMask, ExclusionMask : string);
```

- ↳ Delete files matching FileMask but not ExclusionMask.

DeleteFilesEx introduces item exception list functionality to DeleteFiles. By specifying ExclusionMask you can exclude certain files that would normally be deleted using the DeleteFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could delete all files from the archive except text files by the following:

### Windows

```
DeleteFilesEx('*.*', '*.txt');
```

### Linux

```
DeleteFilesEx('*', '*.txt');
```

See also: DeleteFiles

## DeleteTaggedItems

method

```
procedure DeleteTaggedItems;
```

- ↳ Deletes all tagged files from the archive.

DeleteTaggedItems finds all files that have their Tagged property set to True and marks them to be deleted. The next time the archive is saved, all marked files and their corresponding information are deleted from the archive.

See also: ClearTags, DeleteFiles, Save, TagItems, UnTagItems

## Destroy

destructor

```
destructor Destroy;
```

- ↳ Destroys the archive.

If AutoSave is False, Destroy saves the archive then frees any allocated resources owned by the object.

```
property DOSMode : Boolean
```

- ↳ Forces all new files stored in an archive to have DOS-compatible file names.

Windows 95 and Windows NT introduced long file names, which were not available under older versions of Windows or DOS. The older operating systems cannot handle the long file names; they require a file name no longer than eight characters (8.3 format). If your archive must be read by an older operating system, you can force short file names by setting DOSMode to True.

### Windows

The shortened version of the file name is determined by the Windows GetShortPathName API call.

If DOSMode is True, a file named C:\PROGRAM FILES\README.TXT is stored in the archive as PROGRA~1/README.TXT.

### Linux

The shortened version of the file name is determined by a simple algorithm similar to what is used on Windows to generate short file names.

If DOSMode is True, a file named /home/mydir/longdirname/SomeLongFileName will be stored in the archive as something like home/mydir/longd~01/SomeL~01.

---

## Extract

## method

```
procedure Extract(aItem : TAbArchiveItem; const NewName : string);
```

- ↳ Creates the output file for the specified archive item using NewName, if specified.

The file in the archive is not modified by extract operations. Extract happens immediately. If NewName is left blank then the item's stored file name will be used.

See also: ExtractAt, ExtractFiles, ExtractFilesEx, ExtractTaggedItems

---

## ExtractAt

## method

```
procedure ExtractAt(Index : Integer; NewName : string);
```

- ↳ Extracts an item from an archive when its item index is known.

Index is the zero-based item index. NewName is the file name to be given to the item if different than the stored file name. If NewName is left blank then the item's stored file name will be used.

```
procedure ExtractFiles(const FileMask : string);
```

↳ Extracts all files that match FileMask from the archive.

The files in the archive are not modified by extract operations. The extraction happens immediately; it is not affected by the setting of the AutoSave property.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is extracted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” extracts all files in the archive that have a TXT extension. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be extracted.

The following table shows some example FileMasks and stored file names and indicates whether they would be considered a match:

FileMask	Stored File Name	Match?
*.txt	test.txt	Yes
*.txt	doc/test.txt	Yes
*.txt	bin/test.txt	Yes
doc/*.txt	test.txt	No
doc/*.txt	doc/test.txt	Yes
doc/*.txt	bin/test.txt	No
*/*.txt	test.txt	No
*/*.txt	doc/test.txt	Yes
*/*.txt	bin/test.txt	Yes

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions property on page 220 for a description of how the directory is determined.

See the OnProcessItemFailure event on page 230 for a description of the exceptions that can occur during the extract process.

See also: BaseDirectory, ExtractOptions, ExtractTaggedItems, OnConfirmProcessItem, OnProcessItemFailure

## ExtractFilesEx

method

```
procedure ExtractFilesEx(const FileMask, ExclusionMask : string);
```

- ↳ Extracts files matching FileMask, but not ExclusionMask.

ExtractFilesEx introduces item exception list functionality to ExtractFiles. By specifying ExclusionMask you can exclude certain files that would normally be extracted using the ExtractFiles method.

For example, you could extract all files except text files by the following:

### Windows

```
ExtractFilesEx('*.*', '*.txt');
```

### Linux

```
ExtractFilesEx('*', '*.txt');
```

See also: ExtractFiles

## ExtractOptions

run-time property

```
property ExtractOptions : TAbExtractOptions  
TAbExtractOption = (eoCreateDirs, eoRestorePath);  
TAbExtractOptions = set of TAbExtractOption;
```

Default: [eoCreateDirs]

- ↳ Determines the options to use for extractions from archives.

If ExtractOptions contains eoCreateDirs, directories (including multiple levels of subdirectories) are created as needed to extract files.

If ExtractOptions contains eoRestorePath, path information stored in the archive is retained when files are extracted. The extracted file is placed in the appropriate subdirectory relative to the current BaseDirectory.

For a given file in the archive and a set of ExtractOptions, the following table shows, to which directory the file is extracted:

<b>ExtractOptions</b>	<b>Name in Archive</b>	<b>Directory for Extracted File</b>
[ ]	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
[ eoCreateDirs, eoRestorePath]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), which is created if necessary.
[ eoCreateDirs]	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY, which is created if necessary.
[ eoRestorePath]	README.TXT	BASEDIRECTORY, if it exists. Otherwise an exception is raised.
	DOC/README.TXT	BASEDIRECTORY\DOC (BASEDIRECTORY/DOC on Linux), if it exists. Otherwise, an exception is raised.

See also: [BaseDirectory](#), [ExtractFiles](#), [OnConfirmOverwrite](#), [OnProcessItemFailure](#)

## ExtractTaggedItems

method

```
procedure ExtractTaggedItems;
```

- ↳ Extracts all tagged files from the archive.

ExtractTaggedItems extracts all files that have their Tagged property set to True. The files in the archive are not modified by extract operations. The extraction happens immediately.

The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See the ExtractOptions property on page 220 for a description of how the directory is determined.

See the OnProcessItemFailure event on page 230 for a description of the exceptions that can occur during the extract process.

See also: BaseDirectory, ExtractFiles, ExtractOptions, OnConfirmProcessItem, OnProcessItemFailure, TagItems

## ExtractToStream

method

```
procedure ExtractToStream(
  const aFileName : string; aStream : TStream);
```

- ↳ Extract specified file directly to a stream.

ExtractToStream will extract an item directly to a TStream instance. The instance must already be created. The stream instance is not cleared and the extracted file is appended at the stream's current position.

10

## FindFile

method

```
function FindFile(const aFileName : string) : Integer;
```

- ↳ Searches the archive for a file with an exactly matching file name and returns the index of that item.

FindFile searches through the list of archive files until it finds one with an exact match for the specified file name. It then returns the index of that item. FindFile returns -1 if no match is found. No two items in an archive can have the same file name.

## FindItem

method

```
function FindItem(aItem : TABArchiveItem) : Integer;
```

- ↳ Searches the archive for the specified item and returns the index of that item.

FindItem searches through the list of archive items until it finds the one specified. It then returns the index of that item. FindItem returns -1 if no match is found.

## Freshen

## method

```
procedure Freshen(aItem : TAbArchiveItem);
```

- ↳ Searches the archive for an item with an exactly matching file name and sets its Action property as aaFreshen.

Freshening an item in an archive searches the default directory for a matching file. If found, and different from the one stored in the archive, it replaces the archived copy with the newer file. Freshen operations are performed when the archive is saved.

See also: AutoSave, FreshenFiles, FreshenTaggedItems, Save

## FreshenFiles

## method

```
procedure FreshenFiles(const FileMask : string);
```

- ↳ Freshens the specified files in the archive.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is marked to be freshened. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” marks all files in the archive that have a TXT extension to be freshened. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be freshened.

The next time the archive is saved, the marked files are freshened. If the file is found on disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file.

The directories that are searched to find the file depend on the StoreOptions and BaseDirectory properties. The following table shows, for a given file in the archive and a set of StoreOptions, what Windows directories are searched to find the file to use to freshen the file in the archive:

StoreOptions	Name in Archive	Directories Searched
[ ]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY\DOC
[ soRecurse, soStripPath ]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY\DOC
[ soRecurse ]	README.TXT	BASEDIRECTORY and all subdirectories

<b>StoreOptions</b>	<b>Name in Archive</b>	<b>Directories Searched</b>
	DOC/README.TXT	BASEDIRECTORY\DOC
[soStripPath]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY\DOC

The following table shows examples of Linux directories searched:

<b>StoreOptions</b>	<b>Name in Archive</b>	<b>Directories Searched</b>
[]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY/DOC
[soRecurse, soStripPath]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY/DOC
[soRecurse]	README.TXT	BASEDIRECTORY and all subdirectories
	DOC/README.TXT	BASEDIRECTORY/DOC
[soStripPath]	README.TXT	BASEDIRECTORY
	DOC/README.TXT	BASEDIRECTORY/DOC

See the OnProcessItemFailure event on page 230 for a description of the exceptions that can occur during the freshen process.

10

See also: FreshenFilesEx, FreshenTaggedItems, Save, StoreOptions

## FreshenFilesEx method

```
procedure FreshenFilesEx(const FileMask, ExclusionMask : string);
```

↳ Freshens files that match FileMask but not ExclusionMask.

FreshenFilesEx introduces item exception list functionality to FreshenFiles. By specifying ExclusionMask you can exclude certain files that would normally be freshened by using the FreshenFiles method. The wild card characters '\*' and '?' are allowed in the ExclusionMask as well as path information.

For example, you could freshen all files except text files by the following:

## Windows

```
FreshenFilesEx('*.*', '!*.txt');
```

## Linux

```
FreshenFilesEx('*', '!*.txt');
```

See also: FreshenFiles

## FreshenTaggedItems

method

```
procedure FreshenTaggedItems;
```

↳ Finds all files that have their Tagged property set to True and marks them to be freshened.

The next time the archive is saved, all marked files are freshened. If the file is found on the disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file.

The directories that are searched to find the file depend on the StoreOptions and BaseDirectory properties. See the table in the FreshenFiles method on page 223 for details of determining the directory to search.

See the OnProcessItemFailure event on page 230 for a description of the exceptions that can occur during the freshen process.

See also: FreshenFiles, OnProcessItemFailure, Save, StoreOptions, TagItems

## IsDirty

run-time property

```
property IsDirty : Boolean
```

↳ Indicates if the TAbArchive has been modified since it was last saved.

If IsDirty is True, TAbArchive has been modified since it was last saved. If IsDirty is False, TAbArchive has not been modified since it was last saved.

See also: AutoSave, Save

## **Load** method

---

```
procedure Load;
```

- ↳ Loads an existing archive.

Load first clears the contents of the TAbArchive, and then loads the TAbArchive from the File or Stream specified during Create.

See also: Create, CreateFromStream

## **LogFile** run-time property

---

```
property LogFile : string
```

- ↳ Specifies the text file to use for logging.

LogFile specifies the text file that will receive the log entries during archiving operations if logging has been enabled.

If the text file does not exist, it will be created.

See also: Logging

## **Logging** run-time property

---

```
property Logging : Boolean
```

Default: False

- ↳ Controls whether archive operations are recorded or not.

When Logging is True, each operation (add, delete, extract, freshen, move, replace) performed on an archive will create an entry in a text file. The text file is specified by the LogFile property. The following example shows the format of log entries:

```
D:\Test\Test.zip logging 04/27/1999 6:12:06 PM
FDI.H deleted 04/27/1999 6:12:37 PM
Ffdefine.inc added 04/27/1999 6:12:52 PM
```

## **Mode** run-time, read-only property

---

```
property Mode : Word
```

- ↳ Stores the file open mode of the attached archive.

Mode can be set to any of the file mode constants (fmOpenRead, fmOpenWrite, etc.) defined in the SysUtils unit.

See also: Create, CreateFromStream

## Move

## method

```
procedure Move(  
    aItem : TAbArchiveItem; const NewStoredPath : string);
```

↳ Renames an existing archive item.

Move modifies the stored file name of the archive item only in the memory representation of the archive. It marks the file to be moved (renamed). The next time the archive is saved, the stored file name is physically changed in the archive on disk.

See the OnProcessItemFailure event on page 230 for a description of the exceptions that can occur during the move process.

The following example renames the first item in an archive to NEWPATH/NAME.EXT:

```
Move(ABZipArchive1.Items[0], 'NEWPATH/NAME.EXT');
```

See also: AutoSave, Save

## OnArchiveItemProgress

## event

```
property OnArchiveItemProgress : TAbArchiveItemProgressEvent  
TAbArchiveProgressEvent = procedure(Sender : TObject;  
    Progress : Byte; var Abort : Boolean) of object;
```

↳ Defines an event handler that is called during long operations on a single archive item.

OnArchiveItemProgress is called by the Add, Extract, and Freshen operations on an archive item. It can be used to display the progress of the operation, or even to abort the operation. For example, you might want to display the name of the item being processed, the action being performed on the item, and an indication of the progress.

Progress is a percentage that indicates how far the operation has progressed through the archive item. Valid values for Progress are 0 to 100. If Abort is set to True in the event handler, the current operation is aborted.

See also: OnArchiveProgress

## OnArchiveProgress

event

```
property OnArchiveProgress : TAbArchiveProgressEvent  
TAbArchiveItemProgressEvent = procedure(  
  Sender : TObject; Item : TAbArchiveItem;  
  Progress : Byte; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called once per item when a process steps through the archive.

OnArchiveProgress is called during DeleteFiles, FreshenFiles, ExtractFiles, FileName (a change causes a new archive to be loaded), and Save archive operations. It can be used to display the progress of the operation, or even to abort the operation.

Progress is a percentage that indicates how far the current operation has progressed through the entire archive. Valid values for Progress are 0 to 100. If Abort is set to True in the event handler, the current operation is aborted.

See also: DeleteFiles, ExtractFiles, FreshenFiles, OnArchiveItemProgress, Save

## OnConfirmOverwrite

event

```
property OnConfirmOverwrite : TAbConfirmOverwriteEvent  
TAbConfirmOverwriteEvent = procedure(var FileName : string;  
  var Confirm : Boolean) of object;
```

10

- ↳ Defines an event handler that is called if the output file for an extract operation already exists.

This event handler is called prior to the start of the extract process if the expected output file already exists. If you set Confirm to False, the extract process is aborted for that item. If Confirm is True or you do not provide a handler for this event, the file is overwritten.

See also: ExtractFiles, ExtractTaggedItems

## OnConfirmProcessItem

event

```
property OnConfirmProcessItem : TAbArchiveItemConfirmEvent  
TAbArchiveItemConfirmEvent = procedure(  
  Sender : TObject; Item : TAbArchiveItem;  
  ProcessType : TAbProcessType; var Confirm : Boolean) of object;  
TAbProcessType = (ptAdd, ptDelete, ptExtract, ptFreshen, ptMove);
```

- ↳ Defines an event handler that is called before a process is performed on each item in the archive.

This event handler is called once for each item in an add, delete, extract, freshen, or move operation. If Confirm is set to False, the process is aborted. If a handler is not provided for this event, all processing continues as if Confirm were set to True.

The OnConfirmProcessItem event handler can be used to perform any actions that are necessary when the archive changes. For example, you might want to display a confirmation dialog or create a log of who modified the archive.

See also: AddFiles, DeleteFiles, DeleteTaggedItems, ExtractFiles, ExtractTaggedItems, FreshenFiles, FreshenTaggedItems, Move, ZipfileComment

## OnConfirmSave

event

```
property OnConfirmSave : TAbArchiveConfirmEvent  
TAbArchiveConfirmEvent = procedure(  
  Sender : TObject; var Confirm : Boolean) of object;
```

- ↳ Defines an event handler that is called before the archive is saved.

This event handler is called immediately before an archive is updated. If you set Confirm to False, the update is aborted. If a handler is not provided for this event, all processing continues as if Confirm were set to True.

The OnConfirmSave event can be used to allow the user to close an archive without saving changes. This is most useful when the AutoSave property is False. When the component's FileName property is changed, the Save method is automatically called, which fires the OnConfirmSave event. If the OnConfirmSave event handler sets Confirm to False, then the archive is not updated (all pending changes are discarded).

See also: AutoSave, Save

## OnLoad

event

```
property OnLoad : TABArchiveEvent  
TABArchiveEvent = procedure(Sender : TObject) of object;
```

Defines an event handler that is called immediately after an archive's contents are loaded.

Use OnLoad event to display the name of the current archive or to update a most-recently-used archive list.

See also: Load

## OnProcessItemFailure

event

```
property OnProcessItemFailure : TABArchiveItemFailureEvent  
TABArchiveItemFailureEvent = procedure(Sender : TObject;  
    Item : TABArchiveItem; const ProcessType : TABProcessType;  
    ErrorClass : TABErrorClass; ErrorCode : Integer) of object;  
  
TABProcessType = (  
    ptAdd, ptDelete, ptExtract, ptFreshen, ptMove, ptReplace);  
  
TABErrorClass = (ecAbbrevia, ecInOutError, ecFilerError,  
    ecFileCreateError, ecFileOpenError, ecOther);
```

Defines an event handler that is called when an exception is raised during an add, extract, freshen, move, or replace operation.

10

Abbrevia traps all exceptions that occur during add, extract, freshen, move, or replace operations. The exception is translated to an ErrorClass and ErrorCode. The OnProcessItemFailure is called and you can display a customized error message. Exceptions that are defined by Abbrevia are translated to an ErrorClass of ecAbbrevia. Abbrevia's exceptions (defined in the AbExcept unit) all have an ErrorCode property which uniquely identifies them. The ErrorCode constants are defined in the AbConst unit. You can retrieve an error string corresponding to the Abbrevia exception using the global AbStrRes variable. The following example shows how:

```
if ErrorClass = ecAbbrevia then  
    ShowMessage(AbStrRes[ErrorCode]);
```

If an event handler is not provided for the OnProcessItemFailure event, the user is not informed of the error. Abbrevia is designed to continue processing commands when an error is received without corrupting files or archives.

The following is a list of the possible Error Codes:

<b>ErrorCode</b>	<b>Action</b>	<b>Description</b>
AbDuplicateName	Add, move, replace	The file name already exists in the archive. You might want to display a warning to the user, but it can get ridiculous if the user is attempting to add the same large set of files to a directory twice. The file is not added or moved.
AbInvalidPassword	Extract	The specified password is not valid for extracting the encrypted file. The file is not extracted.
AbNoSuchDirectory	Extract	The destination directory does not exist. The file is not extracted.
AbUnknownCompressionMethod	Extract	The file was compressed with a compression method that is not supported by Abbrevia. The file is not extracted.
AbUserAbort	Add, extract, freshen, replace	The user aborted the process. The file is not processed.
AbZipBadCRC	Extract	The extracted file's CRC doesn't match the stored CRC. The extracted file is deleted.
AbZipVersionNeeded	Extract	The "Version needed to extract" for this file is not supported by this version of Abbrevia. This might occur in the future when a new version of PKZIP is available. The file is not extracted.

See also: AddFiles, ExtractFiles, ExtractTaggedItems, FreshenFiles, FreshenTaggedItems, OnConfirmProcessItem, Replace

## OnRequestImage

event

```
property OnRequestImage : TABRequestImageEvent  
TABRequestImageEvent = procedure(  
  Sender : TObject; ImageNumber : Word;  
  var ImageName : string; var Abort : Boolean) of object;
```

- ↳ Provides a mechanism to obtain a particular archive file name when working with a spanned set.

OnRequestImage provides a mechanism to obtain a particular archive file name when working with a spanned set. ImageName specifies the file of the request archive file within a spanned set. ImageNumber identifies the file's sequence number within the spanned set.

There are two conditions where this event will be fired—when saving an archive, and when extracting files from an archive.

When saving an archive, if the SpanningThreshold is reached, then the OnRequestImage event is fired to obtain the file name (via ImageName) of the next file in the spanned set. If an event handler has not been defined, then the image file name is automatically generated by replacing the last character of the file extension with a sequence number, (e.g. Files.Zip, Files.Z01, Files.Z02, etc.)

When extracting an item from a spanned archive set, OnRequestImage is fired to obtain the archive image containing the item. If the item spans archive images, then OnRequestImage is fired as often as required until the item has been extracted. If an event handler is not assigned, an exception will be raised when trying to extract from a spanned set.

See also: SpanningThreshold

10

## OnSave

event

```
property OnSave : TABArchiveEvent  
TABArchiveEvent = procedure(Sender : TObject) of object;
```

- ↳ Defines an event handler that is called immediately after the archive's contents are saved.

The OnSave event can be used along with the OnConfirmProcessItem event to display status information for the user. For example, in the OnConfirmProcessItem event handler, the status could be set to “change pending”, and in the OnSave event handler, the status could be set to “saved”. This would be most useful when the AutoSave property is False.

See also: OnConfirmProcessItem, OnConfirmSave, Save

**Replace****method**

```
procedure Replace(aItem : TAbArchiveItem);
```

↳ Replaces the specified item in the archive.

The next time the archive is saved, the item is replaced. If the file specified by the item's `FileName` property is found on disk, the compressed data in the archive is replaced with the compressed representation of the current file. If the file cannot be found, an `EAbFileNotFoundException` exception is raised.

See the `OnProcessItemFailure` event on page 230 for a description of the exceptions that can occur during the replacement process.

See also: `OnChange`, `Save`, `StoreOptions`

**Save****method**

```
procedure Save;
```

↳ Updates the archive.

If the archive has not been modified since it was last opened or saved, `Save` returns immediately without modifying the archive.

See also: `AutoSave`, `OnConfirmSave`

**Spanned****property**

```
property Spanned : Boolean
```

↳ Indicates whether the archive is part of a spanned set.

See also: `SpanningThreshold`

## **SpanningThreshold** property

---

```
property SpanningThreshold : Longint
```

Default: 0

↳ Specifies the maximum archive image size.

By specifying SpanningThreshold you can restrict the size of the archive. If the archive size reaches SpanningThreshold when adding or freshening items, the archive is closed and a new archive file is created and the process continues onto the new, spanned archive. This process is repeated until all the entire archive has been saved.

Setting SpanningThreshold to 0 specifies that a single archive file be created, up to MaxLongint bytes in size.

Spanning archive files in this fashion requires that a new file name be supplied each time a new archive file needs to be created. This can be done either by a file name via the OnRequestImage event, or allowing the file name to be automatically generated by appending a two digit sequence number to the original archive file name, (e.g. Files.Zip, Files.Z01, Files.Z02, etc.). There is a limit of 99 extension files that can be auto-generated. For larger spanned sets, you must provide an event handler to supply the file names.

See also: OnRequestImage

## **Status** run-time property

---

10

```
property Status : TABArchiveStatus
```

```
TABArchiveStatus = (asInvalid, asIdle, asBusy);
```

↳ Determines whether an operation is currently being performed on the archive.

When an archive is being initialized, Status is set to asInvalid. When the initialization is finished, Status is set to asIdle. During each operation that modifies the archive, Status is set to asBusy. Status is used internally by Abbrevia to prevent re-entrant operations, but could also be used to display archive activity.

```
property StoreOptions : TAbStoreOptions  
TAbStoreOptions = set of TAbStoreOption;  
TAbStoreOption = (soStripDrive, soStripPath, soRemoveDots,  
    soRecurse, soFreshen, soReplace);  
  
Default: [soStripDrive, soRemoveDots]
```

↳ Determines the options for archive add and freshen operations.

The following table defines the action taken for each option:

Option	Result
soStripDrive	Drive letter information is removed from the stored file name. ( <b>Note:</b> This option is ignored in Linux.)
soStripPath	All path information is removed from the stored file name.
soRemoveDots	All relative path information is removed from the stored file name. For example, if you call AddFiles with a FileMask of "..\TEST.TXT" ("..\TEST.TXT" in Linux), the parent of the current BaseDirectory is searched for a file named "TEST.TXT". If the file is found, it is stored as "TEST.TXT".
soRecurse	Subdirectories of the search path are included in the search for files to add or freshen.
soFreshen	When adding an existing item to the archive, the item is freshened.
soReplace	When adding an existing item to the archive, the item is replaced.

See also: AddFiles, FreshenFiles, FreshenTaggedItems

---

**TagItems** method

```
procedure TagItems(const FileMask : string);
```

- ↳ Tags each archive item whose stored name matches FileMask.

Abbrevia allows you to perform operations on multiple files in an archive. You can extract, freshen, or delete a group of files by first tagging them and then performing the desired operation.

Items can be untagged using ClearTags or UnTagItems. Operations can be performed on the tagged items using the DeleteTaggedItems, ExtractTaggedItems, and FreshenTaggedItems methods.

See also: ClearTags, ExtractTaggedItems, DeleteTaggedItems, FreshenTaggedItems, UnTagItems

---

**TempDirectory** run-time property

```
property TempDirectory : string
```

- ↳ Specifies a temporary directory to use during archive operations.

**Windows**

If this property is the empty string, the system's temporary directory will be used.

**Linux**

If this property is the empty string, the user's home directory will be used.

---

**TestTaggedItems** method

```
procedure TestTaggedItems;
```

- ↳ Performs integrity test on tagged items.

TestTaggedItems verifies the integrity of each tagged item in the archive. Each Tagged item is extracted to a special stream and the item's central directory record, local file header and CRC are checked. The stream is then discarded after the check occurs.

---

**UnTagItems** method

```
procedure UnTagItems(const FileMask : string);
```

- ↳ Sets the Tagged property to False for each archive item whose stored name matches FileMask.

See also: ClearTags, TagItems

---

# TAbArchiveStreamHelper Class

The TAbArchiveStreamHelper class specifies the abstract behaviors of a stream style interface onto an abstract archive. This includes methods for attaching the Helper to an arbitrary data stream, extracting or adding archived data and interpreting or setting item specific header data.

Individual descendants of TAbArchiveStreamHelper implement these methods in ways appropriate to the particular archive type.

## Hierarchy

TObject

TAbArchiveStreamHelper (AbArcTyp)

## Methods

Create	ReadHeader	WriteArchiveItem
ExtractItemData	ReadTail	WriteArchiveTail
FindFirstItem	SeekItem	
FindNextItem	WriteArchiveHeader	

## Reference Section

---

<b>Create</b>	<b>constructor</b>
---------------	--------------------

---

`Constructor Create(AStream : TStream);`

- ↳ Instantiates a new TAbArchiveStreamHelper instance.

The AStream parameter supplies a pre-existing data stream to which the TAbArchiveStreamHelper instance is to be attached.

---

<b>ExtractItemData</b>	<b>abstract method</b>
------------------------	------------------------

---

`procedure ExtractItemData(AStream : TStream);`

- ↳ Extracts raw item data from the attached stream.

The AStream parameter supplies a pre-existing data stream into which the raw item data is to be extracted. This performs no translation (e.g. decryption or decompression) on the data, that must be done externally.

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

---

<b>FindFirstItem</b>	<b>abstract method</b>
----------------------	------------------------

---

`function FindFirstItem : Boolean;`

- ↳ Locates the first archived item in the stream.

The FindFirstItem routine locates the first archived item in an archive, if it does not find a first item the routine returns False; otherwise, it returns True.

When an item is found the attached stream is positioned appropriately to start reading the item. Whether that constitutes needing to read a header or reading data will vary depending on the archive format.

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

## **FindNextItem**

**abstract method**

```
function FindNextItem : Boolean;
```

- ↳ The FindNextItem routine locates the subsequent archived item in an archive, if it does not find any further items the routine returns False; otherwise, it returns True.

When an item is found the attached stream is positioned appropriately to start reading the item. Whether that constitutes needing to read a header or reading data will vary depending on the archive format.

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

## **ReadHeader**

**abstract method**

```
procedure ReadHeader;
```

- ↳ Reads any header data associated with an archived item.

Descendants of TAbArchiveStreamHelper will typically have a data member representing the header for an archived item. Calling this method fills that member with data from the current item (as found using FindFirstItem, FindNextItem).

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

## **ReadTail**

**abstract method**

```
procedure ReadTail;
```

- ↳ Reads any trailing data associated with an archived item.

Descendants of TAbArchiveStreamHelper may have a data member representing the trailing information for an archived item. Calling this method fills that member with data from the current item (as found using FindFirstItem, FindNextItem).

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

## **SeekItem** abstract method

---

```
function SeekItem(Index : Integer) : Boolean;
```

- ↳ Attempts to locate a particular archived item in the stream.

The SeekItem method attempts to make a particular item current. The Index parameter indicates the desired item to be made current, if Index is out of range for the number of items in the archive (or there is some other problem with setting to that index) SeekItem returns False; otherwise, it returns True.

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

## **WriteArchiveHeader** abstract method

---

```
procedure WriteArchiveHeader;
```

- ↳ Writes any header data associated with an archived item.

Descendants of TAbArchiveStreamHelper will typically have a data member representing the header for an archived item. Such a data member should be filled with appropriate values prior to calling this method.

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

## **WriteArchiveItem** abstract method

---

```
procedure WriteArchiveItem(AStream : TStream);
```

- ↳ Writes the actual item data.

The AStream parameter supplies a pre-existing data stream from which the raw item data is to be added. This performs no translation (e.g. encryption or compression) on the data, that should have been performed prior to supplying the stream.

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

## **WriteArchiveTail**

**abstract method**

```
procedure WriteArchiveTail;
```

- ↳ Writes any trailing data associated with an archived item.

Descendants of TAbArchiveStreamHelper may have a data member representing the trailing information for an archived item. Such a data member should be filled with appropriate values prior to calling this method.

This is an abstract method and must be overridden by archive format specific descendants to implement appropriate behavior for the format.

---

# TAbsZipItem Class

The TAbZipItem class describes a single item in a Zip archive.

## Hierarchy

TObject

① TAbArchiveItem (AbArcTyp) .....	204
TAbsZipItem (AbZipTyp)	

## Properties

① Action	① DiskPath	① LastModTimeAsDateTime
① CompressedSize	① ExternalFileAttributes	RelativeOffset
CompressionMethod	ExtraField	ShannonFanoTreeCount
CompressionRatio	FileComment	① StoredPath
① CRC32	① FileName	① Tagged
Decoder	GeneralPurposeBitFlag	① UncompressedSize
DeflationOption	InternalFileAttributes	VersionMadeBy
DictionarySize	① IsEncrypted	VersionNeededToExtract
① DiskFileName	① LastModFileDialog	
DiskNumberStart	① LastModFileType	

10

## Methods

① MatchesDiskName	① MatchesStoredName
-------------------	---------------------

## Reference Section

---

CompressionMethod	run-time property
-------------------	-------------------

```
property CompressionMethod : TABZipCompressionMethod  
TABZipCompressionMethod = (cmStored, cmShrunk, cmReduced1,  
    cmReduced2, cmReduced3, cmReduced4, cmImploded, cmTokenized,  
    cmDeflated, cmEnhancedDeflated, cmDCLImploded, cmBestMethod );
```

↳ Describes the compression method used on this item in the archive.

The compression method field includes all the methods described in APPNOTE.TXT. Not all these methods are supported by Abbrevia, nor by PKZIP.

TABZipCompressionMethod is defined in AbZipTyp. To modify the CompressionMethod, you will need to add AbZipTyp to the uses clause of your unit.

See also: VersionNeededToExtract

---

CompressionRatio	run-time property
------------------	-------------------

```
property CompressionRatio : Double
```

↳ Returns the compression ratio (in percent) for this item in the archive.

The Compression ratio is calculated according to the following:

```
CompressionRatio = 100 * (1 - (CompSize / UncompressedSize))
```

Therefore, files that do not compress at all return a Compression Ratio of 0%, and files that compress to 1/3 their original size have a Compression Ratio of 67%.

**Note:** The compression ratio calculation first removes the 12 byte encryption header from the Compressed Size of password protected files. Thus, an encrypted, stored file returns a 0% CompressionRatio.

See also: TAbArchiveItem.CompressedSize, TAbArchiveItem.UnCompressedSize

---

Decoder	run-time property
---------	-------------------

```
property Decoder : TObject
```

↳ Returns a reference to the Decoder object created to handle encryption and decryption of an archive item using PKZIP compatible techniques.

The Decoder object is created and destroyed as needed during file processing.

See also: IsEncrypted

## DeflationOption run-time property

---

```
property DeflationOption : TAbZipDeflationOption  
TAbZipDeflationOption = (  
    doInvalid, doNormal, doMaximum, doFast, doSuperFast);
```

- Allows the user to select parameters for PKZip's Deflate algorithm.

The valid choices correspond to PKZip 2.04g's -en, -ex, -ef, and -es options. The parameter is ignored for all other compression methods.

TAbZipDeflationOption is defined in AbZipTyp. To modify the DeflationOption, you will need to add AbZipTyp to the uses clause of your unit.

See also: CompressionMethod

## DictionarySize run-time, read-only property

---

```
property DictionarySize : TAbZipDictionarySize  
TAbZipDictionarySize = (dsInvalid, ds4K, ds8K);
```

- Defines the size of the dictionary used to compress a file.

DictionarySize is stored in the GeneralPurposeBitFlag.

TAbZipDictionarySize is defined in AbZipTyp. To read the DictionarySize, you will need to add AbZipTyp to the uses clause of your unit. If the file was not Imploded, then DictionarySize is set to dsInvalid.

See also: CompressionMethod, GeneralPurposeBitFlag

## DiskNumberStart run-time property

---

```
property DiskNumberStart : Word
```

- Returns the first image (or removable disk) number within a spanned set.

If the archive consists of a spanned set, either in archive images or on removable disks, DiskNumberStart contains the spanned set sequence number of the first image (or removable disk) in the set. This sequence number is zero-based so if the item begins on the second image, for example, DiskNumberStart will be one.

**ExtraField**run-time property

---

```
property ExtraField : string
```

- ↳ Provides access to PKZip's ExtraField.

The ExtraField portion of a LocalFileHeader allows for expansion to the zip format.  
Abbrevia simply maintains the ExtraField information.

**FileComment**run-time property

---

```
property FileComment : string
```

- ↳ Provides access to an optional comment which can be stored for each file in an archive.  
Each file in a zip archive can have a different comment.

**GeneralPurposeBitFlag**run-time property

---

```
property GeneralPurposeBitFlag : Word
```

- ↳ Contains information about encryption, compression options, and the presence or absence of a data descriptor.

The general purpose flag is described in APPNOTE.TXT.

See also: DeflationOption, DictionarySize, IsEncrypted, ShannonFanoTreeCount

**InternalFileAttributes**run-time property

---

```
property InternalFileAttributes : Word
```

- ↳ Defines the type of file.

InternalFileAttributes is set to 0 if the file is apparently a binary file, or 1 if it is an Ascii or text file.

The InternalFileAttributes property is set by the compression process.

See also: ExternalFileAttributes

**RelativeOffset**run-time property

---

```
property RelativeOffset : LongInt
```

- ↳ The offset in bytes from the start of the first disk on which the file appears to the start of the local file header.

The zip file's Central Directory File Header stores the location of each archived item's data as an offset.

## ShannonFanoTreeCount

run-time, read-only property

```
property ShannonFanoTreeCount : Byte
```

- ↳ The number of Shannon-Fano trees used for imploded files.

If the compression method for an item was cmImplode, then this property returns the number of Shannon-Fano trees used during compression. The value can be two or three. This information is stored in the GeneralPurposeBitFlag.

See also: CompressionMethod, GeneralPurposeBitFlag

## VersionMadeBy

run-time property

```
property VersionMadeBy : Word
```

- ↳ The version of PKZIP that compressed or created the item.

The upper byte is not used by Abbrevia and is masked off internally. The lower byte of VersionMadeBy indicates the PKZIP version number that created the item. The value divided by 10 indicates the major version, and the value mod 10 indicates the minor version.

For example, if PKZIP 2.04 created the item, the value of VersionMadeBy is 24.

Abbrevia creates PKZIP version 2.0 equivalent zip files.

See also: VersionNeededToExtract

## VersionNeededToExtract

run-time property

```
property VersionNeededToExtract : Word
```

- ↳ The version number of PKZIP required to extract the file from the archive.

The upper byte is not used by Abbrevia and is masked off internally. The lower byte of VersionNeededToExtract indicates the PKZIP version number required to extract the item. The value divided by 10 indicates the major version, and the value mod 10 indicates the minor version.

This may differ from the VersionMadeBy. For example, PKZIP's Store compression method has been available since version 1.0. The VersionNeededToExtract for a stored file in a zip file is 1.0, even if it was written using PKZIP 2.0 or better.

For example, if the item was stored, VersionNeededToExtract is 10.

---

# TAbZipArchive Class

The TAbZipArchive class describes an entire Zip archive. In order to extract, insert, or test files from an archive, handlers for the ExtractHelper, InsertHelper, and TestHelper events must be supplied.

## Hierarchy

TObject

① TAbArchive (AbArcTyp).....	210
TAbZipArchive (AbZipTyp)	

## Properties

① ArchiveName	① ExtractOptions	① Spanned
① AutoSave	① IsDirty	① SpanningThreshold
① BaseDirectory	Items	① Status
CompressionMethodToUse	①LogFile	① StoreOptions
① Count	① Logging	① TempDirectory
CurrentDisk	① Mode	ZipFileComment
DeflationOption	Password	
① DOSMode	PasswordRetries	

## Methods

- ① Add
- ① AddFiles
- ① AddFilesEx
- ① AddFromStream
- ① ClearTags
- ① Create
- ① CreateFromStream
- ① Delete
- ① DeleteAt
- ① DeleteFiles
- ① DeleteFilesEx
- ① DeleteTaggedItems
- ① Destroy
- ① Extract
- ① ExtractAt
- ① ExtractFiles
- ① ExtractFilesEx
- ① ExtractTaggedItems
- ① ExtractToStream
- ① FindFile
- ① FindItem
- ① Freshen
- ① FreshenFiles
- ① FreshenFilesEx
- ① FreshenTaggedItems
- ① Load
- ① Move
- ① Replace
- ① Save
- ① TagItems
- ① TestTaggedItems
- ① UnTagItems

## Events

- ExtractHelper
- ExtractToStreamHelper
- InsertFromStreamHelper
- InsertHelper
- ① OnArchiveItemProgress
- ① OnArchiveProgress
- ① OnConfirmOverwrite
- ① OnConfirmProcessItem
- ① OnConfirmSave
- ① OnLoad
- OnNeedPassword
- ① OnProcessItemFailure
- OnRequestBlankDisk
- ① OnRequestImage
- OnRequestLastDisk
- OnRequestNthDisk
- ① OnSave
- TestHelper

## Reference Section

### **CompressionMethodToUse** property

---

```
property CompressionMethodToUse : TAbZipSupportedMethod
```

```
TAbZipSupportedMethod = (smStored, smDeflated, smBestMethod);
```

Default: smBestMethod

↳ Selects the compression method used when adding or freshening items in the archive.

The possible values for CompressionMethodToUse are:

Option	Description
SmStored	The file is stored without any compression.
SmDeflated	The file is compressed using the deflate method.
smBestMethod	The file is deflated, but if the resulting file is larger than the original, the file is simply stored.

### **CurrentDisk** property

---

```
property CurrentDisk : Word
```

↳ The sequence number of the current open archive image (or removable disk) number within a spanned set.

This sequence number is zero-based. So if the item begins on the second image, for example, DiskNumberStart will be one.

See also: OnRequestImage, SpanningThreshold

## DeflationOption

property

```
property DeflationOption : TAbZipDeflationOption  
TAbZipDeflationOption = (  
  doInvalid, doNormal, doMaximum, doFast, doSuperFast);
```

Default: doNormal

- ↳ Determines whether priority is given to compression or speed in the deflation.

DeflationOption allows you to select the deflation option when CompressionMethodToUse is smDeflated. You can select varying trade-offs between compression and speed. The choices are Maximum (most compression, slowest speed), Normal, Fast, and Super Fast (least compression, fastest speed). The choices correspond to the PKZIP 2.04g -ex, -en, -ef, and -es options.

TAbZipDeflationOption is defined in AbZipTyp. To modify the DeflationOption, you will need to add AbZipTyp to the uses clause of your unit.

See also: CompressionMethodToUse

## ExtractHelper

event

```
property ExtractHelper : TAbArchiveItemExtractEvent  
TAbArchiveItemExtractEvent = procedure(Sender : TObject;  
  Item : TAbArchiveItem; const NewName : string) of object;
```

10

- ↳ Defines an event handler that is called whenever a file is extracted from the archive.

The ExtractHelper event handler must be supplied in order for the ZipArchive to extract files. The event handler is responsible for extracting the specified item. Typically this is done by simply passing the information on to AbUnZip procedure defined in unit AbUnzPrc.

The following example extracts all files from an archive:

```
procedure UnZipHelper(Sender : TObject; Item : TAbArchiveItem;
                      const NewName : string);
begin
  AbUnzip(TAbZipArchive(Sender), TAbZipItem(Item), NewName);
end;

procedure ExtractFiles(ArchiveName : string);
var
  ZipArchive : TAbZipArchive;
begin
  ZipArchive := TAbZipArchive.Create(ArchiveName,
    fmOpenRead or fmShareDenyNone);
  try
    ZipArchive.Load;
    ZipArchive.ExtractHelper := UnZipHelper;
    ZipArchive.ExtractFiles('*.*');
  finally
    ZipArchive.Free;
  end;
end;
```

See also: ExtractToStreamHelper

## ExtractToStreamHelper

event

```
property ExtractToStreamHelper : TAbArchiveItemExtractToStreamEvent
  TAbArchiveItemExtractToStreamEvent = procedure(
    Sender : TObject; Item : TAbArchiveItem;
    OutStream : TStream) of object;
```

Defines an event handler that is called whenever an archive item is to be extracted directly to a TStream descendant.

The ExtractToStreamHelper event handler must be supplied in order for the ZipArchive to extract items directly to a stream. The event handler is responsible for extracting the specified item. Typically this is done by simply passing the information on to the AbUnZipToStream procedure defined in unit AbUnzPrc.

The following example extracts a specified file from an archive directly to a TMemoryStream:

```
procedure UnZipHelper(Sender : TObject; Item : TABArchiveItem;
                      OutStream : TStream);
begin
  if Assigned(OutStream) then
    AbUnzipToStream(TAbZipArchive(Sender), TABZipItem(Item),
                    OutStream);
end;

procedure ExtractFileToStream(
  const ArchiveName, FileName : string;
  ToStream : TMemoryStream);
var
  ZipArchive : TAbZipArchive;
begin
  ZipArchive := TABZipArchive.Create(ArchiveName,
                                      fmOpenRead or fmShareDenyNone);
  try
    ZipArchive.Load;
    ZipArchive.ExtractHelper := UnZipHelper;
    ZipArchive.ExtractToStream(Filename, ToStream);
  finally
    ZipArchive.Free;
  end;
end;
```

10

See also: ExtractHelper

## InsertFromStreamHelper

event

```
property InsertFromStreamHelper :
  TABArchiveItemInsertFromStreamEvent

TABArchiveItemInsertFromStreamEvent = procedure(
  Sender : TObject; Item : TABArchiveItem;
  OutStream, InStream : TStream) of object;
```

Defines an event handler that is called whenever an archive item is to be created directly from a TStream descendant.

The InsertFromStreamHelper event handler must be supplied in order for the ZipArchive to create items directly from a stream. InStream is the uncompressed TStream descendant, and OutStream is the compressed TStream descendant. The event handler is responsible for reading the uncompressed stream, compressing the data, and writing it out to the compressed stream. Typically this is done by simply passing the information on to the AbZipFromStream procedure defined in unit AbZipPrc.

The following example creates an item directly from a TMemoryStream:

```
procedure ZipHelper(Sender : TObject; Item : TAbArchiveItem;
                     OutStream, InStream : TStream);
begin
  if Assigned(InStream) then
    AbZipFromStream(TAbZipArchive(Sender), TAbZipItem(Item),
                     OutStream, InStream);
end;

procedure AddFileFromStream(
  const ArchiveName, NewName : string;
  FromStream : TMemoryStream);
var
  ZipArchive : TAbZipArchive;
begin
  ZipArchive := TAbZipArchive.Create(ArchiveName,
    fmOpenWrite or fmShareDenyNone);
  try
    ZipArchive.Load;
    ZipArchive.InsertFromStreamHelper := ZipHelper;
    FromStream.Position := 0;
    ZipArchive.AddFromStream(NewName, FromStream);
  finally
    ZipArchive.Free;
  end;
end;
```

See also: InsertHelper

10

## InsertHelper

event

```
property InsertHelper : TAbArchiveItemInsertEvent
  TAbArchiveItemInsertEvent = procedure(
    Sender : TObject; Item : TAbArchiveItem;
    OutStream : TStream) of object;
```

Defines an event handler that is called whenever a file is inserted to the archive.

The InsertHelper event handler must be supplied in order for the ZipArchive to add files to the archive. OutStream is the TStream descendant that receives the compressed file data. The event handler is responsible for reading the file, compressing the data, and writing it out to the compressed stream. Typically this is done by simply passing the information on to the AbZip procedure defined in unit AbZipPrc.

The following example adds files to an archive:

```
procedure ZipHelper(Sender : TObject; Item : TAbArchiveItem;
                     OutStream : TStream);
begin
  AbZip(TAbZipArchive(Sender), TABZipItem(Item), OutStream);
end;

procedure AddFiles(const ArchiveName : string);
var
  ZipArchive : TAbZipArchive;
begin
  ZipArchive := TABZipArchive.Create(ArchiveName,
    fmOpenWrite or fmShareDenyNone);
  try
    ZipArchive.Load;
    ZipArchive.InsertHelper := ZipHelper;
    ZipArchive.AddFiles('*.*', 0);
  finally
    ZipArchive.Free;
  end;
end;
```

See also: InsertFromStreamHelper

---

Items	property
-------	----------

10

property Items[Index : Integer] : TABZipItem

↳ Provides a list of the file names for each item in the archive.

Each item in a zip archive is described using a TABZipItem. See the “TABZipItem Class” on page 242. Valid values for Index are 0 through Count -1.

The following example accesses the name of each file in the archive:

```
if (Count > 0) then
  for I := 0 to pred(Count) do
    ListBox1.Items.Add := AbZipArchive1[I].FileName;
```

See also: TABArchive.Count

```
property OnNeedPassword : TAbNeedPasswordEvent  
  TAbNeedPasswordEvent = procedure(  
    Sender : TObject; var NewPassword : string) of object;
```

Defines an event handler that is called to allow entry of a password when decrypting a file.

The OnNeedPassword event handler is called when an encrypted file is being extracted and one of the following occurs:

- The Password property is empty.
- The Password is not valid for the encrypted file and the number of attempts is less than PasswordRetries.

The event does not change the value of the Password property.

The following example displays the password dialog that is included in the AbDlgPwd unit. It allows the user to enter a new password.

```
uses  
  AbDlgPwd;  
  
procedure TForm1.AbZipArchive1NeedPassword(  
  Sender : TObject; var NewPassword : string);  
  
var  
  Dlg : TPassWordDlg;  
begin  
  with TPassWordDlg.Create(Application) do begin  
    try  
      if (ShowModal = mrOK) then  
        NewPassword := Edit1.Text;  
    finally  
      Free;  
    end;  
  end;  
end;
```

See also: Password, PasswordRetries

## OnRequestBlankDisk

event

```
property OnRequestBlankDisk : TAbRequestDiskEvent  
TAbRequestDiskEvent = procedure(  
  Sender : TObject; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called when a blank, removable disk is needed for a spanned archive.

If you do not supply an OnRequestBlankDisk event handler, a dialog box is displayed to prompt the user for a blank disk. If you want to add features such as formatting the disk, scanning the disk for errors, or allowing the user to verify the disk contents, you can do that in an OnRequestBlankDisk event handler. You can even abort spanning by setting Abort to True inside the event handler.

See also: [OnRequestLastDisk](#), [OnRequestNthDisk](#)

## OnRequestLastDisk

event

```
property OnRequestLastDisk : TAbRequestDiskEvent  
TAbRequestDiskEvent = procedure(  
  Sender : TObject; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called when the last removable disk of a spanned archive is needed.

The directory information for PKZIP files is stored on the last disk of a spanned archive. If you do not supply an OnRequestLastDisk event handler, a dialog box is displayed to prompt the user for the last disk. If you want to add features such as allowing the user to verify the disk contents, you can do that in an OnRequestLastDisk event handler. You can even abort reading the spanned archive by setting Abort to True inside the event handler.

See also: [OnRequestBlankDisk](#), [OnRequestNthDisk](#)

## OnRequestNthDisk

event

```
property OnRequestNthDisk : TAbRequestNthDiskEvent  
TAbRequestNthDiskEvent = procedure(Sender : TObject;  
DiskNumber : Byte; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called when a specific removable disk in the spanned archive is needed.

If you do not supply an OnRequestNthDisk event handler, a dialog box is displayed to prompt for the disk specified by DiskNumber. If you want to add features such as allowing the user to verify the disk contents, you can do that in an OnRequestNthDisk event handler. You can even abort reading the spanned archive by setting Abort to True inside the event handler.

See also: [OnRequestBlankDisk](#), [OnRequestLastDisk](#)

## Password

property

```
property Password : string
```

- ↳ The password used for encrypting or decrypting a file.

When a file is added to the archive, Password is used to encrypt it. If Password is empty, the file is not encrypted. When an attempt is made to extract an encrypted file from the archive, Password is used to decrypt the encryption header. If the decryption is successful, the remainder of the file is decrypted and extracted. If the encryption header cannot be successfully decrypted or Password is empty, OnNeedPassword is called (if fewer than PasswordRetries attempts have been made). The OnNeedPassword event handler can then provide a password (possibly prompting the user).

See also: [OnNeedPassword](#), [PasswordRetries](#)

## PasswordRetries

## property

---

property PasswordRetries : Byte

Default: 3

- ↳ Specifies the maximum number of passwords to try when attempting to extract an encrypted file.

When the number of retries is exhausted, an exception is raised. See the OnProcessItemFailure event on page 230 for a description of how exceptions that occur during the extract process are handled.

If PasswordRetries is 0 and Password is the empty string, encrypted files cannot be extracted.

See also: Password

## TestHelper

## event

---

property TestHelper : TAbArchiveItemTestEvent

```
TAbArchiveItemTestEvent = procedure(  
  Sender : TObject; Item : TAbArchiveItem) of object;
```

- ↳ Defines an event handler that is called whenever an item needs to be tested.

The TestHelper event handler must be supplied in order for the ZipArchive to test files in the archive. The event handler is responsible for verifying the files integrity. Typically this is done by simply passing the information on to the AbTestZipItem procedure defined in unit AbZipPrc.

## ZipFileComment

## property

---

property ZipFileComment : string

- ↳ Provides access to the comment stored in the Zip archive.

A comment for the whole archive can be stored in PKZIP-compatible archives.

---

# TAbZipStreamHelper Class

The TAbZipStreamHelper class implements the methods of TAbArchiveStreamHelper appropriately for working with .ZIP archives. It exposes no additional methods, properties, or events.

## Hierarchy

TObject

TAbArchiveStreamHelper (AbArcTyp) .....	237
TAbZipStreamHelper (AbZipTyp)	

---

# TAbGZipItem Class

The TAbZipItem class describes a single item in a GZip archive.

## Hierarchy

TObject

① TAbArchiveItem (AbArcTyp) .....	204
TAbZipItem (AbGZTyp)	

## Properties

① Action	FileComment	IsHeaderCRCPresent
① CompressedSize	FileName	IsText
CompressionMethod	FileSystem	① LastModFileDialog
① CRC32	Flags	① LastModFileType
DiskFileName	HeaderCRC	① LastModTimeAsDateTime
DiskPath	IsEncrypted	① StoredPath
① ExternalFileAttributes	IsExtraFieldPresent	① Tagged
ExtraField	IsFileCommentPresent	① UncompressedSize
ExtraFlags	IsFileNamePresent	StoredPath

10

## Methods

① MatchesDiskName	① MatchesStoredName
-------------------	---------------------

## Reference Section

### CompressionMethod

**read-only property**

```
property CompressionMethod : Byte
```

- ↳ The compression method used by the GZip functionality.

The Compression property indicates which method was used to compress the data in the GZip archive. In the present implementation of GZip this should always return the value 8 indicating that the Deflate algorithm was used.

When creating Gzips, Abbrevia always uses Deflate and so sets the GZip header field reflected by this property to the value 8 automatically.

### DiskFileName

**run-time property**

```
property DiskFileName : string
```

- ↳ The file's complete file name as stored on the disk.

This may differ from the FileName property, which contains the file specification stored in the archive.

✿ **Caution:** A file name may not be stored in the archive at all; the GZip specification does not require that any file name information be included for the item stored in a GZip archive.

See also: DiskPath, FileName, StoredPath

### DiskPath

**run-time property**

```
property DiskPath : string
```

- ↳ The path information of a file that is to be stored in the archive.

DiskPath is obtained by extracting the path information from DiskFileName.

✿ **Caution:** A file name (and hence DiskPath) may not be stored in the archive at all; the GZip specification does not require that any file name information be included for the item stored in a GZip archive.

See also: DiskFileName, MatchesDiskName

## **ExtraField**

## **property**

`property ExtraField : string`

↳ Specifies “extra field” data stored with the GZip item.

The ExtraField portion of a GZip header allows for additional data to be stored with the archived file. This data is not required by the GZip specification and may consist of any sequence of bytes, so the string type may be misleading.

When extracting an item from a GZip archive, this property is set to any ExtraField data found in the GZip.

Setting this property prior to archiving an item causes that data to be stored with the item and the GZip header to be updated as needed to reflect the ExtraField’s presence.

See Also: Flags, IsExtraFieldPresent

## **ExtraFlags**

## **property**

`property ExtraFlags : Byte`

Default : 2

↳ The ExtraField property indicates the “extra field” associated with the archived item.

The currently defined values for this property are as follows:

<b>Value</b>	<b>Meaning</b>
2	Compressor used maximum compression, slowest algorithm
4	Compressor used fastest algorithm

When extracting a file from a GZip this property is set according to what is found in the GZip header.

When creating a GZip, setting this property will cause the Deflate engine to behave accordingly.

See also: Flags

## FileComment

property

property FileComment: string

↳ Provides access to an optional comment which can be stored for a file in an GZip archive.

The GZip specification says this information is optional and may not exist in a given archive.

Setting this property prior to archiving an item causes that FileComment to be stored with the item and the GZip header to be updated as needed to reflect the FileComment's presence.

See also: Flags, IsFileCommentPresent

## FileName

run-time property

property FileName : string

↳ The name under which the archived file is stored.

⚠ Caution: A file name may not be stored in the archive at all; the GZip specification does not require that any file name information be included for the item stored in a GZip archive.

Setting this property prior to archiving an item causes that FileName to be stored with the item and the GZip header to be updated as needed to reflect the FileName's presence.

See also: StoredPath

## FileSystem

property

property FileSystem : TAbGzFileSystem

```
TAbGzFileSystem = (
  osFat, osAmiga, osVMS, osUnix, osVM_CMS, osAtariTOS,
  osHPFS, osMacintosh, osZSystem, osCP_M, osTOPS20,
  osNTFS, osQDOS, osAcornRISCOS, osUnknown);
```

Default: osFat (Windows); osUnix (Linux)

↳ Indicates the file system on which the compressed files were created.

GZip provides a means to indicate which file system the GZipped file came from. This characteristic appears to be rarely used for anything but informational purposes in common GZip implementations, but is included here for completeness.

When, creating a GZip, FileSystem is set to osFat automatically on Windows and to osUnix on Linux.

Setting it to other values will cause that to be stored into the resulting GZip rather than the default, but has no other effect on the compression process.

**Note:** The GZip specification refers to the header field as “OS” or “Operating System.”

Flags	property
-------	----------

property Flags : Byte

↳ Supplies a set of flags representative of certain properties of a GZip archive.

The Flags property indicates whether certain specialized fields are present in a GZip archive's header data.

These options are specified by a bit field with the following values:

Bit	Abbrevia Constant	Significance if Set
0	AB_GZ_FLAG_FTEXT	Contents of GZip supposedly consists of compressed text (rarely used)
1	AB_GZ_FLAG_FHCRC	Header CRC is present
2	AB_GZ_FLAG_FEXTRA	Extra Field is present
3	AB_GZ_FLAG_FNAME	File Name is present
4	AB_GZ_FLAG_FCOMMENT	File Comment is present
5	n/a	Reserved
6	n/a	Reserved
7	n/a	Reserved

10

This bit field is supplied mainly in the interests of completeness.

When opening a GZip it's easier to check the values of the IsExtraFieldPresent, IsFileNamePresent, IsFileCommentPresent, IsHeaderCRCPresent, and IsText properties to determine whether or not these are set.

When creating a GZip, Abbrevia sets these fields appropriately depending on whether the ExtraField, FileComment, or FileName properties have been given values. Abbrevia always sets the HeaderCRC for the GZips it creates, and makes no distinction regarding whether a file is text or not.

See also: ExtraField, ExtraFlags, FileComment, FileName, HeaderCRC, IsExtraFieldPresent, IsFileNamePresent, IsFileCommentPresent, IsHeaderCRCPresent, IsText

## **HeaderCRC**

**read-only property**

**property HeaderCRC : Word**

↳ Supplies a validity check for a GZip's header information.

The GZip specification includes the possibility to have a CRC (Cyclic Redundancy Check) field for the GZip header data.

The GZip specification says this information is optional and may not exist in a given archive.

When creating a GZip, Abbrevia automatically generates and includes this field, and updates the GZip header as needed to reflect the HeaderCRC's presence.

See also: Flags, IsHeaderCRCPresent

## **IsEncrypted**

**run-time property**

**property IsEncrypted : Boolean**

↳ Indicates whether the item is encrypted.

The TAbGZipItem class always returns False for this property because GZip does not support item level encryption.

## **IsExtraFieldPresent**

**property**

**property IsExtraFieldPresent : Boolean**

↳ Specifies whether extra field data is present for the archived item.

See also: Flags, ExtraField

## **IsFileCommentPresent**

**property**

**property IsFileCommentPresent : Boolean**

↳ Specifies whether a FileComment is present for the archived item.

See also: Flags, FileComment

## **IsFileNamePresent**

**property**

**property IsFileNamePresent : Boolean**

↳ Specifies whether a file name is present for the archived item.

See also: Flags, AbArchiveItem.FileName

**IsHeaderCRCPresent****property**

```
property IsHeaderCRCPresent : Boolean
```

- ⌚ Specifies whether a header CRC is present for the archived item.

See also: Flags, HeaderCRC

**IsText****property**

```
property IsText : Boolean;
```

- ⌚ Specifies whether the item in the archive is supposed to be compressed text.

See also: Flags

**StoredPath****run-time property**

```
property StoredPath : string
```

- ⌚ Returns the path stored in the compressed file's header information.

As each file is stored in an archive, information relative to that file is also stored. This can optionally include path information. If path information is included, it can be accessed via StoredPath. If no path information is included, StoredPath returns an empty string.

- ⌚ **Caution:** A file name may not be stored in the archive at all; the GZip specification does not require that any file name information be included for the item stored in a GZip archive.

See also: FileName, MatchesStoredName

---

## TAbGZipArchive Class

The TAbGZipArchive class describes an entire GZip archive. The TAbGZipArchive class exposes no new properties, methods, or events over those supplied by TAbArchive.

### Hierarchy

TObject

TAbArchive (AbArcTyp) . . . . .	210
TAbGZipArchive(AbGZType)	

---

# TAbGZipStreamHelper Class

The TAbGZipStreamHelper class implements the methods of TAbArchiveStreamHelper appropriately for working with .GZIP archives. It exposes no additional methods, properties, or events.

## Hierarchy

TObject

TAbArchiveStreamHelper (AbArcType) .....	237
TAbGZipStreamHelper (AbGZType)	

---

# TAbTarItem Class

The TAbTarItem class describes a single item in a Zip archive.

## Hierarchy

TObject

❶ TAbArchiveItem (AbArcTyp) .....	204
TAbTarItem (AbTarTyp)	

## Properties

❶ Action	ExternalFileAttributes	LinkFlag
Checksum	❶ FileName	LinkName
CompressedSize	GroupID	❶ StoredPath
❶ CRC32	GroupName	❶ Tagged
DevMajor	IsEncrypted	UserID
DevMinor	❶ LastModFileDialog	UserName
❶ DiskFileName	❶ LastModFileType	UncompressedSize
❶ DiskPath	❶ LastModTimeAsDateTime	

## Methods

❶ MatchesDiskName	❶ MatchesStoredName
-------------------	---------------------

10

## Reference Section

---

<b>Checksum</b>	<b>read-only property</b>
-----------------	---------------------------

---

`property Checksum : Word`

↳ Provides the checksum value for the Tar item header.

When an item (typically a file) is added to a Tar archive, a checksum is calculated for the data in the Tar item's header. This serves as a validity check for the header.

When extracting items from a Tar this value reflects the checksum stored with the item. Abbrevia automatically checks the header against this checksum and will raise an `EAbTarBadChecksum` exception if the checksum is bad.

When adding items to a Tar, the checksum is calculated automatically by `TAbTarItem` so there's no need to set this property in that context.

---

<b>CompressedSize</b>	<b>run-time property</b>
-----------------------	--------------------------

---

`property CompressedSize : LongInt`

↳ The size in bytes of the file in its compressed state.

`CompressedSize` is the actual size of the file within the archive. It must be set by the routine that writes the archive. If the file is encrypted, the size of the encryption header is included in `CompressedSize`.

10

Note that for `TAbTarItem` `CompressedSize` will always equal `UncompressedSize` because Tar does not support item level compression.

See also: `UncompressedSize`

---

<b>DevMajor</b>	<b>property</b>
-----------------	-----------------

---

`property DevMajor : Word`

↳ Provides the “Major Device ID” from the Unix file system.

The standard Unix file system (which is also used by Linux) includes the notion that storage devices have identification numbers. This consists of two parts: the Major ID and the Minor ID.

On some Unix implementations it's important to know the device (or class of device) that created the archive, and so the Tar data includes information related to that. Modern Unix implementations (including Linux) generally seem to ignore this fields but, it is provided for backward compatibility.

When extracting files from a Tar, Abbrevia ignores this value.

If you set this property, the value will be stored with the Tar item, but it has no other effect on the archiving process.

See also: DevMinor

## DevMinor

property

```
property DevMinor : Word
```

↳ Provides the “Minor Device ID” from the Unix file system.

The standard Unix file system (which is also used by Linux) includes the notion that storage devices have identification numbers. This consists of two parts: the Major ID and the Minor ID.

On some Unix implementations it's important to know the device (or class of device) that created the archive, and so the Tar data includes information related to that. Modern Unix implementations (including Linux) generally seem to ignore this fields but, it is provided for backward compatibility.

When extracting files from a Tar, Abbrevia ignores this value.

If you set this property, the value will be stored with the Tar item, but it has no other effect on the archiving process.

See also: DevMajor

## ExternalFileAttributes

run-time property

```
property ExternalFileAttributes : LongInt
```

↳ Supplies the external file system attributes for the item.

ExternalFileAttributes is made up from the SysUtils unit file attribute constants; and (on Linux Tar files), additional bits indicating file permissions.

### Windows

The options are faReadOnly, faHidden, faSysFile, faVolumeID, faDirectory, and faArchive. These attributes can be combined by adding their constants or values. For example: (faReadOnly + faHidden) indicates a file with the read-only and hidden attributes set.

## Linux

The options are faLinuxDir, faFileLink faChrFile, faBlkFile, faFifoFile, and faSockFile. These attributes can be combined by adding their constants or values. For example the Kylix definition of the faSysFile constant specifies “faChrFile + faBlkFile + faFifoFile + faSockFile” which is useful for searching for various types however, for an individual file stored in an archive, these values are mutually exclusive.

Additionally the ExternalFileAttributes property allows determining the file system permissions for the file:

Abbrevia Constant	File Permission
AB_TAR_TSUID	Set UID on execution
AB_TAR_TSGID	Set GID on execution
AB_TAR_TSVTX	Save text (sticky bit)
AB_TAR_TUREAD	Read by owner
AB_TAR_TUWRITE	Write by owner
AB_TAR_TUEXEC	Execute/search by owner
AB_TAR_TGREAD	Read by group
AB_TAR_TGWRITE	Write by group
AB_TAR_TGEXEC	Execute/search by group
AB_TAR_TOREAD	Read by other
AB_TAR_TOWRITE	Write by other

10

For example, to check whether a stored file has the readable by group attribute set you could use some code like:

```
if (AbTarItem.ExternalAttributes and AB_TAR_TGREAD)
= AB_TAR_TGREAD then
```

GroupID	property
---------	----------

property GroupID : Word

Default: 0 (Windows); Current User's Group ID (Linux)

↳ Indicates the Unix file system “Group ID” for the archived file.

The standard Unix file system (which is also used by Linux) includes the notion of a user group to which a file belongs. The GroupID property supplies this value for a Tar item.

When extracting files on Windows this value is provided, but is otherwise ignored. When extracting files on Linux the extracted file's Group ID is set to the stored value.

When compressing files, changing this value from the default causes the value to be stored with the file as the Group ID, but has no other effect on the archiving process.

See Also: [GroupName](#), [UserID](#)

---

<b>GroupName</b>	<b>property</b>
------------------	-----------------

`property GroupName : string`

Default: “” (Windows); Current User's Group Name (Linux)

↳ Indicates the Unix file system “Group Name” for the archived file.

The standard Unix file system (which is also used by Linux) includes the notion of a user group to which a file belongs. Such groups can have symbolic names associated with them (e.g. “USERS”). The `GroupName` property supplies this value for a Tar item.

When extracting files on Windows this value is provided but is otherwise ignored. On Linux the `GroupID` property is used to determine the group that the extracted file is assigned to. Abbrevia does not ensure that the stored `GroupID` and `GroupName` match according to the file system. The programmer has the option and opportunity to perform such checks in the `OnProcessItem` event of a TAbArchive descendant.

When compressing files, changing this value from the default causes the value to be stored with the file as the Group Name, but has no other effect on the archiving process. In particular Abbrevia does not ensure that the value set to `GroupName` corresponds to that associated with the `GroupID` property according to the Linux file system.

See Also: [GroupID](#), [UserName](#)

---

<b>IsEncrypted</b>	<b>run-time property</b>
--------------------	--------------------------

`property IsEncrypted : Boolean`

↳ Indicates whether the item is encrypted.

The TAbTarItem class always returns False for this property because Tar does not support item level encryption.

```
property LinkFlag : char
```

Specifies the type of file system entity.

The standard Unix file system (which is also used by Linux) makes use of a number of different types of file system entries that may be stored in a Tar archive. Some of these may not have any actual stored data associated with them, but rather serve to describe the logical structure of the file system.

The codes for each type of link are as follows:

Abbrevia	Constant	Flag Character
AB_TAR_LF_OLDNORMAL	#0 (ASCII Null)	Normal disk file, Unix compatible
AB_TAR_LF_NORMAL	'0'	Normal disk file
AB_TAR_LF_LINK	'1'	Link to previously dumped file
AB_TAR_LF_SYMLINK	'2'	Symbolic link
AB_TAR_LF_CHR	'3'	Character special file
AB_TAR_LF_BLK	'4'	Block special file
AB_TAR_LF_DIR	'5'	Directory
AB_TAR_LF_FIFO	'6'	FIFO special file
AB_TAR_LF_CONTIG	'7'	Contiguous file

10

When extracting files on Windows, this property is set to indicate the type of item as it's processed. Abbrevia triggers events for all objects that may be included in the archive, but ignores any type except the AB\_TAR\_LF\_OLDNORMAL, AB\_TAR\_LF\_NORMAL, and AB\_TAR\_LF\_DIR flags however.

When extracting files on Linux this property is set to indicate the type of item as it's processed. Abbrevia triggers events for all objects that may be included in the archive, but ignores any type except the AB\_TAR\_LF\_OLDNORMAL, AB\_TAR\_LF\_NORMAL, AB\_TAR\_LF\_LINK, AB\_TAR\_LF\_SYMLINK, and AB\_TAR\_LF\_DIR flags.

When archiving files this property may be explicitly set for an item and that value will be stored with the item. Changing the default type may cause the item to be unreadable however.

When archiving files on Linux, the various special items are included in the file if they're specified, either explicitly or as part of a wildcard operation.

See also: LinkName

---

**LinkName** property

```
property LinkName : string;
```

↳ Specifies the referent entity for link file system entities.

The standard Unix file system (which is also used by Linux) makes use of a number of different types of file system entries that may be stored in a Tar archive. Some of these may not have any actual stored data associated with them, but rather serve to describe the logical structure of the file system.

File system links serve as aliases for other file system entities. When such a link is stored in a Tar, the name of its referent is also stored which is represented in Abbrevia by the LinkName property.

See also: LinkFlag

---

**UncompressedSize** run-time property

```
property UncompressedSize : LongInt
```

↳ The size in bytes of the file in its original, uncompressed state.

UncompressedSize is the actual size of the file when it is extracted from the archive.

Note that for TAbTarItem UncompressedSize will always equal CompressedSize because Tar does not support item level compression.

See also: CompressedSize

---

**UserID** property

```
property UserID : Word
```

Default: 0 (Windows); Current User's ID (Linux)

↳ Indicates the Unix file system "User ID" for the archived file.

The standard Unix file system (which is also used by Linux) includes the notion of a particular user to which a file belongs. The UserID property supplies this value for a Tar item.

When extracting files on Windows this value is provided but is otherwise ignored. When extracting files on Linux the extracted file's User ID is set to the stored value.

When compressing files, changing this value from the default causes the value to be stored with the file as the User ID, but has no other effect on the archiving process.

See Also: UserName, GroupID

<b>UserName</b>	<b>property</b>
-----------------	-----------------

---

`property UserName : string`

Default: “ (Windows); Current User’s UserName (Linux)

- ↳ Indicates the Unix file system “User Name” for the archived file. The standard Unix file system (which is also used by Linux) includes the notion of a particular user to which a file belongs. Users may be identified both by number and by a symbolic name (e.g. “NWIRTH” might be the username for a person named “Niklaus Wirth”). The UserName property supplies this value for a Tar item.

When extracting files on Windows this value is provided, but is otherwise ignored. On Linux the UserID property is used to determine the user that the extracted file is assigned to. Abbrevia does not ensure that the stored UserID and UserName match according to the file system.

When compressing files, changing this value from the default causes the value to be stored with the file as the User Name, but has no other effect on the archiving process. In particular, Abbrevia does not ensure that the value set to UserName corresponds to that associated with the UserID property according to the Linux file system. The programmer has the option and opportunity to perform such checks in the OnProcessItem event of a TAbArchive descendant.

See Also: GroupName, UserID

---

## TAbTarArchive Class

The TAbTarArchive class describes an entire Tar archive. The TABTarArchive class exposes no new properties, methods, or events over those supplied by TABArchive.

### Hierarchy

#### TObject

TAbArchive (AbArcTyp).....	210
TAbTarArchive (AbTarTyp)	

---

## TAbTarStreamHelper Class

The TAbTarStreamHelper class implements the methods of TAbArchiveStreamHelper appropriately for working with Tar archives. It exposes no additional methods, properties, or events.

### Hierarchy

TObject

TAbArchiveStreamHelper (AbArcTyp) .....	237
TAbTarStreamHelper (AbTarTyp)	

---

## TAbZLibStream Class

The TAbZLibStream class implements the methods of TAbArchiveStreamHelper appropriately for working with ZLib compressed data streams. It exposes no additional methods, properties, or events.

### Hierarchy

TObject

TAbArchiveStreamHelper (AbArcTyp) .....	237
TAbZLibStream (AbZLTyp)	

---

# TAbCabItem Class

The TAbCabItem class describes a single file in a cabinet.

## Hierarchy

TObject

① TAbArchiveItem (AbArcTyp) .....	204
TAbCabItem (AbCabTyp)	

## Properties

① Action	① ExternalFileAttributes	① LastModTimeAsDateTime
① CompressedSize	① FileName	PartialFile
① CRC32	① IsEncrypted	① StoredPath
① DiskFileName	① LastModFileDialog	① Tagged
① DiskPath	① LastModFileType	① UncompressedSize

## Methods

① MatchesDiskName	① MatchesStoredName
-------------------	---------------------

## Reference Section

### PartialFile

run-time property

```
property PartialFile : Boolean
```

↳ Indicates whether or not the file continues from a previous cabinet.

When PartialFile is False, the file is completely contained in the cabinet. If True, it indicates that the file is a continuation from a previous cabinet in a spanned set.

---

## TAbCabArchive Class

The TAbCabArchive is a low level class that encapsulates a cabinet archive and interfaces with the Microsoft CABINET.DLL to compress and extract files.

### Hierarchy

TObject

① TAbArchive (AbArcTyp).....	210
TAbCabArchive (AbCabTyp)	

### Properties

① ArchiveName	① ExtractOptions	① Logging
① AutoSave	FolderCount	Mode
① BaseDirectory	FolderThreshold	SetID
CabSize	HasNext	① Spanned
CompressionType	HasPrev	① SpanningThreshold
① Count	① IsDirty	① Status
CurrentCab	Items	① StoreOptions
① DOSMode	① LogFile	① TempDirectory

## Methods

- |                     |                      |                      |
|---------------------|----------------------|----------------------|
| ① Add               | ① Destroy            | ① FreshenTaggedItems |
| ① AddFiles          | ① Extract            | ① Load               |
| ① AddFilesEx        | ① ExtractAt          | ① Move               |
| ① AddFromStream     | ① ExtractFiles       | NewCabinet           |
| ① ClearTags         | ① ExtractFilesEx     | NewFolder            |
| Create              | ① ExtractTaggedItems | ① Replace            |
| ① CreateFromStream  | ① ExtractToStream    | ① Save               |
| ① Delete            | ① FindFile           | ① TagItems           |
| ① DeleteAt          | ① FindItem           | ① TestTaggedItems    |
| ① DeleteFiles       | ① Freshen            | ① UnTagItems         |
| ① DeleteFilesEx     | ① FreshenFiles       |                      |
| ① DeleteTaggedItems | ① FreshenFilesEx     |                      |

## Events

- |                         |                        |                        |
|-------------------------|------------------------|------------------------|
| ① OnArchiveItemProgress | ① OnConfirmProcessItem | ① OnProcessItemFailure |
| ① OnArchiveProgress     | ① OnConfirmSave        | ① OnRequestImage       |
| ① OnConfirmOverwrite    | ① OnLoad               | ① OnSave               |

## Reference Section

---

### CabSize read-only property

---

property CabSize : Longint

↳ Returns the size of the cabinet in bytes.

For existing cabinets, CabSize is static since the cabinet contents cannot be modified. When building a new cabinet, CabSize is updated after the folder (compression block) in progress has been completed and flushed to disk. This occurs either by a call to NewFolder or automatically when the FolderThreshold value has been reached.

See also: FolderThreshold, NewFolder

---

### CompressionType property

---

property CompressionType : TAbCabCompressionType

TAbCabCompressionType = (ctNone, ctMSZIP, ctLZX);

Default: ctMSZIP

↳ Specifies which type of compression to use.

CompressionType indicates which type of compression will be used when compressing a cabinet folder. CompressionType can vary from one folder to the next, unless the folder is a continuation from a previous cabinet.

10

---

### Create constructor

---

constructor Create(const FileName : string; Mode : Word);

↳ Constructs the cabinet archive.

This constructor creates an archive with the given FileName and Mode. FileName specifies the cabinet file name to be opened or built, and Mode identifies the read/write mode used.

Mode may be one of two, mutually exclusive values, fmOpenRead or fmOpenWrite. If Mode is fmOpenRead, then a FileDecompressionInterface (FDI) context is used to browse and extract items in an existing cabinet archive. A cabinet file opened with an FDI context is read-only and cannot be modified. All other file open modes, such as fmOpenReadWrite, are masked off.

For example, if Mode = 42 is passed in, it will be masked to 0 (fmOpenRead).

If Mode is fmOpenWrite, then a FileCompressionInterface (FCI) context is used to build a new cabinet archive. Items may be added to a cabinet archive when opened with an FCI context but existing items cannot be deleted or modified. Once a cabinet is closed, it cannot be reopened with an FCI context. Opening an existing cabinet file with an FCI context will delete the contents.

#### **CurrentCab**

**read-only property**

`property CurrentCab : Word`

↳ Returns the zero-based sequence number of the current cabinet within a multi-cabinet set.

If the cabinet is not part of a spanned set, CurrentCab will be zero.

#### **FolderCount**

**read-only property**

`property FolderCount : Word`

↳ Returns the number of folders (compression blocks) in the cabinet.

FolderCount also includes any partial folders if the cabinet is part of a spanned cabinet set.

#### **FolderThreshold**

**property**

`property FolderThreshold : Longint`

↳ The maximum cabinet folder (compression block) size.

Items in a cabinet archive can be compressed across their file boundaries as a single compression block. Such compression blocks are called folders. Compression ratios improve significantly when items are compressed together as opposed to individually. However, there is a trade-off between random access time to an individual item and compression ratio since an entire folder must be decompressed to extract an item from it.

Use FolderThreshold to specify the maximum size of a folder (compression block). When compressing files, this value will be used to determine when to stop the current compression block and start a new one. You can also start a new folder by calling NewFolder.

See also: NewFolder

## HasNext

**read-only property**

```
property HasNext : Boolean
```

↳ Indicates whether the last file in the cabinet spans into the next cabinet of a set.

If HasNext is True, then the current cabinet is part of a spanned cabinet set, and its last folder continues into the next cabinet.

## HasPrev

**read-only property**

```
property HasPrev : Boolean
```

↳ Indicates whether the first file in the cabinet spans from the previous cabinet of a set.

If HasPrev is True, then the current cabinet is part of a spanned cabinet set, and its first folder is a continuation from the previous cabinet.

## Items

**property**

```
property Items[Index : Integer] : TAbCabItem
```

↳ Provides a list of file names for each item in the cabinet.

Each item in a cabinet archive is described using a TAbCabItem. Valid values for Index are 0 through Count -1.

The following example accesses the name of each file in the archive:

10

```
if (Count > 0) then
  for I := 0 to pred(Count) do
    ListBox1.Items.Add := AbCabArchive1[I].FileName;
```

See also: TAbArchive.Count, TAbCabItem

## Mode

**read-only property**

```
property Mode : Word
```

↳ Indicates the read/write status of the cabinet.

Mode is assigned the value passed into the constructor and may be either fmOpenRead or fmOpenWrite.

If Mode is fmOpenRead, then the cabinet has been opened with an FDI context as read-only. In this mode, the cabinet cannot be modified but items can be extracted.

If Mode is fmOpenWrite, then the cabinet has been created with a FCI context as write-only. In this mode, Items may be added to the cabinet, but not modified or deleted.

See also: Create

**NewCabinet****method**

```
procedure NewCabinet;
```

- ☞ Flushes the current cabinet to disk and starts a new one.

Use NewCabinet to force the current cabinet under construction to be written to disk and start a new one. This is done automatically when the cabinet size reaches the value specified by SpanningThreshold. NewCabinet will fire the OnRequestImage event to obtain the new cabinet file name.

See also: OnRequestImage

**NewFolder****method**

```
procedure NewCabinet;
```

- ☞ Flushes the current folder and starts a new one.

Use NewFolder to force the current folder (compression block) to be completed and start a new one. The compression history is reset and a new compression block is started using the compression type specified by CompressionType. This is done automatically when the folder size reached the value specified by FolderThreshold.

See also: CompressionType, FolderThreshold

**SetID****property**

```
property SetID : Word
```

- ☞ Identifies the spanned cabinet set to which the current cabinet belongs, for use by the application.

Setting SetID when the cabinet is opened in an FDI context does not change the value.

See also Create, Mode



---

# Chapter 11: Miscellaneous Components, Classes, and Routines

This chapter describes the components, classes, and routines that do not fit neatly into any other category.

The following components are described:

- TAbMeter is a horizontal or vertical progress indicator that can be attached to an Abbrevia archive component to automatically display the progress of an archive action.
- TAbMakeSelfExe is a component that provides a simple mechanism for combining a zip archive and executable stub when building self-executable zip archives.

The following classes are described:

- TAbDirDlg provides a directory selection dialog.
- TAbPasswordDlg provides a password entry dialog.

The following routines are described:

- AbExistingZipAssociation checks to see if the .ZIP file extension is currently registered to an associated application.
- AbGetZipAssociation returns the path name of the application currently associated with the .ZIP file extension.
- AbRegisterZipExtension associates a specified application with the .ZIP file extension.
- DeflateStream compresses a stream but stores no directory information.
- InflateStream expands a stream compressed with DeflateStream.

---

# TAbMeter Component

The TAbMeter component is a specialized progress bar that can monitor an archive's progress event and display the state of the archive operation in progress. The Abbrevia archive components (TAbZipper, TAbCabExtractor, etc.) have two properties, ArchiveProgressMeter and ItemProgressMeter, to which a TAbMeter component can be assigned. A TAbMeter that is assigned to an archive's ArchiveProgressMeter property will monitor OnArchiveProgress events, and a TAbMeter that is assigned to an archive's ItemProgressMeter property will monitor OnArchiveItemProgress events.

You can configure the meter's size, orientation, and color.

## Hierarchy

TGraphicControl

  TAbCustomMeter (AbMeter)

    TAbMeter (AbMeter)

## Properties

  Orientation                  UsedColor

  UnusedColor                Version

## Methods

  DoProgress                Reset

## Reference Section

### DoProgress

method

```
procedure DoProgress(Progress : Byte);
```

- ↳ Adjusts the progress indicator.

This method is used internally by the Abbrevia archive components to update the progress indicator. Progress may contain a value between 0 and 100.

### Orientation

property

```
property Orientation : TAbMeterOrientation  
TAbMeterOrientation = (moHorizontal, moVertical);
```

Default: moHorizontal

- ↳ Determines whether the meter is horizontal or vertical.

If Orientation is moHorizontal, the meter is drawn so that the progress bar moves from left to right as progress increases. If Orientation is moVertical, the meter is drawn so that the progress bar moves from bottom to top as progress increases.

### Reset

method

```
procedure Reset;
```

- ↳ Resets the progress indicator to zero.

Reset is used internally by the Abbrevia archive components to reset the progress bar.

**UnusedColor****property**

```
property UnusedColor : TColor
```

Default: clBtnFace

- ↳ Provides the color used to draw the unused portion of the progress bar.

**UsedColor****property**

```
property UsedColor : TColor
```

Default: clNavy

- ↳ Provides the color used to draw the used portion of the progress bar.

**Version****property**

```
property Version : string
```

- ↳ The current version number of Abbrevia.

Version is provided so you can identify your Abbrevia version if you need technical support. If you double-click on the Version property in the IDE's object inspector, the Abbrevia About box is displayed.

---

# TAbMakeSelfExe Component

The TAbMakeSelfExe component provides a mechanism that simplifies the process of building a self-extracting zip archive.

A self-extracting archive is an executable file that, when executed, will extract files contained within itself. You can use a self-extracting archive when you need to distribute an archive to someone who doesn't have the tools to extract files from a PKZIP-compatible archive. Abbrevia's components operate on self-extracting archives just as if they were standard PKZIP archives.

The TAbMakeSelfExe component adds an existing zip archive file to an existing executable stub file and modifies references within the archive, resulting in a self-extracting zip archive file. The process is to first create a self-extracting stub and then attach an archive to it. Since you can build your own self-extracting stubs with Abbrevia, you control what happens when the self-extracting archive is run. A simple stub project called SelfStub is included with the Abbrevia source files.

## Hierarchy

### TComponent

❶ TAbBaseComponent (AbBase) .....	54
TAbMakeSelfExe (AbSelfEx)	

## Properties

SelfExe	❶ Version
StubExe	ZipFile

11

## Methods

Execute
---------

## Events

OnGetStubExe	OnGetZipFile
--------------	--------------

## Reference Section

### Execute method

---

```
function Execute : Boolean;
```

- ↳ Builds the self-executable file.

Use Execute to combine the executable stub specified in the StubExe property with the zip file specified in the ZipFile property. If no file is specified in StubExe, the OnGetStubExe event is fired to obtain the file name. If no file is specified in ZipFile, the OnGetZipFile event is fired to obtain the file name. The self-extracting executable will be saved as the file specified by the SelfExe property.

See also: OnGetStubExe, OnGetZipFile, SelfExe, StubExe, ZipFile

### OnGetStubExe event

---

```
property OnGetStubExe : TAbGetFileEvent  
TAbGetFileEvent = procedure(Sender : TObject;  
    var aFileName : string; var Abort : Boolean) of object;
```

- ↳ Defines an event handler that is called to obtain the file name of the executable stub.

OnGetStubExe is called if the StubExe property does not specify the executable stub file name.

### OnGetZipFile event

---

```
property OnGetZipFile : TAbGetFileEvent
```

- ↳ Defines an event handler that is called to obtain the file name of the zip archive.

OnGetZipFile is called if the ZipFile property does not specify the zip archive.

### SelfExe property

---

```
property SelfExe : string
```

- ↳ Specifies the file name of the self-extracting executable.

**SelfExe** specifies the file name of the self-extracting executable file. If **SelfExe** is not set when the **Execute** method is called, the following occurs:

### **Windows**

The file name specified by **ZipFile** will be used with the extension changed to ".EXE."

### **Linux**

The file name specified by **ZipFile** will be used with the extension removed and file permissions for the owner set to include execute.

#### **StubExe**

**run-time property**

---

```
property StubExe : string
```

↳ Specifies the file name of the executable stub.

Use **StubExe** to specify the executable stub to use. If **StubExe** is not set when the **Execute** method is called, then the **OnGetStubExe** event will be fired to obtain the file name of the executable stub.

#### **ZipFile**

**property**

---

```
property ZipFile : string
```

↳ Specifies the file name of the zip archive.

Use **ZipFile** to specify the zip archive to make self-extracting. If **ZipFile** is not set when the **Execute** method is called, then the **OnGetZipFile** event will be fired to obtain the file name of the zip archive.

---

## TAbDirDlg Class

The TAbDirDlg class provides a dialog for selecting a directory.

### Windows

The TAbDirDlg class allows you to use the Windows shell browser dialog to browse the shell for a folder.

### Linux

The TAbDirDlg class creates a custom form that displays a “TreeView” style hierarchy of folders to browse the file system for a folder.

## Hierarchy

TComponent

TAbDirDlg (AbDlgDir)

## Properties

AdditionalText

Caption

SelectedFolder

## Methods

Execute

## Reference Section

### AdditionalText

### property

```
property AdditionalText : string
```

Specifies the text that appears on the dialog box.

AdditionalText appears just below the title bar.

#### Windows

Windows only allocates space for two lines of text for this field so AdditionalText should be limited to 80 - 90 characters.

#### Linux

On Linux the amount of text is only limited by the capacity of the Caption property of the TLabel control but a similar length is suggested to keep the text neatly on the form.

If AdditionalText is set to an empty string, this area of the dialog box is left blank.

See also: Caption

### Caption

### property

```
property Caption : string
```

Specifies the text that appears in the title bar of the dialog box.

Use Caption to specify a custom title for the dialog box.

If Caption is an empty string, Windows automatically supplies the text "Browse for Folder."

The same caption is supplied on Linux for compatibility's sake.

See also: AdditionalText

### Execute

### method

```
function Execute : Boolean;
```

Displays the browser dialog box.

Call Execute to display the browser dialog box. Execute returns True if the dialog box was closed with the OK button, and False if the dialog box was closed with the Cancel button or the close box. Read the SelectedFolder property to determine the path of the selected folder.

See also: SelectedFolder

**SelectedFolder****property**

```
property SelectedFolder : string
```

- ↳ Contains the path of the selected folder.

Read SelectedFolder to get the path to the selected folder. To force the dialog box to select a particular folder on start up, set SelectedFolder to the desired start folder before calling Execute. If the directory in SelectedFolder does not exist, the dialog box will start with MyComputer on Windows, and the user's home directory on Linux.

---

## Zip File Association Routines (Windows Only)

A file association is used to run a specific application when a file with a specific extension is double-clicked within Windows Explorer. The following routines simplify the process of associating zip files with your application. These routines are contained in the AbZipExt unit.

For Windows, the association is registered in the Registry under the primary key, HKEY\_CLASSES\_ROOT. For example, the following Registry keys identify the zip file association with the application, C:\MYFOLDER\MYZIP.EXE:

```
HKEY_CLASSES_ROOT\ZIP
    Name: (Default)      Data: "MyZip.Document"

HKEY_CLASSES_ROOT\MyZip.Document
    Name: (Default)      Data: "MyZip 4.0 document"

HKEY_CLASSES_ROOT\MyZip.Document\Shell\Open\Command
    Name: (Default)      Data: "c:\myfolder\myzip.exe"
```

The first key associates the ".ZIP" file extension with the identifier "MyZip.Document". In essence, this identifies files with the ".ZIP" extension as "MyZip" documents. The second key provides a brief description the file type. This is what is displayed in the "Type" column of Explorer. The third key specifies the path to the application, MYZIP.EXE, that will be used to open a MyZip document.

The example project, Zipreg, shows how these routines can be used to associate zip files with your application.

### Procedures/Functions

AbExistingZipAssociation

AbGetZipAssociation

AbRegisterZipExtension

## Reference Section

---

### AbExistingZipAssociation function

---

```
function AbExistingZipAssociation : Boolean;
```

↳ Indicates whether the “.ZIP” extension has been registered.

Use AbExistingZipAssociation to determine if the “.ZIP” file extension has been registered with Windows. The function returns True if an application is currently associated with .ZIP, and returns False if there currently is no associated application registered.

---

### AbGetZipAssociation function

---

```
function AbGetZipAssociation(  
    var App, ID, FileType : string) : Boolean
```

↳ Returns the current zip file association.

Use AbGetZipAssociation to obtain the current zip file association. If a zip file association has been registered with Windows the function returns True, and App will contain the path to application, ID will contain the association identifier, and FileType will contain the file type description. If no zip file association currently exists, the function returns False.

---

### AbRegisterZipExtension function

---

```
function AbRegisterZipExtension(  
    const App, ID, FileType : string; Replace : Boolean) : Boolean
```

↳ Associates the specified application with zip files.

Use AbRegisterZipExtension to register a zip file association. App specifies the application path. ID specifies the association identifier. FileType specifies the file type description.

---

## Low-level Deflate Compression

For its Zip, ZLib, and GZip support, Abbrevia uses its own implementations of the standard Deflate or Deflate64 algorithms to compress data. To enable wider applicability and use of these algorithms, the low-level compression and decompression routines are interfaced and documented.

The Deflate routine compresses a source stream and writes it to another stream; the second decompresses a source stream into another. Using these routines you can compress and decompress arbitrary streams, without having to use the standard Zip, ZLib or GZip formats.

The two routines take an instance of a special class, TAbDeflateHelper, to define the options for both compression and decompression.

### Procedures/Functions

Deflate

Inflate

## Reference Section

Deflate	function
---------	----------

```
function Deflate(aSource : TStream; aDest : TStream;
    aHelper : TAbDeflateHelper) : Longint;
```

- ↳ Compresses the source stream to the destination stream.

The Deflate function will compress the data in the source stream, aSource, and write the compressed data to the output stream, aDest.

The options for the compression process are given by the aHelper parameter, an instance of the TAbDeflateHelper class. If this is nil, Deflate will create one with default values and use that.

The function result is the checksum of the input, decompressed stream. The aHelper instance defines whether this checksum is a CRC32 or Adler32 value (the default being CRC32).

See also: Inflate

Inflate	function
---------	----------

```
function Inflate(aSource : TStream; aDest : TStream;
    aHelper : TAbDeflateHelper) : Longint;
```

- ↳ Decompresses the source stream to the destination stream.

The Inflate function will decompress the data in the source stream, aSource, and write the decompressed data to the output stream, aDest.

The options for the decompression process are given by the aHelper parameter, an instance of the TAbDeflateHelper class. If this is nil, Inflate will create one with default values and use that.

The function result is the checksum of the output, decompressed stream. The aHelper instance defines whether this checksum is a CRC32 or Adler32 value (the default being CRC32).

See also: Deflate

---

## TheTAbDeflateHelper Class

To aid in the compression or decompression process, the Deflate and Inflate routines take an instance of a special helper class as a parameter. This helper class merely encapsulates configuration parameters for the compression or decompression; it does not in and of itself do any work. If no instance of this class is provided, a default instance will be created and used.

The helper object enables the compression or decompression process to be tuned and monitored. Examples of the properties exposed in this object are the ability to specify compression parameters (akin to PKZIP's space versus time option), to turn compression options on or off for testing or experimentation purposes, to define the password for encryption, to log the process, and to provide a callback for progress monitoring.

Some of the properties exposed in this class require some background knowledge of the Deflate algorithm, in order that they can be used effectively. Please see the "Additional Reading" section on page 24 for more details.

### Hierarchy

TObject

TAbDeflateHelper (AbUtils)

### Properties

AmpleLength	MaxLazyLength	Passphrase
ChainLength	OnProgressStep	PKZipOption
LogFile	Options	WindowSize

### Methods

Create

## Reference Section

---

<b>AmpleLength</b>	<b>property</b>
--------------------	-----------------

---

`property AmpleLength : longint`

Default: 8

↳ Defines the length of the matching string that will stop a search.

To enable the LZ77 compression phase of the Deflate algorithm to work efficiently, previously seen strings are stored in a hash table that uses chaining to resolve collisions. During a search along a chain for a match for the current string, this property defines the length of a “good enough” match to force the search to stop prematurely. In other words, if the search along a hash table chain uncovers a match of at least AmpleLength bytes, the search is stopped and the position returned.

The smaller the “good enough” length, the faster the LZ77 compression phase, but the larger the compressed data. This is a time versus space trade off.

If the best LZ77 compression is to be done, set AmpleLength to -1. This will find the longest match at each step in the LZ77 phase, but will take the longest time.

See also: ChainLength, MaxLazyLength, Options, PKZipOption

---

<b>ChainLength</b>	<b>property</b>
--------------------	-----------------

---

`property ChainLength : longint`

Default: 32

↳ Defines the length of the hash chain to search during compression.

To enable the LZ77 compression phase of the Deflate algorithm to work efficiently, previously seen strings are stored in a hash table that uses chaining to resolve collisions. During a search along a chain for a match for the current string, this property defines how many items in each chain are checked to find the longest match.

The smaller the chain length, the faster the LZ77 compression phase, but the larger the compressed data. This is a time versus space trade-off.

If you want to check the entire chain at each step, set ChainLength to -1. This will produce the optimal compression for the LZ77 phase, but will take the longest time.

See also: AmpleLength, MaxLazyLength, Options, PKZipOption

```
constructor Create;
```

- ↳ Creates a default Deflate helper object.

The constructor creates an instance of the TAbDeflateHelper class that has default values for its properties. The default values are shown in the following table.

Property	Default setting
AmpleLength	8 bytes
ChainLength	32 items
LogFile	No log file is generated
MaxLazyLength	16 bytes
OnProgressStep	No event handler installed
Options	Enable all compression options use lazy matching do not use Deflate64 calculate CRC32 checksum
Passphrase	Empty string (that is, no encryption/decryption is enabled)
WindowSize	32768 bytes

These defaults correspond to the PKZIP –en parameter; that is, normal Deflate compression.

## LogFile

## property

```
property LogFile : string
```

Default: Empty string

- ↳ Defines the name of the log file, if any.

If the LogFile property is set to a file name when the object is used by the Deflate or Inflate routines, information will be written to the file as the compression or decompression process runs. This information is used for debugging purposes only. Although its format is undocumented (and may change with updates to the product), it essentially shows, on compression, the parameters passed to the compression process, the LZ77 string matching information, the Huffman tree structures, and so on. On decompression, this log shows the parameters used, the structure of the compressed stream, and how the stream is decompressed.

Note that the LogFile property, if set, is assumed to be a correct file name, referring to a writeable disk, otherwise a standard exception will be raised when it is opened or used. The log file is always recreated afresh, and is never appended to.

If the LogFile property is the empty string, no log is produced.

---

<b>MaxLazyLength</b>	<b>property</b>
----------------------	-----------------

---

`property MaxLazyLength : longint`

Default: 16

↳ Defines the maximum length of the previous matching string.

To enable the LZ77 compression phase of the Deflate algorithm to work more efficiently, the lazy matching algorithm is used. In essence, lazy matching means calculating the longest match at the previous string position and at the current one and selecting the longest for output. If the previous matched string is equal to or longer than MaxLazyLength, it is used automatically without trying to match at the current position.

The smaller the MaxLazyLength value, the faster the LZ77 compression phase, but the larger the compressed data. This is a time versus space trade-off.

If the best LZ77 compression is to be done, set MaxLazyLength to -1. This will find the best lazy match at each step in the LZ77 phase, but will take the longest time.

If the Options property defines that no lazy matching is to take place, this property is ignored.

See also: AmpleLength, ChainLength, Options, PKZipOption

11

---

<b>OnProgressStep</b>	<b>event</b>
-----------------------	--------------

---

`property OnProgressStep : TABProgressStep`

`TABProgressStep = procedure (aPercentDone : integer) of object;`

Default: nil

↳ Defines the event handler that fires regularly during the process.

If the OnProgressStep property is not set, no monitoring of the progress of the compression or decompression can be done. Otherwise OnProgressStep defines a method that will be called regularly during the process of translating one stream to the other. The event may not be fired at every percentage point though: the frequency will depend on the size of the input stream.

On decompression, the `OnProgressStep` event is fired regularly as data is read from the input stream. Essentially decompression is a linear process with regards to time (the time taken will be roughly proportional to the size of data to decompress), and so the intervals between events will be roughly constant.

On compression, however, the process is more complex. Essentially the Deflate algorithm is a three-step process, repeated as many times as required: an LZ77 compression step, an analysis phase, followed by an optional Huffman compression step. The input stream is read during the LZ77 step and the output stream is written to during the Huffman step. However, between these two steps is an analysis phase where Abbrevia decides the best method of storing the compressed data (it is here, for example, that Abbrevia will decide that the data is uncompressible). The analysis phase uses statistical methods to decide which of several options the data should be compressed with, if at all. Although the other two steps are roughly linear in time, the analysis phase may vary in execution speed. Taking all these points together, it should be noted that although the `OnProgressStep` event will be fired regularly (for instance, every percentage point or every five percentage points), the lengths of the intervals between the events will vary. This effect should only be noticeable with larger input streams.

## Options

property

`property Options : longint`

Default: \$0000000F

Defines the options used to compress data.

The Options property is a bit-mapped long integer defining the various options for compression using Deflate.

Bit	Description	Default
0	Enable use of stored method	On
1	Enable use of static Huffman trees	On
2	Enable use of dynamic Huffman trees	On
3	Use lazy matching	On
4	Use Deflate64 algorithm	Off
5	Use Adler32 checksum	Off

Bits 0, 1, 2, and 3 should always be set, unless you are experimenting with how Deflate compresses without these parameters. These bits are ignored on decompression.

Bit 4 should be clear to create Deflated streams that will work with versions of PKZIP prior to version 4, ZLib, or GZip. On decompression, this bit must be set if the input stream was compressed with Deflate64 (Abbrevia cannot tell from the stream itself whether Deflate64 was used). Setting bit 4 in this situation enables Abbrevia to size the sliding window properly.

Bit 5 defines whether to calculate the CRC32 checksum (bit is clear) or the Adler32 checksum (bit is set) for the decompressed stream. zlib streams use Adler32 checksums, all other Zip variants use CRC32.

See also: AmpleLength, ChainLength, PKZipOption

---

<b>Passphrase</b>	<b>property</b>
-------------------	-----------------

`property Passphrase : string`

Default: empty string

↳ Defines the passphrase used to encrypt or decrypt the compressed data using PKZIP encryption.

The Deflate implementation in Abbrevia enables the compressed stream to be encrypted for extra security. The encryption algorithm used is the standard PKZIP method.

For the standard Zip encryption, a passphrase must be supplied (a collection of words, a phrase, or a sentence), or, less securely, a password. If the Passphrase property is set, the output stream will be encrypted on compression, and the input stream will be decrypted on decompression, both using the standard PKZIP encryption algorithm.

property ZipOption : char

Default: 'n'

↳ Defines the PKZIP deflation option for compression.

The PKZIP program enables the Deflate algorithm to be tuned by use of the -e parameter. This parameter takes an extra letter to indicate whether the compression should be tuned for speed or size. For convenience, Abbrevia's Deflate implementation can be tuned using the same letters. The various options are given in the following table:

x	Maximum compression
n	Normal compression
f	Fast compression
s	Super fast compression
0	No compression

Writing to this property will automatically set the AmpleLength, the ChainLength, the MaxLazyLength, and the Options properties accordingly. The value is case-insensitive.

See also: AmpleLength, ChainLength, MaxLazyLength, Options

property WindowSize : longint

Default: 32768

↳ Defines the sliding window size for the LZ77 compression phase.

For the standard Deflate algorithm, the WindowSize property should be 32,768 (32 \* 1024) or less, with the default of 32,768 being the standard size for Deflate. Smaller window sizes may speed up the LZ77 phase of the Deflate algorithm, at the cost of a slightly larger compressed size.

To compress using the Deflate64 algorithm, the WindowSize property can optionally be set to 65,536 (64 \* 1024). The Options property must first be set to indicate that Deflate64 is to be used before the WindowSize property can be set to 65,536.

When writing to this property, Abbrevia will force the window size to be a multiple of 1024 bytes within the correct range for the Deflate algorithm being used. No exception or error will be raised if this happens.

On decompression, the sliding window size is always set to 32,768 bytes for standard Deflate, or 65,536 for Deflate64. This property is therefore ignored during decompression.

---

## Simple Stream Compression Routines

Abbrevia includes two routines (`DeflateStream` defined in the `AbZipPrc` unit, and `InflateStream` defined in the `AbUnzPrc` unit) that allow low-level compression and decompression of data using streams. Because no directory information is stored, PKZIP and the other Abbrevia components cannot use the data created by these routines. Therefore, these routines should always be used as a pair.

The stream compression routines are ideal for compressing application-specific data. The stream data can be compressed directly, without involving any files.

**Note:** The `DeflateStream` and `InflateStream` are kept for backward compatibility with previous versions of Abbrevia; internally they just call the `Deflate` and `Inflate` routines described in the previous section. New code should use those routines instead.

The `Strmpad` example project in the Abbrevia examples directory illustrates how these routines can be used.

### Procedures/Functions

`DeflateStream`

`InflateStream`

## Reference Section

---

<b>DeflateStream</b>	<b>function</b>
----------------------	-----------------

---

```
procedure DeflateStream(  
  UncompressedStream, CompressedStream : TStream);
```

↳ Compresses the contents of a stream.

DeflateStream uses the PKZIP 2.0 deflate algorithm to compress the contents of UncompressedStream and stores the result in CompressedStream. DeflateStream does not store any header or footer information for the file—it simply stores the compressed data. No encryption is done and no events are generated by this procedure. Generally, CompressedStream should be empty prior to calling DeflateStream.

---

<b>InflateStream</b>	<b>function</b>
----------------------	-----------------

---

```
procedure InflateStream(  
  CompressedStream, UncompressedStream : TStream);
```

↳ Decompresses the contents of a stream.

InflateStream uses the PKZIP 2.0 inflate algorithm to decompress the contents of CompressedStream and stores the result in UncompressedStream. InflateStream does not require any header or footer information for the file, it simply restores the compressed data. No encryption is expected and no events are generated by this procedure. Generally, UncompressedStream should be empty prior to calling InflateStream.



---

## Chapter 12: Abbrevia 3 COM Automation Object (Windows Only)

Abbrevia's data compression capabilities are provided in two component formats. The format you decide to use depends on the kind of your development tool.

For Borland Delphi and C++Builder developers, Abbrevia's native VCL controls are the best way to gain access to Abbrevia's capabilities. The native VCL controls in Abbrevia work just like the ones that come with Delphi and C++Builder. Because VCL controls are written to work specifically with these compilers, they install directly into the compiler's development environment and work like the controls you already use. The functionality embodied by the VCL controls can be compiled directly into your application so there is nothing extra to distribute with the EXE file of your program.

For Internet and intranet development, however, native VCL controls are not a good choice. If you're building Web-based applications you should instead investigate the COM Automation Object that's also included with Abbrevia 3.

Abbrevia's Automation Object gives you full access to Abbrevia's extensive PKZIP-compatible compression engine. For Web developers this means that now Web applications can be created that can compress data "on the fly" before it gets sent to the browser.

## What's an Automation Object?

An *automation object* is a particular kind of COM object that can be controlled by another application, called an automation client. You may have already experienced automation objects such as Microsoft Excel, which allows you to access through code the spreadsheet functionality that is normally accessed through the program's user interface.

The definition for automation objects might naturally lead you to ask then what COM is. What follows is a brief overview, though it's hardly exhaustive. You can learn a lot more about COM on the Microsoft Web site, <http://www.microsoft.com>. There are also a number of books about COM. For a particularly good treatment of COM development with Delphi, *Delphi 4 Unleashed* by Charlie Calvert (SAMS Publishing) is recommended.

COM stands for Component Object Model, a software architecture developed by Microsoft that allows components made by different software vendors to be combined into a variety of applications. COM defines a standard for component interoperability, and Abbrevia's Automation Object adheres to this standard. Abbrevia's Automation Object is a server that can provide Abbrevia-compatible data compression formats (PKZIP, ZLib, GZip, and TAR) to client programs that properly create an instance of it.

Because COM isn't dependent on any particular programming language, COM Objects can be built and used in a variety of application development tools. Abbrevia's Automation Object, for example, was written using Borland Delphi, but you can invoke its capabilities using VBScript if you install the Abbrevia Automation Object on your Microsoft IIS Web server.

**Note:** TurboPower initially embarked on the creation of the Abbrevia Automation Object to meet one of its own needs for the TurboPower Web site. Though you can also use the Abbrevia Automation Object in other development tools capable of hosting Automation Objects, such as Microsoft Visual Basic, TurboPower's most extensive testing has been with Web development. See the TurboPower Web site for additional information about using the Abbrevia Automation Object with other development tools.

Unlike native VCL controls, COM Objects are packaged as EXE files or in dynamic link libraries (DLLs) that must be shipped with your application. The Abbrevia Automation Object is a DLL. If you choose to implement Abbrevia's functionality by using its Automation Object, you will need to ship and properly install the ABBREVIA.DLL file too.

---

## The Abbrevia Type Library

In addition to the Automation Object itself, Abbrevia also ships with a Type Library that describes the object's interface. Type Libraries are necessary so that client programs can get the names of the objects, methods, and properties in the COM Object. Some development tools can read the type library to help simplify your access to the COM Object. You do not have to ship the Abbrevia Type Library with your application.

---

## System Requirements

To use the Abbrevia COM Object, you must have the following hardware and software:

- A computer capable of running Microsoft Windows 95, Windows 98, Windows 2000, or Windows NT 4.0.
- A hosting environment that supports the creation and use of COM objects such as Borland Delphi 2.0 or higher, C++Builder, Microsoft Active Server Pages (ASP) or the Microsoft Windows Scripting Host.
- A hard disk with at least 10 MB of free space.

---

## Installation and Object Registration

Normally, the Abbrevia automation object is installed and registered when you install Abbrevia.

If you did not install the automation object previously, it can be installed directly from the CD- ROM. It is recommended that you exit all other applications before installing.

Insert the TurboPower Product Suite CD and follow the instructions presented by the setup program.

If you are installing from diskettes, run SETUP.EXE to start the installation process. Since SETUP is a Windows application, you must start Windows first. From the taskbar select Start and Run, and type X:\SETUP. Replace X:\ with the appropriate drive letter or the name of the directory if you copied the distribution files to your hard disk.

Setup installs the Abbrevia automation object into the C:\ABBREVIA\COM directory by default. You can specify a different directory if desired.

Setup creates a new program group named Abbrevia if one does not exist (you can specify a different name if desired). The program icons relevant to Abbrevia COM are added to it.

### On-line help

Although this manual provides a complete discussion of the Abbrevia COM object, keep in mind that there is an alternative source of information available. On-line help is available by selecting the Abbrevia COM help icon from the Abbrevia program group. You can also read more about using the COM object on the TurboPower Web site at <http://www.turbopower.com>.

### Readme.hlp

A help file (readme.hlp) is located in the directory in which you installed Abbrevia. This file describes changes to the documentation and new features added after the manual was printed. Please read this before using Abbrevia COM.

---

## Registering the Abbrevia COM Object

Before you can access the Abbrevia automation object in a program you must register it on your computer. Normally, the Abbrevia setup program will handle registration of the automation object for you. However, if you need to re-register the object, or register it on another system, you will need to use the REGSVR32.EXE included with Windows. To register the object, type:

[DRIVE:\PATH\]REGSVR32.EXE [DRIVE:\PATH\]ABBREVIA.DLL

from a DOS prompt, replacing [DRIVE:\PATH\] with the location of REGSVR32.EXE and ABBREVIA.DLL respectively.

To unregister the object type:

[DRIVE:\PATH\]REGSVR32.EXE /U [DRIVE:\PATH\]ABBREVIA.DLL

If you wish to register the object for use in Microsoft Transaction Server, please consult the documentation for Microsoft Transaction Server on package/component installation. A prebuilt package for use with the Microsoft Transaction server is not included with Abbrevia, but can easily be created using the Transaction Server Administration tools.

---

# Using the Abbrevia Automation Object with Microsoft Internet Information Server

After you have registered the Abbrevia automation object on your Web server you can get to work writing Active Server Pages (ASP) that perform data compression chores. ASP files contain HTML code and, optionally, scripting code. The scripting code can be written in either VBScript or JavaScript.

When a browser requests a page from your Web site, the Web server retrieves the requested page, processes any scripts first, then sends the resulting HTML to the browser. More can be learned about ASP files on the Microsoft Web site.

In the scripting code of an ASP file you can declare an instance of the Abbrevia automation object, instantiate it, use it to compress or decompress files, and release it.

In the following example, the names of the files in a zip archive on a Web page are displayed:

```
<%
  Dim Abd
  Dim Item

  Set Abd = CreateObject( "Abbrevia.IZipKit" )
  Abd.License( "xxxxx" )
  Abd.Filename = "C:\TEST.ZIP"

  For Each Item in Abd
    Response.Write Item.FileName & "<BR>"
  Next

  Set Abd = Nothing
%>
```

---

# Using the Abbrevia Automation Object with Microsoft Visual Basic 6

In addition to using the Abbrevia automation object in a Web application, it can also be used in development tools capable of invoking automation objects, such as Microsoft Visual Basic.

In the following simple example a list box with the names of the files in a Zip archive is populated:

```
Private Sub Command1_Click()
    Dim Abd as Abbrevia.ZipKit
    Dim Item as Abbrevia.ZipItem

    Set Abd = CreateObject("Abbrevia.ZipKit")
    Abd.License("xxxxx")
    Abd.Filename = "C:\TEST.ZIP"

    For Each Item in Abd
        List1.AddItem (Item.Filename)
    Next

    Set Abd = Nothing
End Sub
```

The first two lines declare variables that represent the two objects in ABBREVIA.DLL, the ZipKit and the ZipItem. Abd will be used to connect to a Zip archive file and Item to represent a single file inside the Zip archive file.

Once declared, use Set to actually create an instance of the ZipKit by calling CreateObject. Windows uses the text passed to CreateObject to connect the variable Abd with the Abbrevia object that you registered earlier. At this point, Abd points to an active instance of the ZipKit automation object.

The rest of the routine uses Abd properties to set the file name and retrieve a list of items from the ZipKit.

Finally, the automation object is released by setting its variable to Nothing.

---

## IZipItem Object

The IZipItem is the base class that represents an individual item in a zip archive. All of the properties of a zip item can be accessed from this interface, as well as the majority of GZip and TAR items. Properties that pertain only to GZip archive items and TAR archive items are covered in their respective sections. Only an IZipKit can create an IZipItem. However, once created, it can set or retrieve the item's properties.

### Properties

Action	DiskFileName	IsEncrypted
CompressedSize	DiskNumberStart	LastModFileDialogTime
CompressionMethod	DiskPath	Password
CompressionRatio	ExternalFileAttributes	StoredPath
CRC32	ExtraField	Tagged
CRC32St	FileComment	UnCompressedSize
DeflationOption	FileName	VersionMadeBy
DictionarySize	InternalFileAttributes	VersionNeededToExtract

## Reference Section

Action	read-only integer property
--------	----------------------------

`object.Action`

The `object` is always an `IZipItem` object.

↳ Describes the pending process for this archive item.

The Action property describes the processing that needs to occur on this item. During the save operation on the archive, each `IZipItem`'s Action property is checked, and any pending operations occur. The `IZipItem`'s Action property is then reset to aaNone.

The action can return one of the following constants from TArchiveAction:

Constant	Value	Description
aaFailed	0	The last action performed on this item failed.
aaNone	1	No pending action on this item.
aaAdd	2	This item will be added to the archive during the next save operation.
aaDelete	3	This item will be deleted to the archive during the next save operation.
aaFreshen	4	This item will be freshened in the archive during the next save operation.
aaMove	5	This item will be moved within the archive during the next save operation.
aaStreamAdd	6	This item will be added to the archive from a memory stream during the next save operation.

12

If the `IZipKit` that includes this `IZipItem` has its AutoSave property set to False, then processing of the archive actions is deferred until either:

- The archive that includes this item is saved using the archive's Save method.
- Multiple operations on a single item require the archive to be saved.

See also: `IZipKit.Save`

**CompressedSize**read-only integer property

---

*object.CompressedSize*The *object* is always an **IZipItem** object.

- ☞ The size in bytes of the file in its compressed state.

CompressedSize returns the actual size of the item within the archive. If the file is encrypted, the size of the encryption header is included.

See also: UnCompressedSize

**CompressionMethod**read/write integer property

---

*object.CompressionMethod*The *object* is always an **IZipItem** object.

- ☞ Describes the compression method used on this item in the archive.

The CompressionMethod field returns the method used to compress this item in the archive. The methods described in APPNOTE.TXT are included but not all these methods are supported by Abbrevia, nor by PKZIP.

The possible compression methods can be one of the following constants from the TZipCompressionMethod enumeration in the COM object:

Constant	Value	Description
cmStored	0	The file is stored (no compression).
cmShrunk	1	The file was compressed using the Shrink method.
cmReduced1	2	The file was compressed using the Reduce method with a compression factor of 1.
cmReduced2	3	The file was compressed using the Reduce method with a compression factor of 2.
cmReduced3	4	The file was compressed using the Reduce method with a compression factor of 3.
cmReduced4	5	The file was compressed using the Reduce method with a compression factor of 4.
cmImploded	6	The file was compressed using the the Implode method.
cmTokenized	7	Reserved for the Tokenize method.
cmDeflated	8	The file was compressed using the Deflate method.
cmEnhancedDeflated	9	Reserved for the enhanced Deflate method.
cmDCLImploded	10	PKWARE Data Compression Library Imploding.
cmBestMethod	11	Use the best method of compression based on the file type data.

See also: IZipKit.Save

## **CompressionRatio**

**read-only integer property**

---

*object.CompressionRatio*

The *object* is always an **IZipItem** object.

- ↳ Returns the compression ratio (in percent) for this item in the archive.

The calculated compression ratio is returned according to the following formula:

$\text{CompressionRatio} = 100 * (1 - (\text{CompSize} / \text{UncompressedSize}))$ .

Therefore, files that do not compress at all return a Compression Ratio of 0%, and files that compress to 1/3 their original size have a Compression Ratio of 67%.

**Note:** The compression ratio calculation first removes the 12 byte encryption header from the Compressed Size of password protected files. Thus, an encrypted, stored file returns a 0% CompressionRatio.

See also: CompressedSize, UnCompressedSize

## **CRC32**

**read-only integer property**

---

*object.CRC32*

The *object* is always an **IZipItem** object.

- ↳ A 32-bit CRC for the original file represented by this item in the archive.

The CRC of the original file is stored in the header information for a compressed file to provide a verification check that the item was properly darchived. Two files with the same CRC and the same size are almost certainly identical.

See also: IZipKit.Save

## **CRC32St**

**read-only string property**

---

*object.CRC32st*

The *object* is always an **IZipItem** object.

- ↳ A 32-bit CRC for the original file represented by this item in the archive.

The CRC of the original file. This property is calculated using the CRC32 property of the IZipItem, and converted to a string of up to eight hex characters (e.g., 54F19BE4).

See also: CRC32St, LastModFileDialogTime, UnCompressedSize

## DeflationOption

read/write integer property

*object.DeflationOption*

The *object* is always an **IZipItem** object.

- Allows the user to select parameters for PKZIP's Deflate algorithm.

The valid choices correspond to PKZIP 2.04g's -en, -ex, -ef, and -es options. The parameter is ignored for all other compression methods.

Listed below are the possible options from the TZipDeflationOption enumeration (defined in the COM object) that may be set or returned:

Constant	Value	Description
doInvalid	0	Deflate not used to compress the file.
doNormal	1	Balanced between speed and compression. This is the default setting.
doMaximum	2	Most compression, slowest speed.
doFast	3	Good compression but speed has priority.
doSuperFast	4	Least compression, fastest speed.

The choices correspond to the PKZIP 2.04g -ex, -en, -ef, and -es options.

See also: CompressionMethod

## DictionarySize

read-only integer property

*object.DictionarySize*

The *object* is always an **IZipItem** object.

12

- Describes the size of the dictionary used during the compression process of a file that has been compressed using PKZIP's Implode method.

Return values are defined in the TZipDictionarySize enumeration in the COM object:

Constant	Value	Description
dsInvalid	0	Imploding was not used to compress the file.
ds4K	1	An 8K sliding window was used during Imploding.
ds8K	2	An 4K sliding window was used during Imploding.

See also: CompressionMethod

## DiskFileName

read-only string property

`object.DiskFileName`

The `object` is always an **IZipItem** object.

☞ The file's complete file name as stored on the disk.

This may differ from the `FileName` property, which contains the file specification stored in the archive.

● **Caution:** `DiskFileName` may not be stored in the archive. PKZIP files do not store this information. When a PKZIP archive is first opened, the `DiskFileName` of each item in the archive will contain the same information found in the `FileName` property.

See also: `DiskPath`

## DiskNumberStart

read-only integer property

`object.DiskNumberStart`

The `object` is always an **IZipItem** object.

☞ The first image (or removable disk) number within a spanned set.

If the archive consists of a spanned set, either in archive images or on removable disks, `DiskNumberStart` contains the spanned set sequence number of the first image (or removable disk) in the set. This sequence number is zero-based so if the item begins on the second image, for example, `DiskNumberStart` will return a value of 1.

## DiskPath

read-only string property

`object.DiskPath`

The `object` is always an **IZipItem** object.

☞ The path information of a file that is to be stored in the archive.

`DiskPath` returns a string obtained by extracting the path information from `DiskFileName`.

See also: `DiskFileName`

## **ExternalFileAttributes**

**read-only integer property**

*object.ExternalFileAttributes*

The *object* is always an **IZipItem** object.

- ↳ An MS-DOS directory attribute byte.

Listed in the following table are the possible file attributes from TFileAttributes that can be set or returned:

<b>Constant</b>	<b>Value</b>	<b>Description</b>
faReadOnly	&H00000001	FILE_ATTRIBUTE_READONLY
faHidden	&H00000002	FILE_ATTRIBUTE_HIDDEN
faSysFile	&H00000004	FILE_ATTRIBUTE_SYSTEM
faVolumeID	&H00000008	
faDirectory	&H00000010	FILE_ATTRIBUTE_DIRECTORY
faArchive	&H00000020	FILE_ATTRIBUTE_ARCHIVE

These attributes can be combined by adding their constants or values. For example, a read-only and hidden file has a value of (faReadOnly + faHidden).

## **ExtraField**

**read/write string property**

*object.ExtraField*

The *object* is always an **IZipItem** object.

- ↳ Provides access to PKZIP's ExtraField.

The ExtraField portion of a LocalFileHeader allows for expansion to the zip format. Abbrevia simply maintains the ExtraField information.

## **FileComment**

**read/write string property**

*object.FileComment*

The *object* is always an **IZipItem** object.

- ↳ An optional comment which can be stored for each file in an archive.

Each file in a zip archive can have a different comment.

<b>FileName</b>	read/write string property
-----------------	----------------------------

*object.FileName*

The *object* is always an **IZipItem** object.

- ↳ The name under which the archived file is stored.

See also: [StoredPath](#)

<b>InternalFileAttributes</b>	read/write integer property
-------------------------------	-----------------------------

*object.InternalFileAttributes*

The *object* is always an **IZipItem** object.

- ↳ Defines the type of file this item represents.

**InternalFileAttributes** is set to 0 if the file is apparently a binary file, or 1 if it is an ASCII or text file. The **InternalFileAttributes** property is set by the compression process.

See also: [ExternalFileAttributes](#)

<b>IsEncrypted</b>	read-only boolean property
--------------------	----------------------------

*object.IsEncrypted*

The *object* is always an **IZipItem** object.

- ↳ Indicates whether the item is encrypted or not.

<b>LastModFileDialogTime</b>	read-only date property
------------------------------	-------------------------

*object.LastModFileDialogTime*

The *object* is always an **IZipItem** object.

- ↳ The last date and time the compressed file was modified.

<b>Password</b>	read/write string property
-----------------	----------------------------

*object.Password*

The *object* is always an **IZipItem** object.

- ↳ The password used to encrypt or decrypt the file.

When a file is added to, extracted from or freshened in the archive, **Password** is used to encrypt/decrypt it. If the password for the item is empty, the parent object's password is used or no encryption or decryption is done.

## **StoredPath**

**read-only string property**

*object*.**StoredPath**

The *object* is always an **IZipItem** object.

>Returns the path stored in the compressed file's header information.

As each file is stored in an archive, information relative to that file is also stored. This can optionally include path information. If path information is included, it can be accessed via StoredPath. If no path information is included, StoredPath returns an empty string.

## **Tagged**

**read/write boolean property**

*object*.**Tagged**

The *object* is always an **IZipItem** object.

>Selects or deselects items for archive operations, such as the delete, extract and freshen methods.

Tagged archive items are ones that have been selected.

See also: IZipKit.ClearTags, IZipKit.TagItems, IZipKit.UnTagItems,  
IZipKit.DeleteTaggedItems, IZipKit.ExtractTaggedItems,  
IZipKit.FreshenTaggedItems

## **UnCompressedSize**

**read-only integer property**

*object*.**UnCompressedSize**

The *object* is always an **IZipItem** object.

Returns the size in bytes of the file in its original, uncompressed state.

UnCompressedSize is the actual size of the file before it was compressed.

See also: CompressedSize

## **VersionMadeBy**

**read-only integer property**

---

*object.VersionMadeBy*

The *object* is always an **IZipItem** object.

- ↳ The version of PKZip that compressed or created this item.

To determine the major and minor version information, simply divide the returned value by 100. The whole number part is the major version, and the fractional part is the minor version.

See also: VersionNeededToExtract

## **VersionNeededToExtract**

**read-only integer property**

---

*object.VersionNeededToExtract*

The *object* is always an **IZipItem** object.

- ↳ The version number of PKZip required to extract the file from the archive.

To determine the major and minor version information, simply divide the returned value by 100. The whole number part is the major version, and the fractional part is the minor version.

The value returned by this property may differ from the VersionMadeBy. For example, PKZIP's Store compression method has been available since version 1.0. The VersionNeededToExtract for a stored file in a zip file is 1.0, even if it was written using PKZIP 2.0 or better.

See also: VersionMadeBy

---

## IGZipItem Object

The IGZipItem interface encapsulates the entire functionality of the IZipItem interface. In addition, properties specific to GZip archive items are introduced. Refer to the IZipItem interface for complete coverage of the interface properties pertaining to all archive items.

### Properties

FileSystem

## Reference Section

### FileSystem

read/write integer property

`Object.FileSystem`

The `object` is always an `IZipItem` object.

↳ Indicates the file system on which the compressed files were created.

Default: osFat (Windows); osUnix (Linux)

GZip provides a means to indicate which file system the GZipped file came from. This characteristic appears to be rarely used for anything but informational purposes in common GZip implementations, but is included here for completeness. The following table defines the meanings for the possible values for this property:

Value	Meaning
0	Fat
1	Amiga
2	VMS
3	Unix
4	VM_CMS
5	AtariTOS
6	HPFS
7	Macintosh
8	ZSystem
9	CP_M
10	TOPS20
11	NTFS
12	QDOS
13	AcornRISCOS
14	Unknown

Creating a GZip, FileSystem is set to Fat (0) automatically on Windows and to Unix (3) on Linux.

Setting it to other values will cause that to be stored into the resulting GZip rather than the default, but has no other effect on the compression process.

**Note:** The GZip specification refers to the header field as "OS" or "Operating System"

---

## ITARZipItem Object

The ITARZipItem interface encapsulates the entire functionality of the IZipItem interface. In addition, properties specific to TAR archive items are introduced. Refer to the IZipItem interface for complete coverage of the interface properties pertaining to all archive items.

### Properties

GroupID	UserID
GroupName	UserName

## Reference Section

GroupID	read/write integer property
---------	-----------------------------

`object.GroupID`

The `object` is always an `ITarZipItem` object.

↳ Indicates the Unix file system “Group ID” for the archived file.

Default: 0 (Windows); Current User’s Group ID (Linux)

The standard Unix file system (which is also used by Linux) includes the notion of a user group to which a file belongs. The GroupID property supplies this value for a Tar item.

When extracting files, this value is provided for informational purposes, but is otherwise ignored.

When compressing files, changing this value from the default causes the value to be stored with the file as the Group ID, but has no other effect on the archiving process.

See also: `GroupName`, `UserID`

GroupName	read/write string property
-----------	----------------------------

`object.GroupName`

The `object` is always an `ITarZipItem` object.

↳ Indicates the Unix file system “Group Name” for the archived file.

Default: “” (Windows); Current User’s Group Name (Linux)

The standard Unix file system (which is also used by Linux) includes the notion of a user group to which a file belongs, such groups can have symbolic names associated with them (e.g. “USERS”). The GroupName property supplies this value for a Tar item.

When extracting files, this value is provided for informational purposes, but is otherwise ignored.

When compressing files, changing this value from the default causes the value to be stored with the file as the Group Name, but has no other effect on the archiving process; in particular Abbrevia does no checking that the value set to GroupName corresponds to that associated with the GroupID property according to the Linux file system.

See also: `GroupID`, `UserName`

---

<b>UserID</b>	read/write integer property
---------------	-----------------------------

---

`object.UserID`

The *object* is always an **ITarZipItem** object.

↳ Indicates the Unix file system “User ID” for the archived file.

Default: 0 (Windows); Current User’s ID (Linux)

The standard Unix file system (which is also used by Linux) includes the notion of a particular user to which a file belongs. The UserID property supplies this value for a Tar item.

When extracting files, this value is provided for informational purposes but is otherwise ignored.

When compressing files, changing this value from the default causes the value to be stored with the file as the User ID, but has no other effect on the archiving process.

See also: GroupID, UserName

---

<b>UserName</b>	read/write string property
-----------------	----------------------------

---

`object.UserName`

The *object* is always an **ITarZipItem** object.

↳ Indicates the Unix file system “User Name” for the archived file.

Default: “ ” (Windows); Current User’s UserName (Linux)

The standard Unix file system (which is also used by Linux) includes the notion of a particular user to which a file belongs, users may be identified both by number and by a symbolic name (e.g. “NWIRTH” might be the username for a person named “Niklaus Wirth”). The UserName property supplies this value for a Tar item.

When extracting files, this value is provided for informational purposes, but is otherwise ignored.

When compressing files, changing this value from the default causes the value to be stored with the file as the User Name, but has no other effect on the archiving process; in particular Abbrevia does no checking that the value set to UserName corresponds to that associated with the UserID property according to the Linux file system. The programmer has the option and opportunity to perform such checks in the OnProcessItem event of a TAbArchive descendant.

See also: GroupName, UserID

---

# IZipKit Object

The IZipKit is an automation server object that gives you the ability to browse a zip archive, add files and extract files. IZipKit is the base object that is created to allow access to all properties, events, and methods of the automation object.

## Properties

AutoSave	ExtractOptions	Spanned
BaseDirectory	FileName	SpanningThreshold
CompressionMethodToUse	Item	Status
CompressionType	LogFile	StoreOptions
Count	Logging	TempDirectory
DeflationOption	Password	ZipFileComment
DOSMode	PasswordRetries	

## Methods

Add	ExtractAt	Replace
AddFromStream	ExtractTaggedItems	Save
ClearTags	ExtractToStream	TagItems
Delete	Find	TestTaggedItems
DeleteAt	Freshen	UnTagItems
DeleteTaggedItems	FreshenTaggedItems	
Extract	License	

## Events

OnArchiveItemProgress	OnConfirmSave	OnRequestImage
OnArchiveProgress	OnLoad	OnRequestLastDisk
OnChange	OnNeedPassword	OnRequestNthDisk
OnConfirmOverwrite	OnProcessItemFailure	OnSave
OnConfirmProcessItem	OnRequestBlankDisk	

## Reference Section

### Add method

`object.Add(FileMask[, ExclusionMask[, SearchAttr]])`

The `object` is always an **IZipKit**.

- ↳ Adds the files that match FileMask to the archive.

The add syntax has these parts:

Part	Type	Description
FileMask	String	Required. An expression identifying the file(s) to add to the archive. Wild card characters ('*', '?') are allowed in the FileMask.
ExclusionMask	String	Optional. An expression identifying files that are excluded from any files matched in FileMask. Wild card characters ('*', '?') are allowed in the ExclusionMask as well as path information.
SearchAttr	Integer	Optional. MSDOS directory attributes to include in file mask search. See <code>IZipItem.ExternalFileAttributes</code>

When searching for files to add to the archive, the method searches certain directories, depending on the `StoreOptions` and `BaseDirectory` properties. The following table shows, for a given FileMask and set of `StoreOptions`, which directories are searched for the file. If the file is found, it is stored in the archive with the name shown in the last column.

StoreOptions	FileMask	Directories Searched	Name in Archive
0	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
soRecurse+ soStripPath	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT

StoreOptions	FileMask	Directories Searched	Name in Archive
	X:\DOC\README.TXT	X:\DOC	README.TXT
soRecurse	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT or SUBDIR/README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
soStripPath	README.TXT	BASEDIRECTORY	README.TXT
StoreOptions	FileMask	Directories searched	Name in archive
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT
	X:\DOC\README.TXT	X:\DOC	README.TXT

## AddFromStream method

`object.AddFromStream(FileName, Stream)`

The `object` is always an **IZipKit**.

⚡ Create and add an item directly from a stream.

The AddFromStream syntax has these parts:

Part	Type	Description
FileName	String	Required. This parameter identifies the file name of the item.
Stream	VT_ARRAY,	
VT_UI1	Required. This parameter will be a variant array of unsigned one byte characters.	

AddFromStream will create a zip item by compression data directly from a stream of data. A file name must be provided via NewName for the new zip item that is created. Adding data directly to an archive from a stream causes the archive to be automatically saved since the stream object may be subsequently freed or modified.

## AutoSave

read/write boolean property

*object*.**AutoSave**

The *object* is always an **IZipKit**.

☞ Controls whether changes to the archive are performed immediately or not.

When AutoSave is True, each change to the archive forces the disk image of the archive to be updated immediately. When AutoSave is False, the default, changes to the disk image are done when one of the following occur:

- The archive is explicitly saved using the Save method.
- Multiple operations on a single item require the archive to be saved.
- FileName is changed, causing a new archive to be opened.

**Note:** when AutoSave is True, only a single archive file may be created. Multiple archive file spanning requires that AutoSave is False. Also, keep in mind that saving the archive each time a file is added can be very time consuming. For these reasons, it is recommended that AutoSave be used only with relatively small archives.

See also: Save, FileName

## BaseDirectory

read/write string property

*object*.**BaseDirectory**

The *object* is always an **IZipKit**.

☞ The default path for add, extract and freshen operations on the archive.

Base directory is not necessarily the same as the directory of the archive.

12

When you add a file to an archive, if the FileName is a relative file specification (e.g., "README.TXT", or "EXAMPLES\README.TXT"), BaseDirectory is used as the starting point to search for the file. When you extract a file from an archive, it is extracted to the current BaseDirectory or a subdirectory of the current BaseDirectory. For information on how the BaseDirectory is used in each case, see the Add, Extract or Freshen methods.

☞ **Caution:** When used within Active Server Pages (ASP), the base directory needs to be set. If the BaseDirectory is not set, then all add, extract and freshen operations will be using a base directory of \WINNT\SYSTEM32. A simple way to set this is by using the following:

```
object.BaseDirectory = Server.MapPath( " / " )
```

This will set the BaseDirectory to the root of the Web site, wherever it may be located.

## **ClearTags**

**method**

*object*.**ClearTags**

The *object* is always an **IZipKit**.

- ↳ Untags all items in the archive.

See TagItems on page 366 for information on how tagging works.

## **CompressionMethodToUse**

**read/write integer property**

*object*.**CompressionMethodToUse**

The *object* is always an **IZipKit**.

- ↳ Selects the compression method used when adding or freshening items in the archive.

The possible values for CompressionMethodToUse are:

<b>Option</b>	<b>Description</b>
smStored	The file is stored without any compression.
smDeflated	The file is compressed using the deflate method.
smBestMethod	The file is deflated, but if the resulting file is larger than the original, the file is simply stored.

## **CompressionType**

**property**

*object*.**CompressionType**

The *object* is always an **IZipKit**.

Default: ctZip

- ↳ Specifies the compression type of the archive.

When opening an archive CompressionType is set to the type of archive as determined by examining the archive's file extension.

When creating an archive, the compression type must be set to the type of archive desired to be created.

## Count

read-only integer property

*object*.Count

The *object* is always an **IZipKit**.

- ↳ Returns the number of items available from the Item property.
- ⌚ **Caution:** Count returns the number of items in the memory representation of the archive. If AutoSave is False, and items have been added to or deleted from the archive, the number of items in the memory representation can be different than the number of items in the disk archive.

See also: AutoSave

## DeflationOption

read/write integer property

*object*.DeflationOption

The *object* is always an **IZipKit**.

- ↳ Determines whether priority is given to compression or speed during compression.

DeflationOption allows you to select the deflation option when CompressionMethodToUse is smDeflated or smBestMethod. You can select varying trade-offs between compression and speed. The choices are:

Constant	Value	Description
doInvalid	0	Deflate not used to compress the file.
doNormal	1	Balanced between speed and compression. This is the default setting.
doMaximum	2	Most compression, slowest speed.
doFast	3	Good compression but speed has priority.
doSuperFast	4	Least compression, fastest speed.

The choices correspond to the PKZIP 2.04g -ex, -en, -ef, and -es options.

See also: CompressionMethodToUse

## Delete

method

`object.Delete(FileMask[, ExclusionMask])`

The *object* is always an **IZipKit**.

- ↳ Delete files except those specified by ExclusionMask from the archive.

The Delete syntax has these parts:

Part	Type	Description
FileMask	String	Required. An expression identifying the file(s) to delete from the archive. Wild card characters ('*', '?') are allowed in the FileMask.
ExclusionMask	String	Optional. An expression identifying files that are excluded from any files matched in FileMask. Wild card characters ('*', '?') are allowed in the ExclusionMask as well as path information.

## DeleteAt

method

`object.DeleteAt(index)`

The *object* is always an **IZipKit**.

- ↳ Deletes the item at the specified item index.

The DeleteAt syntax has these parts:

Part	Type	Description
Index	Integer	Required. The index of the item to be deleted form the archive.

DeleteAt can be used to delete an item from an archive when its index is known. Index is the zero-based item index of the item to be deleted.

## DeleteTaggedItems

method

`object.DeleteTaggedItems()`

The *object* is always an **IZipKit**.

- ↳ Finds all files that have their Tagged property set to True and marks them to be deleted.

The next time the archive is saved, all marked files and their corresponding information are deleted from the archive.

See the TagItems method on page 366 for information about how tagging works.

## DOSMode

read/write boolean property

*object*.DOSMode

The *object* is always an **IZipKit**.

↳ Forces all new files stored in an archive to have DOS-compatible file names.

Windows 95 and Windows NT introduced long file names, which were not available under older versions of Windows or DOS. The older operating systems cannot handle the long file names; they require a file name no longer than eight characters (8.3 format).

If your archive must be read by an older operating system, you can force short file names by setting DOSMode to True. For example, if DOSMode is True, a file named C:\PROGRAM FILES\README.TXT is stored in the archive as PROGRA~1/README.TXT.

## Extract

method

*object*.Extract(*FileMask*[, *ExclusionMask*])

The *object* is always an **IZipKit**.

↳ Extracts files except those specified by ExclusionMask.

The Extract syntax has these parts:

Part	Type	Description
FileMask	String	Required. An expression identifying the file(s) to extract from the archive. Wild card characters ('*', '?') are allowed in the FileMask.
ExclusionMask	String	Optional. An expression identifying files that are excluded from any files matched in FileMask. Wild card characters ('*', '?') are allowed in the ExclusionMask as well as path information.

The files in the archive are not modified by extract operations. The extraction happens immediately. Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is extracted. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” extracts all files in the archive that have a TXT extension. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be extracted.

The following table shows some example FileMasks and stored file names and indicates whether they would be considered a match:

FileMask	Stored Filename	Match?
*.txt	test.txt	Yes
*.txt	doc/test.txt	Yes
*.txt	bin/test.txt	Yes
doc/*.txt	test.txt	No
doc/*.txt	doc/test.txt	Yes
doc/*.txt	bin/test.txt	No
*/*.txt	test.txt	No
*/*.txt	doc/test.txt	Yes
*/*.txt	bin/test.txt	Yes

## ExtractAt method

`object.ExtractAt(Index[, NewName])`

The `object` is always an **IZipKit**.

↳ Extracts an item with known item index.

The ExtractAt syntax has these parts:

Part	Type	Description
Index	Integer	Required. The index of the item to be deleted from the archive.
NewName	String	Optional. An expression identifying the new file name to give the item extracted.

ExtractAt can be used to extract an item from a zip archive when its item index is known. Index is the zero-based item index. NewName is the file name to be given to the item, once extracted, if different than the stored file name. If NewName is left blank then the item's stored file name will be used.

*object*.**ExtractOptions**

The *object* is always an **IZipKit**.

↳ Determines the options for archive extract operations.

If ExtractOptions contains eoCreateDirs, directories (including multiple levels of subdirectories) are created as needed to extract files.

If ExtractOptions contains eoRestorePath, path information stored in the archive is retained when files are extracted. The extracted file is placed in the appropriate subdirectory relative to the current BaseDirectory.

The following table shows the possible values that can be used:

Constant	Value	Description
eoCreateDirs	&H00000000	Creates directories as needed during extraction.
eoRestorePath	&H00000001	Restores path information during extraction.

The following table shows, for a given file in the archive and a set of ExtractOptions, to which directory the file is extracted:

ExtractOptions	Name in Archive	Directory for Extracted File
0	README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
	DOC/README.TXT	BASEDIRECTORY, if it exists. Otherwise, an exception is raised.
eoCreateDirs+eoRestorePath	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY\DOC, which is created if necessary.
eoCreateDirs	README.TXT	BASEDIRECTORY, which is created if necessary.
	DOC/README.TXT	BASEDIRECTORY, which is created if necessary.
eoRestorePath	README.TXT	BASEDIRECTORY, if it exists. Otherwise an exception is raised.
	DOC/README.TXT	BASEDIRECTORY\DOC, if it exists. Otherwise, an exception is raised.

## ExtractTaggedItems

method

*object.ExtractTaggedItems()*

The *object* is always an **IZipKit**.

- ☞ Extracts all files that have their Tagged property set to True.

The files in the archive are not modified by extract operations. The extraction happens immediately. The files are extracted into certain directories, depending on the ExtractOptions and BaseDirectory properties. See ExtractOptions for a description of how the directory is determined.

See the TagItems method on page 366 for information about how tagging works.

## ExtractToStream

method

[ *Stream* = ] *object.ExtractToStream(FileName)*

The *object* is always an **IZipKit**.

- ☞ Extracts the specified item directly to a stream.

The ExtractToStream syntax has these parts:

Part	Type	Description
FileName	String	Required. This parameter identifies the file name of the item to extract.
Stream	VT_ARRAY,	
VT_UI1	The return value will be a variant array of unsigned one byte characters	

ExtractToStream will extract an item directly to an array. Any data already stored in the array will be cleared.

## FileName

read/write string property

*object.FileName*

The *object* is always an **IZipKit**.

- ☞ The name of the archive.

If FileName is changed, the current archive is saved, if necessary, and closed. If FileName is not blank, a new archive is created and loaded.

## Find

method

```
[ Index = ] object.Find(FileName)
```

The *object* is always an **IZipKit**.

- ↳ Searches the archive for a file with an exactly matching file name and returns the index of that item.

The Find syntax has these parts:

Part	Type	Description
FileName	String	Required. This parameter identifies the file name of the item to find.
Index	Integer	The return value will be the index of the found item

Find searches through the list of archive files until it finds one with an exact match for the specified file name. It then returns the index of that item. Find returns -1 if no match is found. No two items in an archive can have the same file name.

## Freshen

method

```
object.Freshen(FileMask[ , ExclusionMask])
```

The *object* is always an **IZipKit**.

- ↳ Freshen files in the archive except those specified by ExclusionMask.

The Freshen syntax has these parts:

Part	Type	Description
FileMask	String	Required. An expression identifying the file(s) to freshen in the archive. Wild card characters ('*', '?') are allowed in the FileMask.
ExclusionMask	String	Optional. An expression identifying files that are excluded from any files matched in FileMask. Wild card characters ('*', '?') are allowed in the ExclusionMask as well as path information.

Each file in the archive is compared to FileMask. If an archive item's stored file name matches FileMask, the file is marked to be freshened. If no directory information is included in FileMask, directory information in the archive item's file name is ignored during the search. For example, the FileMask “\*.TXT” marks all files in the archive that have a TXT extension to be freshened. If directory information is included in FileMask, the directory information in the archive item's file name must match for the file to be freshened.

The next time the archive is saved, the marked files are freshened. If the file is found on disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file.

The directories that are searched to find the file depend on the StoreOptions and the BaseDirectory properties.

The following table shows, for a given file in the archive and a set of StoreOptions, what directories are searched to find the file to use to freshen the file in the archive:

<b>StoreOptions</b>	<b>FileMask</b>	<b>Directories Searched</b>	<b>Name in Archive</b>
0	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
soRecurse+ soStripPath	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	README.TXT
	X:\DOC\README.TXT	X:\DOC	README.TXT
StoreOptions	FileMask	Directories searched	Name in archive
soRecurse	README.TXT	BASEDIRECTORY and all subdirectories	README.TXT or SUBDIR/README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	DOC/README.TXT
	\DOC\README.TXT	DEFAULTDRIVE:\DOC	DOC/README.TXT
	X:\DOC\README.TXT	X:\DOC	DOC/README.TXT
soStripPath	README.TXT	BASEDIRECTORY	README.TXT
	DOC\README.TXT	BASEDIRECTORY\DOC	README.TEXT
	\DOC\README.TEXT	DEFAULTDRIVE:\DOC	README.TEXT
	X:\DOC\README.TEXT	X:\DOC	README.TEXT

```
object.FreshenTaggedItems()
```

The *object* is always an **IZipKit**.

- ↳ Finds all files that have their Tagged property set to True and marks them to be freshened. a

The next time the archive is saved, all marked files are freshened. If the file is found on the disk and it has been modified since it was last stored in the archive, the compressed data in the archive is replaced with the compressed representation of the current file. The directories that are searched to find the file depend on the StoreOptions and BaseDirectory properties. See the table in FreshenFiles for details of determining the directory to search.

See TagItems on page 366 for information about how tagging works.

---

Item	read-only property
------	--------------------

---

```
[ ZipItem = ] object.Item(index)
```

The *object* is always an **IZipKit**.

- ↳ Item contains an interface for each item in the archive.

The Item method syntax has these parts:

---

Part	Type	Description
Index	Integer	Required. This parameter identifies the index of the item to return.
ZipItem	IZipItem	The return value will be an interface to a ZipItem object.

---

Each item in an archive is described using a IZipItem interface. See the IZipItem Object on page 321 for more information on this object. Item is also the default collection for the ZipKit object. The following example accesses each item's FileName in the archive:

```
For Each Item in ZipKit
    MsgBox(Item.FileName)
Next
```

See also: Count

## License

## method

[ Licensed = ] object.**License**(Key)

The object is always an IZipKit.

- ↳ Verify the license key.

The License syntax has these parts:

Part	Type	Description
Key	String	The license key.
Licensed	Boolean	True if accepted, False otherwise.

The License method must be called to license the Abbrevia Automation Object for run-time use. This applies when using Visual Basic's CreateObject() call or the CoCreateInstance COM API call.

The License method must be the first method you call after instantiating the Abbrevia Automation Object. If you do not call this method, no other Abbrevia IZipKit methods or properties will work.

Please refer to DEPLOY.HLP for more information about licensing and deploying the Abbrevia Automation Object.

## LogFile

## read/write string property

object.**LogFile**

The object is always an IZipKit.

- ↳ Specifies the text file to use for Logging.

LogFile specifies the text file that will receive the log entries during archiving operations if Logging has been enabled. If the text file does not exist, it will be created.

See also: Logging

## Logging

read/write boolean property

*object*.Logging

The *object* is always an **IZipKit**.

- ↳ Controls whether archive operations are recorded or not.

When Logging is True, each operation (add, delete, extract, freshen) performed on an archive will create an entry in a text file. The text file is specified by the LogFile property. The following example shows the format of log entries:

```
D:\Test\Test.zip logging 04/27/1999 6:12:06 PM  
FDI.H deleted 04/27/1999 6:12:37 PM  
Ffdefine.inc added 04/27/1999 6:12:52 PM
```

## OnArchiveItemProgress

event

*object*\_OnArchiveItemProgress(*Item*, *Progress*, *Abort*)

The *object* is always an **IZipKit**.

- ↳ Defines an event handler that is called during long operations on a single archive item.

The OnArchiveItemProgress syntax has these parts:

Part	Type	Description
Item	IZipItem	The item that action is being taken on.
Progress	Byte	Percentage that indicates how far the operation has progressed through the archive item. Valid values for Progress are 0 to 100.
Abort	Boolean	Setting abort to True causes the current operation to be aborted.

12

OnArchiveItemProgress is called by Add, Extract, and Freshen operations on an archive item. You can use it to display the progress of the operation, or even to abort the operation. For example, you might want to display the name of the item being processed, the action being performed on the item, and an indication of the progress.

## OnArchiveProgress

event

*object\_OnArchiveProgress(Progress, Abort)*

The *object* is always an **IZipKit**.

- ☞ Defines an event handler that is called once per item when a process steps through the archive.

The OnArchiveProgress syntax has these parts:

Part	Type	Description
Progress	Byte	Percentage that indicates how far the operation has progressed through the archive. Valid values for Progress are 0 to 100.
Abort	Boolean	Setting abort to True causes the current operation to be aborted.

OnArchiveProgress is called during the Delete, DeleteAt, Freshen, Extract, ExtractAt, and Save archive operations. Changing the FileName also signals this event (a change causes a new archive to be loaded). This event can be used to display the progress of the operation, or even to abort the operation.

## OnChange

event

*object\_OnChange()*

The *object* is always an **IZipKit**.

- ☞ Defines an event handler that is called when the archive changes.

OnChange is called when the contents of the archive change (immediately after an add, delete, freshen, or move operation) or when a property of an item in the archive changes. The OnChange event can be changed to update any components that display the contents of the archive or information relating to the archive (e.g., the number of files in the archive).

## OnConfirmOverwrite

event

*object*\_OnConfirmOverwrite(*Name*, *Confirm*)

The *object* is always an **IZipKit**.

- ☞ Defines an event handler that is called if the output file for an extract operation already exists.

The OnConfirmOverwrite syntax has these parts:

Part	Type	Description
Name	String	A new name for the file being extracted.
Confirm	Boolean	Setting Confirm to False aborts the process.

This event handler is called prior to the start of the extraction process if the expected output file already exists. If a handler for this event is not provided, the file is overwritten without warning. Setting a new Name for the file, setting Confirm to True will continue the extraction of the file.

## OnConfirmProcessItem

event

*object*\_OnConfirmProcessItem(*Item*, *ProcessType*, *Confirm*)

The *object* is always an **IZipKit**.

- ☞ Defines an event handler that is called before a process is performed on each item in the archive.

The OnConfirmProcessItem syntax has these parts:

Part	Type	Description
Item	IZipItem	The item that action is being taken on.
ProcessType	TProcessType	The action being taken on the item. See the chart below for possible values.
Confirm	Boolean	Setting Confirm to False aborts the process.

ProcessType can be one of the following (TProcessType):

Constant	Value	Description
ptAdd	0	The item is being added to the archive.
ptDelete	1	The item is being deleted from the archive.
ptExtract	2	The item is being extracted from the archive.
ptFreshen	3	The item is being freshened in the archive.
ptMove	4	The item is being moved within the archive.
ptReplace	5	The item is being replaced within the archive.

This event handler is called once for each item in an Add, Delete, Extract, Freshen, or Move loop. If a handler is not provided for this event, all processing continues as if Confirm were set to True.

The OnConfirmProcessItem event handler can be used to perform any actions that are necessary when the archive changes. For example, you might want to display a confirmation dialog or create a log of who modified the archive.

#### **OnConfirmSave** event

*object*\_OnConfirmSave(*Confirm*)

The *object* is always an **IZipKit**.

↳ Defines an event handler that is called before the archive is saved.

The OnConfirmSave syntax has these parts:

Part	Type	Description
Confirm	Boolean	Setting Confirm to False aborts the saving of the archive and pending changes are lost.

This event handler is called immediately before an archive is updated. If a handler is not provided for this event, all processing continues as if Confirm were set to True.

The OnConfirmSave event can be used to allow the user to close an archive without saving changes. This is most useful when the AutoSave property is False. When the object's FileName property is changed, the Save method is automatically called, which fires the OnConfirmSave event.

## OnLoad

event

*object*\_OnLoad()

The *object* is always an **IZipKit**.

↳ Defines an event handler that is called immediately after an archive's contents are loaded.

Use OnLoad event to display the name of the current archive or to update a most-recently-used archive list.

## OnNeedPassword

event

*object*\_OnNeedPassword(*NewPassword*)

The *object* is always an **IZipKit**.

↳ Defines an event handler that is called to allow entry of a password when decrypting a file.

The OnNeedPassword syntax has these parts:

Part	Type	Description
NewPassword	String	The password used to decrypt the file.

The OnNeedPassword event handler is called when an encrypted file is being extracted and one of the following occurs:

- The Password property is empty.
- The Password is not valid for the encrypted file and the number of attempts is less than PasswordRetries.

## OnProcessItemFailure

event

```
object_OnProcessItemFailure(Item, ProcessType, ErrorClass,  
    ErrorCode, ErrorString)
```

The *object* is always an **IZipKit**.

↳ Defines an event handler that is called when an exception is raised during an Add, Extract, Refreshen, Move, or Replace process.

The OnProcessItemFailure syntax has these parts:

Part	Type	Description
Item	IZipItem	The item that processing failed on.
ProcessType	TProcessType	The action that was preformed on the item when the failure happened. See the chart below for possible values.
ErrorClass	TErrorClass	The class that the error belongs to.
ErrorCode	TErrorCode	The unique error code of the error that occurred.
ErrorString	String	The text string describing the error that occurred.

ProcessType can be one of the following (TProcessType):

Constant	Value	Description
ptAdd	0	The item is being added to the archive.
ptDelete	1	The item is being deleted from the archive.
ptExtract	2	The item is being extracted from the archive.
ptFreshen	3	The item is being freshened in the archive.
ptMove	4	The item is being moved within the archive.
ptReplace	5	The item is being replaced within the archive.

ErrorClass can be one of the following (TErrorClass):

Constant	Value	Description
eclAbbrevia	0	Exceptions that are defined by Abbrevia.
eclInOutError	1	File cannot be read from or written to.
eclFileError	2	General file error.
eclFileCreateError	3	File cannot be created.
eclFileOpenError	4	File cannot be opened.
eclOther	5	General catch-all error.

ErrorCode for ErrorClass eclAbbrevia can be one of the following (TErrorCode):

Constant	Value	Description
ecDuplicateName	0	[Add, Move, Replace] The file name already exists in the archive. You might want to display a warning to the user, but it can get ridiculous if the user is attempting to add the same large set of files to a directory twice. The file is not added or moved.
ecInvalidPassword	1	[Extract] The specified password is not valid for extracting the encrypted file. The file is not extracted.
ecNoSuchDirectory	2	[Extract] The destination directory does not exist. The file is not extracted.
ecSpannedItemNotFound	7	[Extract] The next archive was not found in a spanned archive set.
ecUnknownCompressionMethod	3	[Extract] The file was compressed with a compression method that is not supported by Abbrevia. The file is not extracted.

Constant	Value	Description
ecUserAbort	4	[Add, Extract, Freshen, Replace] The user aborted the process. The file is not processed.
ecZipBadCRC	5	[Extract] The extracted file's CRC doesn't match the stored CRC. The extracted file is deleted.
ecZipVersionNeeded	6	[Extract] The "Version needed to extract" for this file is not supported by this version of Abbrevia. This might occur in the future when a new version of PKZIP is available. The file is not extracted.

Abbrevia traps all exceptions that occur during Add, Extract, Freshen, Move, or Replace actions. The exception is translated to an ErrorClass and ErrorCode. The OnProcessItemFailure is called and you can display a customized error message. Abbrevia's exceptions all have an ErrorCode property which uniquely identifies them.

If you do not provide an event handler for OnProcessItemFailure, the user is not informed of the error. Abbrevia is designed to continue processing commands when an error is received without corrupting files or archives.

### OnRequestBlankDisk

event

*object*\_OnRequestBlankDisk(*Abort*)

The *object* is always an IZipKit.

Defines an event handler that is called when a blank, removable disk is needed for a spanned archive.

The OnRequestBlankDisk syntax has these parts:

Part	Type	Description
Abort	Boolean	Abort spanning by setting to True.

If an OnRequestBlankDisk event handler is not supplied, a dialog box with an OK button is displayed to prompt the user for a blank disk. If you want to add features such as formatting the disk, scanning the disk for errors, or allowing the user to verify the disk contents, you can do that in an OnRequestBlankDisk event handler.

**Note:** When used within Active Server Pages (ASP) all dialog boxes are suppressed and the event returns True, aborting the operation.

`object_OnRequestImage( ImageNumber, ImageName, Abort )`

The *object* is always an **IZipKit**.

☞ Defines an event handler to obtain a spanned archive file name.

The OnRequestImage syntax has these parts:

Part	Type	Description
ImageNumber	Integer	Identifies the file's sequence number within the spanned set.
ImageName	String	The file name of the requested image in a spanned archive.
Abort	Boolean	Abort reading the spanned archive by setting True.

OnRequestImage provides a mechanism to obtain a particular file name when working with a spanned set.

There are two conditions where this event will be fired—when saving an archive, and when extracting files from an archive.

When saving an archive, if the SpanningThreshold is reached, the OnRequestImage event is fired to obtain the file name (via ImageName) of the next file in the spanned set. If an event handler has not be defined, then the image file name is automatically generated by replacing the last two characters of the file extension with a sequence number, (e.g., FILES.ZIP, FILES.Z01, FILES.Z02, etc.).

When extracting an item from a spanned archive set, OnRequestImage is fired to obtain the archive image containing the item. If the item spans archive images, then OnRequestImage is fired as often as required until the item has been extracted. If an event handler is not assigned then an ecSpannedImageUnsupported exception will be raised when trying to extract from a spanned set.

## OnRequestLastDisk

event

*object*\_OnRequestLastDisk(*Abort*)

The *object* is always an **IZipKit**.

- ☞ Defines an event handler that is called when the last removable disk of a spanned archive is needed.

The OnRequestLastDisk syntax has these parts:

Part	Type	Description
Abort	Boolean	Abort reading the spanned archive by setting True.

The directory information for PKZIP files is stored on the last disk of a spanned archive. If you do not supply an OnRequestLastDisk event handler, a dialog box with an OK button is displayed to prompt the user for the last disk. If you want to add features such as allowing the user to verify the disk contents, you can do that in an OnRequestLastDisk event handler.

**Note:** When used within Active Server Pages (ASP) all dialog boxes are suppressed and the event returns True, aborting the operation.

## OnRequestNthDisk

event

*object*\_OnRequestNthDisk(*DiskNumber*, *Abort*)

The *object* is always an **IZipKit**.

- ☞ Defines an event handler that is called when a specific removable disk in the spanned archive is needed.

The OnRequestNthDisk syntax has these parts:

Part	Type	Description
DiskNumber	Integer	The disk number of the spanned archive needed.
Abort	Boolean	Abort reading the spanned archive by setting True.

If an OnRequestNthDisk event handler is not supplied, a dialog box with an OK button is displayed to prompt for the disk specified by DiskNumber. If you want to add features such as allowing the user to verify the disk contents, you can do that in an OnRequestNthDisk event handler.

**Note:** When used within Active Server Pages (ASP) all dialog boxes are suppressed and the event returns True, aborting the operation.

## OnSave

event

*object*.**OnSave()**

The *object* is always an **IZipKit**.

- ↳ Defines an event handler that is called immediately after the archive's contents are saved.

The OnSave event can be used along with the OnConfirmProcessDialogItem event to display status information for the user. For example, in the OnConfirmProcessItem event handler, the status could be set to "change pending", and in the OnSave event handler, set the status to "saved". This would be most useful when the AutoSave property is False.

## Password

read/write string property

*object*.**Password**

The *object* is always an **IZipKit**.

- ↳ The password used for encrypting or decrypting a file.

When a file is added to the archive, Password is used to encrypt it. If Password is empty, the file is not encrypted.

## PasswordRetries

read/write integer property

*object*.**PasswordRetries**

The *object* is always an **IZipKit**.

- ↳ Specifies the maximum number of passwords to try when attempting to extract an encrypted file.

When the number of retries is exhausted, an exception is raised. See the OnProcessItemFailure event on page 357 for a description of how exceptions that occur during the extract process are handled.

If PasswordRetries is 0 and Password is empty, encrypted files cannot be extracted.

## Replace

method

`object.Replace(FileName)`

The *object* is always an **IZipKit**.

- ↳ Replaces an item in the archive.

The Replace syntax has the following parts:

Part	Type	Description
FileMask	String	An expression identifying the file(s) to replace in the archive. Wild card characters ('*', '?') are allowed in the FileMask

Replace will update an item to the archive unilaterally, without checking for the most recent version first.

See also: Freshen

## Save

method

`object.Save`

The *object* is always an **IZipKit**.

- ↳ Updates the archive on disk.

If the archive has not been modified since it was last opened or saved, Save returns immediately without modifying the archive.

See also: AutoSave

## Spanned

read-only boolean property

`object.Spanned`

The *object* is always an **IZipKit**.

- ↳ Indicates whether the archive is part of a spanned set.

See also: SpanningThreshold

## SpanningThreshold

read/write integer property

*object.SpanningThreshold*

The *object* is always an **IZipKit**.

↳ Specifies the maximum archive image size.

By specifying SpanningThreshold you can restrict the size of the archive. If the archive size reaches SpanningThreshold when adding or freshening items, the archive is closed and a new archive file is created and the process continues onto the new, spanned archive. This process is repeated until the entire archive has been saved.

Setting SpanningThreshold to 0 specifies a single archive file.

Spanning archive files in this fashion requires that a file name is automatically generated by appending a two digit sequence number to the original archive file name, (e.g. Files.Zip, Files.Z01, Files.Z02, etc.). There is a limit of 99 extension files that can be auto-generated.

## Status

read-only integer property

*object.Status*

The *object* is always an **IZipKit**.

↳ Determines whether an operation is currently being performed on the archive.

Status is used internally by Abbrevia to prevent re-entrant operations, but could also be used to determine archive activity.

Status can return one of the following values form TArchiveStatus:

Constant	Value	Description
asInvalid	0	Set when the archive is being initialized.
asIdle	1	Initialization is finished and no other activity.
asBusy	2	An operation is modifying the archive.

*object.StoreOptions*

The *object* is always an **IZipKit**.

- ↳ Determines the options for archive add and freshen operations.

The following table shows the possible values that can be used (TStoreOptions):

Constant	Value	Description
soStripDrive	&H00000001	Drive letter information is removed from the stored file name.
soStripPath	&H00000002	All path information is removed from the stored file name.
soRemoveDots	&H00000004	All relative path information is removed from the stored file name. For example, if you call Add with a FileMask of "...\\TEST.TXT", the parent of the current BaseDirectory is searched for a file named "TEST.TXT." If the file is found, it is stored as "TEST.TXT."
soRecurse	&H00000008	Subdirectories of the search path are included in the search for files to add or freshen.
soFreshen	&H00000010	Existing items are freshened when they are added to the archive.
soReplace	&H00000020	Existing items are replaced when they are added to the archive.

The StoreOptions can be combined to include more than one option. For example, to recurse subdirectories and freshen the files that are added:

*object.StoreOptions = (soRecurse + soFreshen)*

---

<b>TagItems</b>	<b>method</b>
-----------------	---------------

---

*object*.**TagItems**(*FileMask*)

The *object* is always an **IZipKit**.

↳ Tags each archive item whose stored name matches FileMask.

Abbrevia allows you to perform operations on multiple files in an archive. A group of files can be extracted, freshened, or deleted by first tagging them and then performing the desired operation.

Items can be untagged using ClearTags or UnTagItems. Operations can be performed on the tagged items using the DeleteTaggedItems, ExtractTaggedItems, and FreshenTaggedItems methods.

---

<b>TempDirectory</b>	<b>read/write string property</b>
----------------------	-----------------------------------

---

*object*.**TempDirectory**

The *object* is always an **IZipKit**.

↳ Specifies a temporary directory to use during archive operations.

If this property is left empty, the system's temporary directory will be used.

---

<b>TestTaggedItems</b>	<b>method</b>
------------------------	---------------

---

*object*.**TestTaggedItems**( )

The *object* is always an **IZipKit**.

↳ Performs integrity test on tagged items.

TestTaggedItems verifies the integrity of each tagged item in the archive. Each Tagged item's central directory record, local file header and CRC are checked.

## UnTagItems

method

*object.UnTagItems(FileMask)*

The *object* is always an **IZipKit**.

- ↳ Sets the Tagged property to False for each archive item whose stored name matches FileMask.

The UnTagItems syntax has the following part:

Part	Type	Description
FileMask	String	An expression identifying the file(s) to untag in the archive. Wild card characters ('*', '?') are allowed in the FileMask.

## ZipFileComment

read/write string property

*object.ZipFileComment*

The *object* is always an **IZipKit**.

- ↳ The comment stored in the Zip archive.

A comment for the archive can be stored in PKZIP-compatible archives.



# Identifier Index

## A

AbExistingZipAssociation 300  
AbGetZipAssociation 300  
AbRegisterZipExtension 300  
AbXxx 64, 160  
Action 205, 322  
ActiveRow 177  
Add 211, 338  
AddFile 192  
AddFiles 85, 120, 137, 211  
AddFilesEx 87, 121, 139, 213  
AddFolder 192  
AddFromStream 87, 139, 213, 339  
AdditionalText 297  
AllocationSize 193  
Alternate 175  
AlternateText 175  
AmpleLength 304  
ArchiveName 214  
ArchiveProgressMeter 57, 140  
Attributes 140, 177, 190  
AutoSave 88, 141, 214, 340

## B

BaseDirectory 57, 142, 215, 340  
BeginUpdate 178

## C

CabComponent 188  
CabSize 110, 284  
Caption 297  
ChainLength 304  
Checksum 270  
ClearSelections 178

ClearTags 57, 142, 215, 341  
CloseArchive 57, 142  
cmXxx 324  
Colors 178  
ColWidths 179  
CompressedSize 205, 270, 323  
CompressionMethod 243, 261, 323  
CompressionMethodToUse 88, 143, 249, 341  
CompressionRatio 243, 325  
CompressionType 58, 121, 284, 341  
Count 58, 143, 179, 215, 342  
CRC32 205, 325  
CRC32St 325  
Create 194, 215, 238, 284, 305  
CreateFromStream 216  
CurrentCab 110, 285  
CurrentDirectory 194  
CurrentDisk 249

## D

Decoder 243  
DefaultColWidth 179  
DefaultRowHeight 179  
Deflate 302  
DeflateStream 311  
DeflationOption 89, 144, 244, 250, 326, 342  
Defrag 195  
Delete 216, 343  
DeleteAt 89, 144, 216, 343  
Deleted 175  
DeletedText 175  
DeleteFile 195  
DeleteFiles 89, 145, 216  
DeleteFilesEx 90, 145, 217  
DeleteFolder 195  
DeleteTaggedItems 90, 145, 217, 343

Destroy 196, 217  
 DevMajor 270  
 DevMinor 271  
 DictionarySize 244, 326  
 DirectoryEntries 196  
 DiskFileName 206, 261, 327  
 DiskNumberStart 244, 327  
 DiskPath 206, 261, 327  
 DisplayOptions 180  
 DoProgress 291  
 DOSMode 90, 146, 218, 344  
 doXxx 180, 342

**E**

EndUpdate 181  
 EntryType 190  
 Execute 294, 297  
 ExternalFileAttributes 206, 271, 328  
 Extract 218, 344  
 ExtractAt 76, 101, 114, 128, 146, 218, 345  
 ExtractFiles 76, 101, 114, 128, 147, 219  
 ExtractFilesEx 77, 102, 115, 129, 148, 220  
 ExtractHelper 250  
 ExtractItemData 238  
 ExtractOptions 77, 102, 115, 129, 148, 220,  
 346  
 ExtractTaggedItems 78, 104, 116, 130, 149,  
 222, 347  
 ExtractToStream 79, 104, 150, 222, 347  
 ExtractToStreamHelper 251  
 ExtraField 245, 262, 328  
 ExtraFlags 262

**F**

FileComment 245, 263, 328  
 FileName 58, 150, 196, 207, 263, 329, 347  
 FileSystem 263, 333  
 Find 348  
 FindFile 59, 150, 222

FindFirstItem 238  
 FindItem 59, 150, 222  
 FindNextItem 239  
 Flags 264  
 FolderCount 110, 285  
 FolderThreshold 122, 285  
 Freshen 223, 348  
 FreshenFiles 91, 151, 223  
 FreshenFilesEx 93, 152, 224  
 FreshenTaggedItems 93, 153, 225, 350

**G**

GeneralPurposeBitFlag 245  
 GroupID 272, 335  
 GroupName 273, 335

**H**

HasNext 110, 286  
 HasPrev 111, 286  
 HeaderCRC 265  
 HeaderRowHeight 181  
 Headings 181  
 Hierarchy 153

**I**

IGZipItem 332  
 Inflate 302  
 InflateStream 311  
 InsertFromStreamHelper 252  
 InsertHelper 253  
 InternalFileAttributes 245, 329  
 IsDirty 225  
 IsEncrypted 207, 265, 273, 329  
 IsExtraFieldPresent 265  
 IsFileCommentPresent 265  
 IsFileNamePresent 265  
 IsHeaderCRCPresent 266

- IsText 266  
 Item 350  
 ItemProgressMeter 59, 153  
 Items 72, 111, 154, 186, 188, 254, 286  
 IZipKit 337
- L**
- LastModFileDialog 207  
 LastModFileDialogTime 329  
 LastModFileTime 207  
 LastModified 190  
 LastModTimeAsDateTime 208  
 License 351  
 LinkFlag 274  
 LinkName 275  
 Load 226  
 LogFile 59, 154, 226, 305, 351  
 Logging 60, 154, 226, 352  
 LZXWindowSize 122
- M**
- MatchesDiskName 208  
 MatchesStoredName 208  
 MaxLazyLength 306  
 Mode 226, 286  
 Move 94, 155, 227
- N**
- NewCabinet 123, 287  
 NewFolder 123, 287
- O**
- OnAfterOpen 197  
 OnArchiveItemProgress 60, 155, 227, 352  
 OnArchiveProgress 61, 156, 228, 353
- P**
- OnBeforeClose 197  
 OnBeforeDirDelete 197  
 OnBeforeDirModified 198  
 OnBeforeFileDelete 198  
 OnBeforeFileModified 198  
 OnChange 61, 156, 182, 353  
 OnConfirmOverwrite 79, 104, 117, 131, 156, 228, 354  
 OnConfirmProcessItem 62, 157, 229, 354  
 OnConfirmSave 94, 157, 229, 355  
 OnGetStubExe 294  
 OnGetZipFile 294  
 OnLoad 62, 158, 230, 356  
 OnMouseWheel 158  
 OnNeedPassword 80, 105, 159, 255, 356  
 OnProcessItemFailure 63, 159, 230, 357  
 OnProgressStep 306  
 OnRequestBlankDisk 95, 161, 256, 359  
 OnRequestImage 65, 161, 232, 360  
 OnRequestLastDisk 72, 162, 256, 361  
 OnRequestNthDisk 73, 162, 257, 361  
 OnSave 95, 123, 162, 232, 362  
 OnSorted 182  
 OnWindowsDrop 163  
 Open 199  
 OpenArchive 65  
 OpenFile 199  
 Options 164, 307  
 Orientation 291

## R

ReadHeader 239  
ReadTail 239  
RelativeOffset 245  
RenameFile 200  
RenameFolder 200  
Replace 96, 166, 233, 363  
Reset 291

## S

Save 96, 166, 233, 363  
SeekItem 240  
SelCount 182  
SelectAll 183  
Selected 175, 183  
SelectedFolder 298  
SelectedText 175  
SelectedZipItem 166  
SelfExe 294  
SetID 111, 287  
ShannonFanoTreeCount 246  
Size 191  
SizeOnDisk 201  
smXxx 143  
SortAttributes 184  
soXxx 97, 168  
Spanned 233, 363  
SpanningThreshold 66, 167, 234, 364  
StartBlock 190  
Status 66, 168, 234, 364  
StoredPath 208, 266, 330  
StoreOptions 97, 124, 168, 235, 365  
StubExe 295  
Style 169

## T

TAbArchive 210  
TAbArchiveItem 204

TAbArchiveStreamHelper 237  
TAbBaseBrowser 56  
TAbBaseComponent 54  
TAbBaseViewer 176  
TAbCabArchive 282  
TAbCabBrowser 109  
TAbCabExtractor 113  
TAbCabItem 280  
TAbCabKit 126  
TAbCabView 187  
TAbColors 174  
TAbCompoundFile 191  
TAbCustomCabBrowser 108  
TAbCustomCabExtractor 112  
TAbCustomCabKit 125  
TAbCustomMakeCab 118  
TAbCustomUnZipper 74  
TAbCustomZipBrowser 70  
TAbCustomZipKit 98  
TAbCustomZipOutline 134  
TAbCustomZipper 82  
TAbDeflateHelper 303  
TAbDirDlg 296  
TAbGZipArchive 267  
TAbGZipItem 260  
TAbGZipStreamHelper 268  
TAbMakeCab 119  
TAbMakeSelfExe 293  
TAbMeter 290  
TAbTarArchive 277  
TAbTarItem 269  
TAbTarStreamHelper 278  
TAbZipArchive 247  
TAbZipBrowser 71  
TAbZipItem 242  
TAbZipKit 99  
TAbZipOutline 135  
TAbZipper 83  
TAbZipStreamHelper 259  
TAbZipView 185  
TAbZLibStream 279  
Tagged 209, 330

TagItems 66, 170, 236, 366  
TempDirectory 67, 170, 236, 366  
TestHelper 258  
TestTaggedItems 81, 106, 171, 236, 366

## U

UnCompressedSize 330  
UncompressedSize 209, 275  
UnTagItems 67, 171, 236, 367  
UnusedColor 292  
UsedColor 292  
UserID 275  
UserName 276

## V

vaXxx 177, 181  
Version 55, 171, 184, 201, 292  
VersionMadeBy 246, 331

VersionNeededToExtract 246, 331  
VolumeLabel 201

## W

WindowSize 309  
WriteArchiveHeader 240  
WriteArchiveItem 240  
WriteArchiveTail 241

## Z

zaXxx 140  
ZipComponent 186  
ZipFile 295  
ZipFileComment 73, 258, 367  
ZipfileComment 171  
ZipItem 321  
zooXxx 164  
zosXxx 169



---

# Subject Index

## A

Abbrevia Type Library 315  
accessing  
    archive comment 245  
    archive items 186  
    cabinet archive 188  
    comment 258, 263  
    ExtraField 245  
    PKZIP 328  
Active Server Pages 319  
adding  
    files to archive 85  
    folder to compound file 192  
    matching files except those excluded 87,  
        121, 139  
    matching files to archive 137, 171, 338  
    matching files to cabinet 120  
additional reading 24  
adjusting meter 291  
Allocation Area 191  
allocation size 190  
allocation unit, size in bytes 193  
archive  
    32-bit CRC 205  
    accessing comment 245  
    accessing items 186  
    adding files 85, 211  
    adding files except those excluded 87  
    adding items 211  
    adding matching files 137, 171, 338  
    adding matching files except those  
        excluded 139  
    changing 61, 88, 156, 182, 353  
    clearing tags 215  
    closing 57, 142  
    confirming overwrite 104, 156, 228  
    confirming process 62

archive (continued)  
    confirming save 94  
    creating 65, 215  
    creating item directly from stream 213,  
        216, 252  
    creating zip item directly from stream 87  
decoder 243  
decrypting 164  
default path 215, 340  
deleting files 89, 145, 216  
deleting files except those excluded 90,  
    145, 343  
deleting item 89, 144, 216  
deleting matching files except those  
    excluded 217  
deleting tagged items 90, 145, 217  
describing compression method 243  
destroying 217  
determining displayed attributes 140  
determining extract options 77, 102, 148,  
    346  
determining how nodes are drawn 164  
determining store options 97, 168, 235,  
    365  
DOS-compatible file names 90, 146, 218,  
    344  
dropping files from Windows 163  
encrypting 73, 164  
extract options 220  
extracting all matching files 76, 101  
extracting file to a stream 222  
extracting files 147, 250  
extracting files except those excluded 77,  
    102, 148, 344  
extracting item 218  
extracting item directly to stream 79, 104,  
    150, 347  
extracting item with known index 345

- archive (continued)  
extracting known item index 76, 101, 146  
extracting matching files 219  
extracting matching files except those excluded 220  
extracting tagged items 78, 104, 149, 222, 347  
extracting to stream 251  
file names 154  
finding file 150  
finding item 150  
finding matching files 348  
freshening 223  
freshening files 91, 151, 223  
freshening files except those excluded 93, 152, 348  
freshening tagged items 93, 225, 350  
inserting file 253  
item file names 72  
list of file names 254  
loading 62, 226  
loading contents 158, 230, 356  
long operations 60, 155  
marking items to be freshened 153  
maximum number of password retries 80, 165  
modifying 225  
moving 94  
name 150, 347  
needing password 80  
number of items in 58, 143  
obtaining file name 65, 161  
opening 65  
overwrite 79  
password 73, 159, 164  
performing integrity test on items 171  
performing process on item 157  
PKZIP 17  
prioritizing compression or speed 89, 144, 250, 342  
process failure 63, 159  
process pending 205  
archive (continued)  
process stepping 156, 353  
progress 61  
recording operations 60, 154, 226, 352  
renaming 94, 155  
renaming existing item 227  
replacing item 96, 166, 233, 363  
requesting blank disk 95, 161, 256, 359  
requesting disk 72  
requesting image 232  
requesting last disk 162, 361  
requesting last disk for spanned archive 256  
requesting specific disk 73, 162, 361  
requesting specific disk for spanned archive 257  
retrieving disk file name 58  
retrieving file index 59  
retrieving item index 59  
retrieving Zip comment 171  
rotating mouse wheel 158  
saving 95, 141, 157, 162, 166, 214, 229, 232, 233, 340, 355, 362  
saving updates on disk 96  
searching for file 222  
searching for item 222  
selecting compression method 88, 143, 249, 341  
self-extracting 17  
sequence number 249  
spanned file name 360  
spanned set 233, 363  
specifying compression type 58, 341  
specifying default path 142  
specifying maximum image size 66, 167, 234, 364  
specifying number of items 179  
specifying temporary directory 67, 170, 236  
specifying text file 351  
status 66, 168, 234  
stored file name 214

- archive (continued)
- storing file open mode 226
  - supplying file comment 73
  - tagging items 66, 170, 236, 366
  - temporary directory 366
  - untagging 57
  - untagging items 67, 142, 171, 236, 341, 367
  - updating on disk 363
  - User ID 336
  - User Name 336
  - verifying integrity of tagged item 106
  - verifying integrity of tagged items 81
- archive item, pending process 322
- attribute
- displaying 140
  - MS-DOS directory 328
  - sorting 184
- automation object
- Active Server Pages 319
  - defined 314
  - installing 317
  - license method required 351
  - Microsoft Transaction Server 318
  - object registration 317, 318
  - on-line help 317
  - system requirements 316
  - Type Library 315
  - unregistering object 318
- B**
- beginning viewer update 178
- building self-executable file 294
- C**
- cab component, specifying 188
- CAB file support 107
- cabinet
- adding matching files 120
  - adding matching files except those excluded 121
  - application-defined set ID number 111
  - confirming overwrite 131
  - constructing archive 284
  - creating new 287
  - determining adding and freshening options 124
  - determining archive extract options 129
  - extracting all files except those excluded 115, 129
  - extracting all matching files 128
  - extracting known item index 114, 128
  - extracting matching files 114
  - extracting tagged items 116, 130
  - identifying 287
  - indicating if file is a continuation 281
  - indicating read/write status 286
  - indicating whether chained to next 110, 286
  - indicating whether spans from previous 111, 286
  - list of file names 286
  - LZX compression engine 122
  - maximum folder size 122, 285
  - number in a set 110
  - number of folders in 110
  - retrieving current 285
  - retrieving number of folders 285
  - retrieving size in bytes 284
  - saving 123
  - setting ID number 111
  - setting window size 122
  - size 110
  - specifying compression type 121
  - starting new disk 123
  - starting new folder 123

- cabinet archive, accessing 188
- cabinet file compression 21
- CABINET.DLL 107
- changing
  - archive 61, 88, 156, 182, 353
  - heading text 181
- checksum value 270
- clearing
  - tag 215
- closing
  - archive 57, 142
  - compound file 197
- color
  - alternate font coloring 175
  - background alternate row 175
  - deleted items font color 175
  - items to be deleted 175
  - of unused meter 292
  - of used meter 292
  - selected items 175
  - selected items font 175
  - set 178
- column
  - changing heading text 181
  - determining width 179
  - determining width within viewer 179
  - specifying header button height 181
- comment
  - accessing 258, 263
  - for ZIP file 367
- Component Object Model
  - defined 314
  - See Automation Object 314
- compound file
  - adding folder 192
  - adding single file 192
  - closing 197
  - creating 194
  - current directory 194
  - defragmenting 195
  - deleting 195, 198
  - deleting folder 195
- compound file (continued)
  - disk size 201
  - freeing memory 196
  - internal structure 189
  - name 196
  - number of directory entries 196
  - opening 194, 197, 199
  - populating tree view 200
  - version of engine 201
  - volume label 201
- compressed file
  - last date and time modified 329
- compressing
  - source stream 302
  - stream contents 311
- compression
  - cabinet file 21
  - capabilities 17
  - classes 203
  - deflating 16
  - encryption 23
  - GZip 20
  - imploding 16
  - method 323
  - method used by GZip 261
  - overview 13
  - ratio 325
  - reducing 15
  - shrinking 15
  - spanning 23
  - specifying type 58, 284
  - storing 15
  - stream 18
  - TAR 20
  - techniques 15
  - ZLib capabilities 19
- compression method, selecting 143, 341
- confirming
  - archive overwrite 104, 156
  - archive save 94
  - cabinet overwrite 131

- confirming (continued)
    - file overwrite 228, 354
    - overwrite 354
  - creating
    - archive 65, 215
    - archive from stream 216
    - compound file 194
    - default helper object 305
    - stream 238
  - zip item directly from stream 87, 139
  - current version of Abbrevia 292
- D**
- decompressing
    - source stream 302
    - stream contents 311
  - decrypting
    - data 308
    - file 164, 255, 257, 362
  - defining
    - dictionary size 244
    - file type 245
    - length of matching string 304
    - LZ77 sliding window size 309
    - maximum length of matching string 306
    - options to compress data 307
    - PKZIP deflation option 309
  - Deflate algorithm, selecting parameters 326
  - defragmenting, compound file 195
  - deleting
    - archive 216
    - archive files except those excluded 145
    - compound file 195
    - directory 197
    - files 217
    - files except those excluded 90, 343
    - files from archive 89, 145, 216
  - deleting (continued)
    - folder from compound file 195
    - item 216
    - item at specified index 343
    - item from archive 89, 144
    - tagged items 90, 217, 343
    - tagged items from archive 145
  - deployment
    - COM Object license 351
    - DEPLOY.HLP 351
  - destroying, archive 217
  - determining
    - archive extract options 77, 102, 129, 148, 346
    - archive store options 97, 168, 235, 365
    - cabinet adding and freshening options 124
    - column width 179
    - displayed archive attributes 140
    - how nodes are drawn 164
    - if meter is horizontal or vertical 291
    - viewer behavior 180
    - viewer column width 179
    - viewer row height 179
  - dialog box
    - displaying 297
    - specifying text in title bar 297
    - specifying text that appears 297
  - dictionary size 326
  - directory
    - compound file 194
    - deleting 197
    - modifying 198
  - disk size of compound file 201
  - displaying
    - current version of Abbrevia 55
    - dialog box 297
    - item attributes 177
    - items in hierarchy 153
    - tree 169

**E**

encrypting  
 archive 73  
 data 308  
 file 164, 257, 362  
 item 265, 273  
 encryption 17  
 executable stub  
   file name 295  
   name of 294  
 extracting  
   all files from cabinet except those excluded  
     129  
   all matching archive files 76, 101  
   all matching files from cabinet 128  
   archive item directly to stream 150, 347  
   archive item to stream 79, 251  
   data from stream 238  
   file to a stream 222  
   files except those excluded 77, 102  
   files from archive 147, 250  
   files from archive except those excluded  
     148, 344  
   files from cabinet except those excluded  
     115  
   item directly to stream 104  
   item from archive 218  
   item from archive with known index 345  
   known cabinet item index 128  
   known item index 76, 101  
   known item index from archive 146  
   known item index from cabinet 114  
   matching files from archive 219  
   matching files from cabinet 114  
   tagged items from archive 78, 104, 149,  
     222, 347  
   tagged items from cabinet 130

**F**

file  
   adding single to compound 192  
   adding to archive 85, 139, 338  
   adding to cabinet 121  
   attributes 206  
   building self-executable 294  
   complete name stored on disk 327  
   confirming overwrite 228  
   creating output 218  
   decrypting 164  
   defining dictionary size 244  
   defining type 245, 329  
   deleting 216, 217, 343  
   deleting from archive 89, 145  
   encrypting 164  
   encryption 17  
   extracting 219, 220, 344  
   extracting from archive 76, 147, 148, 250  
   extracting from cabinet 129  
   extracting matching from archive 101  
   extracting matching from cabinet 114  
   finding 150, 348  
   freshening 91, 151, 152, 223, 348  
   freshening except those excluded 93  
   indicating system where created 263  
   inserting into archive 253  
   internal structure 189  
   last date and time modified 208  
   last date modified 207  
   last time modified 207  
   matching name 208  
   modifying 198  
   name 206, 207, 261, 263, 329  
   number of Shannon-Fano trees used 246  
   obtaining name 161  
   offset in bytes 245

file (continued)

- opening 199
- optional comment 328
- path 206, 208, 261
- path information 327
- renaming 200
- retrieving path stored in header 266
- size of compressed 205, 270
- size of uncompressed 209, 275
- specifying 154, 226
- specifying system entity 274
- system 333
- uncompressed size in bytes 330
- writing to 199

File Allocation Table (FAT) 191

file header path 330

finding

- archive file 150
- archive item 150
- first archived item in stream 238
- matching archive files 348
- subsequent archived item in stream 239

folder

- creating new 287
- path 298
- renaming 200

freeing compound file memory 196

freshening

- archive 223
- archive files 91, 151
- archive files except those excluded 152, 348
- file 223
- files except those excluding 93
- tagged items 93, 225, 350

## G

glyph

- displaying in attribute 165
- displaying in file 165

GZip

- compression 20
- compression method 261
- flags 264
- header information 265
- specifying extra field data 262

## H

hash chain, defining length of to search 304

header

- reading data 239
- writing 240

helper object, creating default 305

hierarchy, displaying items 153

## I

index

- extracting item from 76
- extracting known item 101
- retrieving archive file 59
- retrieving archive item 59

indicating

- cabinet read/write status 286
- whether cabinet chained to next 110
- whether cabinet spans from previous 111

inheritance 9

inserting file into archive 253

installing 4

- Automation Object 317

Internet development 313

item

- accessing 186
- adding directly from stream 339
- compression ratio 243
- creating 339
- deleting 216, 217
- deleting at specified item index 343
- deleting from archive 89, 144, 145
- deleting tagged 90, 343

item (continued)  
    displaying attributes 177  
    displaying in hierarchy 153  
    encrypting 207, 265, 273, 329  
    external file system attributes 271  
    extracting 222  
        extracting directly to stream 104, 150, 347  
    extracting from archive 149, 218, 345, 347  
    extracting from cabinet 130  
    extracting tagged from archive 78, 104  
    extracting to stream 79  
failure 357  
file names 72  
finding 150  
freshening 93, 225, 350  
interface 350  
marking to be freshened 153  
number available 342  
number in archive 143  
number of 215  
number selected 182  
performing integrity test 171, 236, 366  
renaming 155, 227  
replacing 96, 166, 233  
retrieving reference 166  
selecting 209  
selecting all in viewer 183  
selection status 183  
sorting attributes 184  
specifying number 179  
table row number 177  
tagging 170, 236, 330, 366  
testing 258  
unselecting 178  
untagging 142, 171, 236, 341, 367  
verifying integrity 81, 106  
writing data 240  
writing trailing data 241

IZipKit  
    license verification 351  
    status 364  
    zip file comment 367

**L**

license key, verifying 351  
loading  
    archive 226  
        archive contents 62, 158, 230, 356  
    locating archived item in stream 240  
log file, name of 305  
long archive operations 155  
LZ77, defining sliding window size 309  
LZX compression engine, setting window size 122

**M**

marking, archive items to be freshened 153  
matching string  
    defining length of 304  
    defining maximum length 306  
maximum cabinet folder size 285  
meter  
    adjusting 291  
    color of unused portion 292  
    color of used portion 292  
    determining if horizontal or vertical 291  
    resetting to zero 291  
method, selecting 88  
Microsoft Transaction Server 318  
miscellaneous routines overview 289  
modifying  
    archive 225  
    directory 198  
    file 198

mouse wheel, rotating 158  
 moving  
   archive 94  
 MS-DOS directory attribute 328

## N

name  
   of archive 150, 347  
   of compound file 196  
   of log file 305  
 naming conventions 6  
 number  
   of cabinets in set 110  
   of folders in cabinet 110  
   of items selected 182

## O

obtaining file name 161  
 on-line help 6  
 opening  
   archive 65  
   compound file 194, 197, 199  
 overwrite  
   confirming 354  
 overwriting  
   archive 79

## P

passphrase  
   decrypting data 308  
   encrypting data 308  
 password  
   archive 73, 159  
   decrypting files 255, 257, 329, 356, 362  
   encrypting files 257, 329, 362  
 maximum number of retries 80, 105, 165,  
   258, 362

password (continued)  
   needing 105  
   needing to decrypt file 80  
 path  
   of folder 298  
   specifying default 142  
   specifying default archive 57  
 performing  
   integrity test on items 171  
   integrity test on tagged items 236, 366  
 PKZIP 17  
   accessing 328  
   defining deflation option 309  
   spanning 17  
   version 246, 331  
   version needed to extract files 246  
   version number required to extract file 331  
 populating compound file tree view 200  
 prioritizing  
   archive compression or speed 89  
   compression or speed 144, 342  
 progress, archive 61

## R

recording  
   archive operations 60, 154, 226, 352  
 re-enabling screen repainting 181  
 registering ZIP extension 300  
 renaming  
   archive 94  
   archive item 155, 227  
   file 200  
   folder 200  
 replacing  
   archive item 166  
   item in archive 96, 233, 363  
 requesting  
   archive image 232  
   blank disk for spanned archive 95, 161,  
   256, 359

- requesting (continued)  
    last disk of spanned archive 72, 162,  
        256, 361  
    specific disk for spanned archive 162,  
        257, 361  
    specific spanned archive disk 73  
resetting meter to zero 291  
retrieving  
    archive disk file name 58  
    archive file index 59  
    archive item index 59  
    cabinet size in bytes 284  
    file path stored in header 266  
    item reference 166  
    number of folders in cabinet 285  
    number of items in archive 58  
    Zip comment 171  
Root Directory block 190  
row  
    determining height 179  
    sorting 182  
Run-Time License 351
- S**
- save, confirming 94  
saving  
    archive 95, 141, 157, 162, 166, 229, 232,  
        233, 340, 355, 362  
    archive updates on disk 96  
    cabinet 123  
screen, re-enabling repainting 181  
search, stopping 304  
searching  
    archive for file 222  
    archive for item 222  
selecting  
    all items in viewer 183  
    archive compression method 143, 341  
    archive method to use 88
- selecting (continued)  
    compression method 249  
    parameters for Deflate algorithm 326  
selection status of item 183  
self-delegation 11  
self-extracting executable file name 294  
sequence diagram 11  
sequence number 249  
set of row colors 178  
setting  
    window size for LZX compression engine  
        122  
Shannon-Fano trees 246  
signature 189  
size  
    of cabinet 110  
    of dictionary 326  
sorting  
    item attributes 184  
    rows 182  
source stream  
    compressing 302  
    decompressing 302  
spanned set, first image number 327  
specifying  
    archive compression type 58, 341  
    archive text file 351  
    cab component 188  
    cabinet compression type 121  
    column header height 181  
    compression type 284  
    default archive path 57, 142  
    dialog box title bar text 297  
    file 226  
    file to use 154  
    maximum archive image size 66, 167, 234,  
        364  
    number of items in archive 179  
    synchronized meter 57, 59, 140, 153  
    temporary archive directory 67, 170, 236  
    text file 59

specifying (continued)  
     text that appears on the dialog box 297  
     type of file system entity 274  
     zip component 186  
 starting  
     new cabinet disk 123  
     new cabinet folder 123  
 status  
     archive 66, 168, 234  
 stopping a search 304  
 stream  
     compressing contents 311  
     compression 18  
     creating 238  
     creating zip item from 87  
     decompressing contents 311  
     extracting data from 238  
     finding first archived item 238  
     finding subsequent archived item 239  
     locating archived item 240  
     supplying archive file comment 73  
 synchronized meter, specifying 57, 59, 140, 153  
 System Block 189  
 system requirements 3

**T**

tagging  
     archive items 66, 170, 236, 366  
     item 330  
 tail, reading 239  
 TAR compression 20  
 Tar item, checksum value 270  
 technical support 7  
 temporary archive directory 366  
 text file, specifying 59  
 tree, displaying 169

**U**

UML 8  
 Unified Modeling Language 8  
 Unix file system  
     indicating Group ID 272  
     indicating Group Name 273  
     indicating User ID 275  
     indicating User Name 276  
     providing Major Device ID 270  
     providing Minor Device ID 271  
 unselecting items 178  
 untagging  
     archive 57  
     archive items 67, 171, 236, 341, 367  
     items in archive 142  
 updating archive on disk 363  
 updating flag 190  
 User ID 336  
 User Name 336

**V**

verifying  
     integrity of tagged item 106  
     integrity of tagged items 81  
     license key 351  
 version  
     current 292  
     defined 190  
     displaying 55  
     of compound file engine 201  
     of PKZIP 246  
     of PKZIP needed to extract files 246  
     showing current 184  
 viewer  
     beginning update 178  
     determining behavior 180

viewer (continued)  
determining column width 179  
determining height of rows 179  
selecting all items 183  
visibility characters 9  
volume label 190  
volume label of compound file 201

## Z

zip archive, file name 294, 295  
Zip comment, retrieving 171

zip component, specifying 186  
ZIP extension, registering 300  
zip file  
application association 300  
association 300  
zip item  
32-bit CRC 325  
32-bit CRCSt 325  
compressed file size 323  
creating directly from stream 139  
ZLib compression 19



The TurboPower family of tools—  
Winners of 6 *Delphi Informant* Readers' Choice Awards  
for 2001! Company of the Year in 2000 and 2001.

**F**or over fifteen years you've depended on TurboPower to provide the best tools and libraries for your development tasks. Now try LockBox 2 and XMLPartner—two of TurboPower's best selling products risk free. Both are fully compatible with Borland Kylix, Delphi, and C++Builder, and are backed with our expert support and 60-day money back guarantee.

DEVELOP, DEBUG, OPTIMIZE  
FROM START TO FINISH, TURBOPOWER  
HELPS YOU BUILD YOUR BEST



Try the full range of  
TurboPower products.  
Download free Trial-Run Editions  
from our Web site.

[www.turbopower.com](http://www.turbopower.com)

## LOCKBOX 2™

*LockBox 2 is the gateway to powerful standard cryptographic algorithms like RSA, DES, and Rijndael (the new AES). You can make use of LockBox's implementations of the secure hashing algorithms MD5 and SHA-1, or even add digital signature capabilities using the DSA and RSASSA algorithms.*

## XMLPARTNER PROFESSIONAL™

*TurboPower's newest cross-platform toolkit for XML provides all the reading, writing, and editing power of our entry-level product, XMLPartner, then adds advanced manipulation, transformation, and presentation components—all in a single package! Our new XSL Processor with XPath support, filter set, and EXMLPro utility make it easy to add XML capabilities to your applications.*

Abbrevia 3 requires Microsoft Windows (9X, Me, NT or 2000), or Linux operating systems, and Borland Delphi 3 and above, C++Builder 3 and above, or Borland Kylix

TurboPower Software Company  
©2001, TurboPower Software Co.

 **TURBOPOWER®**  
Software Company